# Package 'familiar'

May 22, 2026

**Title** End-to-End Automated Machine Learning and Model Evaluation

**Version** 2.0.1

**Description** Single unified interface for end-to-end modelling of regression, categorical and time-to-event (survival) outcomes. Models created using familiar are self-containing, and their use does not require additional information such as baseline survival, feature clustering, or feature transformation and normalisation parameters. Model performance, calibration, risk group stratification, (permutation) variable importance, individual conditional expectation, partial dependence, and more, are assessed automatically as part of the evaluation process and exported in tabular format and plotted, and may also be computed manually using export and plot functions. Where possible, metrics and values obtained during the evaluation process come with confidence intervals.

**URL** https://github.com/oncoray/familiar

**BugReports** https://github.com/oncoray/familiar/issues

**Depends** R (>= 4.0.0)

**License** EUPL

**Encoding** UTF-8

**VignetteBuilder** knitr

**Imports** data.table, methods, rlang (>= 1.0.0), rstream, survival

**Suggests** BART, callr (>= 3.4.3), cluster, COREstats, coro, dynamicTreeCut, e1071 (>= 1.7.5), fastcluster, fastglm, ggplot2 (>= 4.0.0), glmnet, gtable, harmonicmeanp, isotree (>= 0.6.0), knitr, labeling, laGP, maxstat, microbenchmark, nnet, paletteer, power.transform, praznik, proxy, randomForestSRC, ranger, rmarkdown, scales, testthat (>= 3.0.0), xml2, xgboost (>= 3.0.0)

**Collate** 'FamiliarS4Classes.R' 'FamiliarS4Generics.R' 'BatchNormalisation.R' 'BootstrapConfidenceInterval.R' 'CheckArguments.R' 'CheckHyperparameters.R' 'CheckPackages.R' 'ClassBalance.R' 'ClusteringMethod.R' 'Clustering.R' 'ClusterRepresentation.R' 'Normalisation.R'

'CombatNormalisation.R' 'LearnerS4Naive.R' 'DataObject.R'
'DataParameterChecks.R' 'DataPreProcessing.R'
'DataProcessing.R' 'DataServerBackend.R' 'ErrorMessages.R'
'ExperimentData.R' 'ExperimentSetup.R' 'Familiar.R'
'FamiliarCollection.R' 'FamiliarCollectionExport.R'
'FamiliarData.R' 'FamiliarDataComputation.R'
'FamiliarDataComputationPredictionData.R' 'PredictionTable.R'
'FamiliarDataComputationAUCCurves.R'
'FamiliarDataComputationCalibrationData.R'
'FamiliarDataComputationCalibrationInfo.R'
'FamiliarDataComputationConfusionMatrix.R'
'FamiliarDataComputationDecisionCurveAnalysis.R'
'FamiliarDataComputationFeatureExpression.R'
'FamiliarDataComputationFeatureSimilarity.R'
'FamiliarDataComputationHyperparameters.R'
'FamiliarDataComputationICE.R'
'FamiliarDataComputationModelPerformance.R'
'FamiliarDataComputationPermutationVimp.R'
'FamiliarDataComputationRiskStratificationData.R'
'FamiliarDataComputationRiskStratificationInfo.R'
'FamiliarDataComputationSHAP.R'
'FamiliarDataComputationSampleSimilarity.R'
'FamiliarDataComputationUnivariateAnalysis.R'
'FamiliarDataComputationUtilities.R'
'FamiliarDataComputationVimp.R' 'FamiliarDataElement.R'
'FamiliarEnsemble.R' 'FamiliarHyperparameterLearner.R'
'FamiliarModel.R' 'FamiliarNoveltyDetector.R'
'FamiliarObjectConversion.R' 'Transformation.R'
'FamiliarObjectUpdate.R' 'FamiliarSharedS4Methods.R'
'FamiliarVimpMethod.R' 'FeatureInfo.R'
'FeatureInfoParameters.R' 'FunctionWrapperUtilities.R'
'HyperparameterOptimisation.R'
'HyperparameterOptimisationMetaLearners.R'
'HyperparameterOptimisationUtilities.R'
'HyperparameterS4BayesianAdditiveRegressionTrees.R'
'HyperparameterS4GaussianProcess.R'
'HyperparameterS4RandomSearch.R' 'HyperparameterS4Ranger.R'
'Imputation.R' 'Iterations.R' 'LearnerMain.R'
'LearnerRecalibration.R' 'LearnerRiskStratification.R'
'LearnerS4Cox.R' 'LearnerS4GLM.R' 'LearnerS4GLMnet.R'
'LearnerS4KNN.R' 'LearnerS4MBoost.R' 'LearnerS4NaiveBayes.R'
'LearnerS4RFSRC.R' 'LearnerS4Ranger.R' 'LearnerS4SVM.R'
'LearnerS4SurvivalRegression.R' 'LearnerS4XGBoost.R'
'LearnerSurvivalProbability.R' 'Logger.R' 'MetricS4.R'
'MetricS4AUC.R' 'MetricS4Brier.R' 'MetricS4ConcordanceIndex.R'
'MetricS4ConfusionMatrixMetrics.R' 'MetricS4Regression.R'
'NoveltyDetectorMain.R' 'NoveltyDetectorS4IsolationTree.R'
'NoveltyDetectorS4NoneNoveltyDetector.R' 'OutcomeInfo.R'

'PairwiseSimilarity.R' 'ParallelFunctions.R' 'ParseData.R'
'ParseSettings.R' 'PlotAUCcurves.R' 'PlotAll.R'
'PlotCalibration.R' 'PlotColours.R' 'PlotConfusionMatrix.R'
'PlotDecisionCurves.R' 'PlotFamiliarPlot.R'
'PlotFeatureRanking.R' 'PlotFeatureSimilarity.R' 'PlotGTable.R'
'PlotICE.R' 'PlotInputArguments.R' 'PlotKaplanMeier.R'
'PlotModelPerformance.R' 'PlotPermutationVariableImportance.R'
'PlotSampleClustering.R' 'PlotShapDependence.R'
'PlotShapForce.R' 'PlotShapSummary.R' 'PlotShapWaterfall.R'
'PlotUnivariateImportance.R' 'PlotUtilities.R'
'PredictS4Methods.R' 'ProcessTimeUtilities.R' 'Random.R'
'RankBordaAggregation.R' 'RankMain.R' 'RankSimpleAggregation.R'
'RankStabilityAggregation.R' 'SocketServer.R'
'StringUtilities.R' 'TaskEvaluate.R' 'TaskFeatureInfo.R'
'TaskLearn.R' 'TaskLearnerHyperparameters.R' 'TaskMain.R'
'TaskNoveltyDetector.R' 'TaskNoveltyDetectorHyperparameters.R'
'TaskVimp.R' 'TaskVimpHyperparameters.R' 'TestDataCreators.R'
'TestFeatureInfo.R' 'TestFunctions.R' 'TestTrain.R'
'TestTrainNovelty.R' 'TestVimp.R' 'TrimUtilities.R'
'Utilities.R' 'UtilitiesS4.R' 'VimpMain.R'
'VimpS4Concordance.R' 'VimpS4CoreLearn.R' 'VimpS4Correlation.R'
'VimpS4MutualInformation.R' 'VimpS4OtherMethods.R'
'VimpS4Regression.R' 'VimpTable.R' 'aaa.R'

**Config/testthat/parallel** true

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** Alex Zwanenburg [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0342-9545>),
Steffen Löck [aut],
German Cancer Research Center (DKFZ) [cph],
Technische Universität Dresden [cph]

**Maintainer** Alex Zwanenburg <alexander.zwanenburg@nct-dresden.de>

**Repository** CRAN

**Date/Publication** 2026-05-22 18:50:02 UTC

# Contents

---

`aggregate_vimp_table` *Aggregate variable importance from multiple variable importance objects.*

---

## Description

This methods aggregates variable importance from one or more `vimpTable` objects.

## Usage

```
aggregate_vimp_table(x, aggregation_method, rank_threshold = NULL, ...)

## S4 method for signature 'list'
aggregate_vimp_table(x, aggregation_method, rank_threshold = NULL, ...)

## S4 method for signature 'character'
aggregate_vimp_table(x, aggregation_method, rank_threshold = NULL, ...)

## S4 method for signature 'vimpTable'
aggregate_vimp_table(x, aggregation_method, rank_threshold = NULL, ...)

## S4 method for signature 'NULL'
aggregate_vimp_table(x, aggregation_method, rank_threshold = NULL, ...)

## S4 method for signature 'experimentData'
aggregate_vimp_table(x, aggregation_method, rank_threshold = NULL, ...)
```

## Arguments

| | |
|---|---|
| x | Variable importance (vimpTable) object, a list thereof, or one or more paths to these objects. |
| aggregation_method | Method used to aggregate variable importance. The available methods are described in the *feature selection methods* vignette. |
| rank_threshold | Rank threshold used within several aggregation methods. See the *feature selection methods* vignette for more details. |
| ... | unused parameters. |

## Value

A vimpTable object with aggregated variable importance data.

---

# as_data_object *Creates a valid data object from input data.*

---

## Description

Creates a dataObject object from input data. Input data can be a data.frame or data.table, a path to such tables on a local or network drive, or a path to tabular data that may be converted to these formats.

In addition, a familiarEnsemble or familiarModel object can be passed along to check whether the data are formatted correctly, e.g. by checking the levels of categorical features, whether all expected columns are present, etc.

## Usage

```
as_data_object(data, ...)

## S4 method for signature 'dataObject'
as_data_object(data, object = NULL, ...)

## S4 method for signature 'data.table'
as_data_object(
  data,
  object = NULL,
  batch_id_column = waiver(),
  sample_id_column = waiver(),
  series_id_column = waiver(),
  development_batch_id = waiver(),
  validation_batch_id = waiver(),
  outcome_name = waiver(),
  outcome_column = waiver(),
  outcome_type = waiver(),
  event_indicator = waiver(),
  censoring_indicator = waiver(),
  competing_risk_indicator = waiver(),
  class_levels = waiver(),
  exclude_features = waiver(),
  include_features = waiver(),
  reference_method = waiver(),
  check_stringency = "strict",
  .no_features_required = FALSE,
  ...
)

## S4 method for signature 'ANY'
as_data_object(
  data,
  object = NULL,
  sample_id_column = waiver(),
  batch_id_column = waiver(),
  series_id_column = waiver(),
  ...
)
```

## Arguments

| | |
|---|---|
| data | A data.frame or data.table, a path to such tables on a local or network drive, or a path to tabular data that may be converted to these formats. |
| ... | Unused arguments. |
| object | A familiarEnsemble or familiarModel object that is used to check consistency of these objects. |

batch_id_column

(**recommended**) Name of the column containing batch or cohort identifiers. This parameter is required if more than one dataset is provided, or if external validation is performed.

In familiar any row of data is organised by four identifiers:

- The batch identifier batch_id_column: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.
- The sample identifier sample_id_column: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.
- The series identifier series_id_column: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.
- The repetition identifier: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries for the same series of the same sample in the same batch that share the same outcome are encountered.

sample_id_column

(**recommended**) Name of the column containing sample or subject identifiers. See batch_id_column above for more details.

If unset, every row will be identified as a single sample.

series_id_column

(*optional*) Name of the column containing series identifiers, which distinguish between measurements that are part of a series for a single sample. See batch_id_column above for more details.

If unset, rows which share the same batch and sample identifiers but have a different outcome are assigned unique series identifiers.

development_batch_id

(*optional*) One or more batch or cohort identifiers to constitute data sets for development. Defaults to all, or all minus the identifiers in validation_batch_id for external validation. Required if external validation is performed and validation_batch_id is not provided.

validation_batch_id

(*optional*) One or more batch or cohort identifiers to constitute data sets for external validation. Defaults to all data sets except those in development_batch_id for external validation, or none if not. Required if development_batch_id is not provided.

outcome_name

(*optional*) Name of the modelled outcome. This name will be used in figures created by familiar.

If not set, the column name in outcome_column will be used for binomial, multinomial, and continuous outcomes. For other outcomes (survival and competing_risk) no default is used.

`outcome_column` (**recommended**) Name of the column containing the outcome of interest. May be identified from a formula, if a formula is provided as an argument. Otherwise an error is raised. Note that `survival` and `competing_risk` outcome type outcomes require two columns that indicate the time-to-event or the time of last follow-up and the event status.

`outcome_type` (**recommended**) Type of outcome found in the outcome column. The outcome type determines many aspects of the overall process, e.g. the available variable importance methods and learners, but also the type of assessments that can be conducted to evaluate the resulting models. Implemented outcome types are:

- `binomial`: categorical outcome with 2 levels.
- `multinomial`: categorical outcome with 2 or more levels.
- `continuous`: general continuous numeric outcomes.
- `survival`: survival outcome for time-to-event data.

If not provided, the algorithm will attempt to obtain outcome_type from contents of the outcome column. This may lead to unexpected results, and we therefore advise to provide this information manually.

Note that `competing_risk` survival analysis are not fully supported, and is currently not a valid choice for `outcome_type`. The `count` outcome type was deprecated in version 2.0.0, and superseded by `continuous`.

`event_indicator`

(**recommended**) Indicator for events in `survival` and `competing_risk` analyses. `familiar` will automatically recognise `1`, `true`, `t`, `y` and `yes` as event indicators, including different capitalisations. If this parameter is set, it replaces the default values.

`censoring_indicator`

(**recommended**) Indicator for right-censoring in `survival` and `competing_risk` analyses. `familiar` will automatically recognise `0`, `false`, `f`, `n`, `no` as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values.

`competing_risk_indicator`

(**recommended**) Indicator for competing risks in `competing_risk` analyses. There are no default values, and if unset, all values other than those specified by the `event_indicator` and `censoring_indicator` parameters are considered to indicate competing risks.

`class_levels` (*optional*) Class levels for `binomial` or `multinomial` outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column.

`exclude_features`

(*optional*) Feature columns that will be removed from the data set. Cannot overlap with features in `signature`, `novelty_features` or `include_features`.

`include_features`

(*optional*) Feature columns that are specifically included in the data set. By default all features are included. Cannot overlap with `exclude_features`, but may overlap `signature`. Features in `signature` and `novelty_features` are always included. If both `exclude_features` and `include_features` are provided, `include_features` takes precedence, provided that there is no overlap between the two.

`reference_method`
: (*optional*) Method used to set reference levels for categorical features. There are several options:
  - `auto` (default): Categorical features that are not explicitly set by the user, i.e. columns containing boolean values or characters, use the most frequent level as reference. Categorical features that are explicitly set, i.e. as factors, are used as is.
  - `always`: Both automatically detected and user-specified categorical features have the reference level set to the most frequent level. Ordinal features are not altered, but are used as is.
  - `never`: User-specified categorical features are used as is. Automatically detected categorical features are simply sorted, and the first level is then used as the reference level. This was the behaviour prior to familiar version 1.3.0.

`check_stringency`
: Specifies stringency of various checks. This is mostly:
  - `strict`: default value used for `summon_familiar`. Thoroughly checks input data. Used internally for checking development data.
  - `external_warn`: value used for `extract_data` and related methods. Less stringent checks, but will warn for possible issues. Used internally for checking data for evaluation and explanation.
  - `external`: value used for external methods such as `predict`. Less stringent checks, particularly for identifier and outcome columns, which may be completely absent. Used internally for `predict`.

`.no_features_required`
: Internal flag to signify that data without features is allowed. Default: FALSE (most processing steps require features).

## Details

You can specify settings for your data manually, e.g. the column for sample identifiers (`sample_id_column`). This prevents you from having to change the column name externally. In the case you provide a `familiarModel` or `familiarEnsemble` for the `object` argument, any parameters you provide take precedence over parameters specified by the object.

## Value

A `dataObject` object.

---

# `as_familiar_collection` *Conversion to familiarCollection object.*

---

## Description

Creates a `familiarCollection` objects from `familiarData`, `familiarEnsemble` or `familiarModel` objects.

## Usage

```
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)

## S4 method for signature 'familiarCollection'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)

## S4 method for signature 'familiarData'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)

## S4 method for signature 'familiarEnsemble'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)

## S4 method for signature 'familiarModel'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)

## S4 method for signature 'familiarDataElementPredictionTable'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)
```

```
## S4 method for signature 'dataObject'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)

## S4 method for signature 'data.table'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)

## S4 method for signature 'list'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)

## S4 method for signature 'character'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)

## S4 method for signature 'ANY'
as_familiar_collection(
  object,
  familiar_data_names = NULL,
  collection_name = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | `familiarCollection` object, or one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some `familiarData` objects can be created from prediction tables (`familiarDataElementPredictionTable`). Other `familiarData` objects can |

be created from data (`dataObject`, or `data.table`). Please check *details* for more information.

`familiar_data_names`
: Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name`
: Name of the collection.

`...`
: Arguments passed on to `.extract_data`

  `data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.

  `is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.

  `cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.

  `time_max` Time point which is used as the benchmark for e.g. cumulative risks generated by random forest, or the cut-off value for Uno's concordance index. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

  `evaluation_times` One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

  `aggregation_method` Method for aggregating variable importances for the purpose of evaluation. Variable importances are determined during feature selection steps and after training the model. Both types are evaluated, but feature selection variable importance is only evaluated at run-time.
  See the documentation for the `vimp_aggregation_method` argument in `summon_familiar` for information concerning the different available methods.
  If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

  `rank_threshold` The threshold used to define the subset of highly important features during evaluation.
  See the documentation for the `vimp_aggregation_rank_threshold` argument in `summon_familiar` for more information.
  If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

  `ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:

  - `median` (default): Use the median of the predicted values as the ensemble value for a sample.
  - `mean`: Use the mean of the predicted values as the ensemble value for a sample.

`metric` One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`features` Features that should be considered for extracting information from. Typically called in external workflows, e.g. for plotting. Internally, i.e. from summon_familiar, this variable is not used.

`feature_cluster_method` The method used to perform clustering. These are the same methods as for the `cluster_method` configuration parameter: `none`, `hclust`, `agnes`, `diana` and `pam`.
`none` cannot be used when extracting data regarding mutual correlation or feature expressions.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`feature_linkage_method` The method used for agglomerative clustering in `hclust` and `agnes`. These are the same methods as for the `cluster_linkage_method` configuration parameter: `average`, `single`, `complete`, `weighted`, and `ward`.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`feature_cluster_cut_method` The method used to divide features into separate clusters. The available methods are the same as for the `cluster_cut_method` configuration parameter: `silhouette`, `fixed_cut` and `dynamic_cut`.
`silhouette` is available for all cluster methods, but `fixed_cut` only applies to methods that create hierarchical trees (`hclust`, `agnes` and `diana`).
`dynamic_cut` requires the `dynamicTreeCut` package and can only be used with `agnes` and `hclust`.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`feature_similarity_threshold` The threshold level for pair-wise similarity that is required to form feature clusters with the `fixed_cut` method.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`feature_similarity_metric` Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and `feature_similarity_metric` therefore has the same options as `cluster_similarity_metric`: `mcfadden_r2`, `cox_snell_r2`, `nagelkerke_r2`, `spearman`, `kendall` and `pearson`.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_cluster_method` The method used to perform clustering based on distance between samples. These are the same methods as for the `cluster_method` configuration parameter: `hclust`, `agnes`, `diana` and `pam`.
`none` cannot be used when extracting data for feature expressions.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_linkage_method` The method used for agglomerative clustering in `hclust` and `agnes`. These are the same methods as for the `cluster_linkage_method` configuration parameter: `average`, `single`, `complete`, `weighted`, and `ward`.

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_similarity_metric` Metric to determine pairwise similarity between samples. Similarity is computed in the same manner as for clustering, but `sample_similarity_metric` has different options that are better suited to computing distance between samples instead of between features: `gower`, `euclidean`.

The underlying feature data is scaled to the $[0, 1]$ range (for numerical features) using the feature values across the samples. The normalisation parameters required can optionally be computed from feature data with the outer 5% (on both sides) of feature values trimmed or winsorised. To do so append `_trim` (trimming) or `_winsor` (winsorising) to the metric name. This reduces the effect of outliers somewhat.

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`icc_type` String indicating the type of intraclass correlation coefficient (`1`, `2` or `3`) that should be used to compute robustness for features in repeated measurements during the evaluation of univariate importance. These types correspond to the types in Shrout and Fleiss (1979). If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.

`data_element` String indicating which data elements are to be extracted. Default is `all`, but specific elements can be specified to speed up computations if not all elements are to be computed. This is an internal parameter that is set by, e.g. the `export_model_vimp` method.

`sample_limit` (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.

This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. `list("sample_similarity"=100, "permutation_vimp"=1000)`.

This parameter can be set for the following data elements: `sample_similarity`, `shap`, `permutation_vimp`, and `ice_data`.

`n_important_features` (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.

This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. `list("ice_data"=10, "permutation_vimp"=5)`.

This parameter can be set for the following data elements: `ice_data`, `permutation_vimp`, and `shap`.

`detail_level` (*optional*) Sets the level at which results are computed and aggregated.

- `ensemble`: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20

bootstraps and computing the model performance of the ensemble model for each bootstrap.

- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.

`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.

A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`. This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.

If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a

named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`. This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`aggregate_results` (*optional*) Flag that signifies whether results should be aggregated during evaluation. If `estimation_type` is `bias_correction` or `bc`, aggregation leads to a single bias-corrected estimate. If `estimation_type` is `bootstrap_confidence_interval` or `bci`, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if `estimation_type` is `point`.
The default value is equal to `TRUE` except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with `detail_level` and `estimation_type`, a non-default `aggregate_results` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"=TRUE, , "model_performance"=FALSE)`. This parameter exists for the same elements as `estimation_type`.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

`stratification_method` (*optional*) Method for determining the stratification threshold for creating survival groups. The actual, model-dependent, threshold value is obtained from the development data, and can afterwards be used to perform stratification on validation data.
The following stratification methods are available:

- `median` (default): The median predicted value in the development cohort is used to stratify the samples into two risk groups. For predicted outcome values that build a continuous spectrum, the two risk groups in the development cohort will be roughly equal in size.
- `mean`: The mean predicted value in the development cohort is used to stratify the samples into two risk groups.
- `mean_trim`: As `mean`, but based on the set of predicted values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- `mean_winsor`: As `mean`, but based on the set of predicted values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.

- `fixed`: Samples are stratified based on the sample quantiles of the predicted values. These quantiles are defined using the `stratification_threshold` parameter.
- `optimised`: Use maximally selected rank statistics to determine the optimal threshold (Lausen and Schumacher, 1992; Hothorn et al., 2003) to stratify samples into two optimally separated risk groups.

One or more stratification methods can be selected simultaneously.
This parameter is only relevant for `survival` outcomes.

## Details

A `data` argument is expected if the `object` argument is a `familiarEnsemble` object or one or more `familiarModel` objects.

## Value

A `familiarCollection` object.

---

`as_familiar_data` *Conversion to familiarData object.*

---

## Description

Creates `familiarData` a object from `familiarEnsemble` or `familiarModel` objects.

## Usage

```
as_familiar_data(object, ...)

## S4 method for signature 'familiarData'
as_familiar_data(object, ...)

## S4 method for signature 'familiarEnsemble'
as_familiar_data(object, name = NULL, ...)

## S4 method for signature 'familiarDataElementPredictionTable'
as_familiar_data(object, name = NULL, ...)

## S4 method for signature 'dataObject'
as_familiar_data(object, name = NULL, ...)

## S4 method for signature 'familiarModel'
as_familiar_data(object, ...)

## S4 method for signature 'list'
as_familiar_data(object, ...)
```

```
## S4 method for signature 'character'
as_familiar_data(object, ...)

## S4 method for signature 'ANY'
as_familiar_data(object, ...)
```

## Arguments

| | |
|---|---|
| object | A `familiarData` object, or a `familiarEnsemble` or `familiarModel` objects that will be internally converted to a `familiarData` object. Paths to such objects can also be provided. |
| ... | Arguments passed on to `.extract_data` |

`data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.

`is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.

`cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.

`time_max` Time point which is used as the benchmark for e.g. cumulative risks generated by random forest, or the cut-off value for Uno's concordance index. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

`evaluation_times` One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

`aggregation_method` Method for aggregating variable importances for the purpose of evaluation. Variable importances are determined during feature selection steps and after training the model. Both types are evaluated, but feature selection variable importance is only evaluated at run-time.
See the documentation for the `vimp_aggregation_method` argument in `summon_familiar` for information concerning the different available methods.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`rank_threshold` The threshold used to define the subset of highly important features during evaluation.
See the documentation for the `vimp_aggregation_rank_threshold` argument in `summon_familiar` for more information.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

metric One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

features Features that should be considered for extracting information from. Typically called in external workflows, e.g. for plotting. Internally, i.e. from summon_familiar, this variable is not used.

feature_cluster_method The method used to perform clustering. These are the same methods as for the cluster_method configuration parameter: none, hclust, agnes, diana and pam.
none cannot be used when extracting data regarding mutual correlation or feature expressions.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_linkage_method The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_cluster_cut_method The method used to divide features into separate clusters. The available methods are the same as for the cluster_cut_method configuration parameter: silhouette, fixed_cut and dynamic_cut.
silhouette is available for all cluster methods, but fixed_cut only applies to methods that create hierarchical trees (hclust, agnes and diana). dynamic_cut requires the dynamicTreeCut package and can only be used with agnes and hclust.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_threshold The threshold level for pair-wise similarity that is required to form feature clusters with the fixed_cut method.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_metric Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and feature_similarity_metric therefore has the same options as cluster_similarity_metric: mcfadden_r2, cox_snell_r2, nagelkerke_r2, spearman, kendall and pearson.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

sample_cluster_method The method used to perform clustering based on distance between samples. These are the same methods as for the cluster_method configuration parameter: hclust, agnes, diana and pam.
none cannot be used when extracting data for feature expressions.

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_linkage_method` The method used for agglomerative clustering in `hclust` and `agnes`. These are the same methods as for the `cluster_linkage_method` configuration parameter: `average`, `single`, `complete`, `weighted`, and `ward`.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_similarity_metric` Metric to determine pairwise similarity between samples. Similarity is computed in the same manner as for clustering, but `sample_similarity_metric` has different options that are better suited to computing distance between samples instead of between features: `gower`, `euclidean`.
The underlying feature data is scaled to the $[0, 1]$ range (for numerical features) using the feature values across the samples. The normalisation parameters required can optionally be computed from feature data with the outer 5% (on both sides) of feature values trimmed or winsorised. To do so append `_trim` (trimming) or `_winsor` (winsorising) to the metric name. This reduces the effect of outliers somewhat.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`icc_type` String indicating the type of intraclass correlation coefficient (`1`, `2` or `3`) that should be used to compute robustness for features in repeated measurements during the evaluation of univariate importance. These types correspond to the types in Shrout and Fleiss (1979). If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.

`data_element` String indicating which data elements are to be extracted. Default is `all`, but specific elements can be specified to speed up computations if not all elements are to be computed. This is an internal parameter that is set by, e.g. the `export_model_vimp` method.

`sample_limit` (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. `list("sample_similarity"=100, "permutation_vimp"=1000)`.
This parameter can be set for the following data elements: `sample_similarity`, `shap`, `permutation_vimp`, and `ice_data`.

`n_important_features` (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. `list("ice_data"=10, "permutation_vimp"=5)`.
This parameter can be set for the following data elements: `ice_data`, `permutation_vimp`, and `shap`.

`detail_level` (*optional*) Sets the level at which results are computed and aggregated.

- `ensemble`: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.
A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.
If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.

- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

aggregate_results (*optional*) Flag that signifies whether results should be aggregated during evaluation. If estimation_type is bias_correction or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.
The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE, , "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

stratification_method (*optional*) Method for determining the stratification threshold for creating survival groups. The actual, model-dependent, threshold value is obtained from the development data, and can afterwards be used to perform stratification on validation data.
The following stratification methods are available:

- median (default): The median predicted value in the development cohort is used to stratify the samples into two risk groups. For predicted outcome values that build a continuous spectrum, the two risk groups in the development cohort will be roughly equal in size.
- mean: The mean predicted value in the development cohort is used to stratify the samples into two risk groups.

- mean_trim: As mean, but based on the set of predicted values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- mean_winsor: As mean, but based on the set of predicted values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- fixed: Samples are stratified based on the sample quantiles of the predicted values. These quantiles are defined using the stratification_threshold parameter.
- optimised: Use maximally selected rank statistics to determine the optimal threshold (Lausen and Schumacher, 1992; Hothorn et al., 2003) to stratify samples into two optimally separated risk groups.

One or more stratification methods can be selected simultaneously.
This parameter is only relevant for survival outcomes.

name Name of the familiarData object. If not set, a name is automatically generated.

## Details

The data argument is required if familiarEnsemble or familiarModel objects are provided.

## Value

A familiarData object.

---

# as_familiar_ensemble *Conversion to familiarEnsemble object.*

---

## Description

Creates familiarEnsemble a object from familiarModel objects.

## Usage

```
as_familiar_ensemble(object, ...)

## S4 method for signature 'familiarEnsemble'
as_familiar_ensemble(object, ...)

## S4 method for signature 'familiarModel'
as_familiar_ensemble(object, ...)

## S4 method for signature 'familiarNoveltyDetector'
as_familiar_ensemble(object, ...)

## S4 method for signature 'list'
as_familiar_ensemble(object, ...)
```

```
## S4 method for signature 'character'
as_familiar_ensemble(object, ...)

## S4 method for signature 'ANY'
as_familiar_ensemble(object, ...)
```

## Arguments

| | |
|---|---|
| object | A familiarEnsemble object, or one or more familiarModel objects that will be internally converted to a familiarEnsemble object. Paths to such objects can also be provided. |
| ... | Unused arguments. |

## Value

A familiarEnsemble object.

---

# as_prediction_table *Convert to prediction table object*

---

## Description

Creates a prediction table object from input data.

## Usage

```
as_prediction_table(
  x,
  type,
  y = waiver(),
  batch_id = waiver(),
  sample_id = waiver(),
  series_id = waiver(),
  repetition_id = waiver(),
  time = waiver(),
  class_levels = waiver(),
  value_range = waiver(),
  event_indicator = waiver(),
  censoring_indicator = waiver(),
  learner = waiver(),
  vimp_method = waiver(),
  model_object = NULL,
  data = NULL
)
```

## Arguments

| | |
|---|---|
| x | Values predicted using a learner. For all but `classification` problems, predicted values should be a single vector of values in any format that results in a single-column `data.table` using `data.table::as.data.table`. For `classification` problems, predicted values are probabilities for each class. Here, it is recommended to ensure probabilities can be mapped to their respective class, e.g. using a named list. |
| type | The type of prediction table that should be created. The following types are available:<br>• `regression`: The predicted values are values for a regression.<br>• `classification`: The predicted values are probabilities for specific classes.<br>• `hazard_ratio`: The predicted values are hazard ratios.<br>• `cumulative_hazard`: The predicted values are cumulative hazards at time `time`.<br>• `expected_survival_time`: The predicted values are expected survival times.<br>• `survival_probability`: The predicted values are survival probabilities at time `time`. |
| y | Known outcome value corresponding to each entry in `x`. For survival-related outcomes, two sets of values are expected, corresponding to the observed time and event status, respectively. Alternatively, a `survival::Surv` object can be provided. |
| batch_id | (*optional*) Array of batch or cohort identifiers.<br>In familiar any row of data is organised by four identifiers:<br>• The batch identifier `batch_id`: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.<br>• The sample identifier `sample_id`: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.<br>• The series identifier `series_id`: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.<br>• The repetition identifier `repetition_id`: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries for the same series of the same sample in the same batch that share the same outcome are encountered. |
| sample_id | (*optional*) Array of sample or subject identifiers. See `batch_id` above for more details.<br>If unset, every row will be identified as a single sample. |
| series_id | (*optional*) Array of series identifiers, which distinguish between measurements that are part of a series for a single sample. See `batch_id` above for more details. |
| repetition_id | (*optional*) Array of repetition identifiers, which distinguishes between repeated measurements within a single series. See `batch_id` above for more details. |

| | |
|---|---|
| time | Time point at which the predicted values are generated e.g. the cumulative risks generated by random forest.<br>This parameter is only relevant for survival outcomes. |
| class_levels | (*optional*) Class levels for binomial or multinomial outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column. |
| value_range | Range of observed, **not predicted**, values.<br>This parameter is only relevant for continuous outcomes. |
| event_indicator | (**recommended**) Indicator for events in survival and competing_risk analyses. familiar will automatically recognise 1, true, t, y and yes as event indicators, including different capitalisations. If this parameter is set, it replaces the default values. |
| censoring_indicator | (**recommended**) Indicator for right-censoring in survival and competing_risk analyses. familiar will automatically recognise 0, false, f, n, no as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values. |
| learner | The type of learner that generated the predictions. |
| vimp_method | The type of variable importance method for identifying the features included by the learner that generated the predictions. |
| model_object | A familiarModel or familiarEnsemble that can be used (and is used internally) for setting several of the other arguments of this function. |
| data | A familiar dataObject object that can be used (and is used internally) for setting many of the other arguments of this function. |

## Value

A prediction table object.

---

# coef *Extract model coefficients*

---

## Description

Extract model coefficients

## Usage

```
coef(object, ...)

## S4 method for signature 'familiarModel'
coef(object, ...)
```

### Arguments

| | |
|---|---|
| object | a familiarModel object |
| ... | additional arguments passed to coef methods for the underlying model, when available. |

### Details

This method extends the coef S3 method. For some models coef requires information that is trimmed from the model. In this case a copy of the model coefficient is stored with the model, and returned.

### Value

Coefficients extracted from the model in the familiarModel object, if any.

---

## dataObject-class *Data object*

---

### Description

The dataObject class is used to resolve the issue of keeping track of pre-processing status and data loading inside complex workflows, e.g. nested predict functions inside a calibration function.

### Slots

data NULL or data table containing the data. This is the data which will be read and used.

preprocessing_level character indicating the level of pre-processing already conducted.

outcome_type character, determines the outcome type.

outcome_info Outcome information object, which contains additional information concerning the outcome, such as class levels.

feature_info List of objects containing feature information, e.g., name, class levels, transformation, normalisation and clustering parameters. Optional.

data_column_info Object containing column information.

data_id Data identifier for dataset. Set using internal routines if the dataObject was created from a delayedDataObject

run_id Run identifier for dataset. Set using internal routines if the dataObject was created from a delayedDataObject

validation Identifies if validation or development samples were loaded. Set using internal routines if the dataObject was created from a delayedDataObject.

sample_seed Seed used for creating a bootstrap of the data.

`delayedDataObject-class` *Data object with delayed loading*

## Description

The delayed loading object provides an interface to the backend data. This data object is typically used within the evaluation pipeline to load data when needed.

## Slots

`data` NULL or data table containing the data. If present (not `NULL`), data is considered loaded. This should not happen – load_data_object auto- matically creates a dataObject from the delayed-DataObject.

`preprocessing_level` character indicating the level of pre-processing already conducted. `"none"` by default.

`outcome_type` character, determines the outcome type.

`outcome_info` Outcome information object, which contains additional information concerning the outcome, such as class levels.

`feature_info` List of objects containing feature information, e.g., name, class levels, transformation, normalisation and clustering parameters. Optional.

`data_column_info` Object containing column information.

`data_id` integer. Defines the data_id of the dataset that should be loaded.

`run_id` integer. Defines the run_id of the dataset that should be load. Together with data_id, run_id and validation allows for looking up the sample set. If run_id is left unset (NA_integer_), this will force the run_id to be set using the model, vimp_method or ensemble object. This is used during the evaluation process to load data specifically related to training, internal validation and external validation. The run-tables (which contain information about data partitioning) associated with these objects are used to look-up the run_id based on the data_id (that is always explicitly set). The perform_task method for familiarTaskEvaluate uses this aspect explicitly.

`validation` logical. This determines which internal data set will be loaded. If TRUE, the validation data will be loaded, whereas FALSE loads the development data.

`aggregate_on_load` logical. Determines whether data is aggregated after loading.

`sample_set_on_load` NULL or vector of sample identifiers to be loaded. Overrides any `sample_seed` that may have been provided.

## experimentData-class *Experiment data*

### Description

An experimentData object contains information concerning the experiment. These objects can be used to instantiate multiple experiments using the same iterations, feature information and variable importance.

### Details

experimentData objects are primarily used to improve reproducibility, since these allow for training models on a shared foundation.

### Slots

`experiment_setup` Contains regarding the experimental setup that is used to generate the iteration list.

`iteration_list` List of iteration data that determines which instances are assigned to training, validation and test sets.

`feature_info` Feature information objects. Only available if the experimentData object was generated using the `precompute_feature_info` or `precompute_vimp` functions.

`vimp_hyperparameter_list` List of hyperparameters for variable importance objects. Only available if the experimentData object was created using the `precompute_vimp` function.

`vimp_table_list` List of variable importance table objects. Only available if the experimentData object was created using the `precompute_vimp` function.

`project_id` Identifier of the project that generated the experimentData object.

`familiar_version` Version of the familiar package used to create this experimentData.

### See Also

`precompute_data_assignment` `precompute_feature_info`, `precompute_vimp`

## export_all *Extract and export all data.*

### Description

Extract and export all data from a familiarCollection.

## Usage

```
export_all(object, dir_path = NULL, aggregate_results = waiver(), ...)

## S4 method for signature 'familiarCollection'
export_all(object, dir_path = NULL, aggregate_results = waiver(), ...)

## S4 method for signature 'ANY'
export_all(object, dir_path = NULL, aggregate_results = waiver(), ...)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| ... | Arguments passed on to .extract_data, as_familiar_collection |

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

time_max Time point which is used as the benchmark for e.g. cumulative risks generated by random forest, or the cut-off value for Uno's concordance index. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

aggregation_method Method for aggregating variable importances for the purpose of evaluation. Variable importances are determined during feature selection steps and after training the model. Both types are evaluated, but feature selection variable importance is only evaluated at run-time.

See the documentation for the vimp_aggregation_method argument in summon_familiar for information concerning the different available methods.

If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

rank_threshold The threshold used to define the subset of highly important features during evaluation.

See the documentation for the vimp_aggregation_rank_threshold argument in summon_familiar for more information.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

metric One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

features Features that should be considered for extracting information from. Typically called in external workflows, e.g. for plotting. Internally, i.e. from summon_familiar, this variable is not used.

feature_cluster_method The method used to perform clustering. These are the same methods as for the cluster_method configuration parameter: none, hclust, agnes, diana and pam.
none cannot be used when extracting data regarding mutual correlation or feature expressions.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_linkage_method The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_cluster_cut_method The method used to divide features into separate clusters. The available methods are the same as for the cluster_cut_method configuration parameter: silhouette, fixed_cut and dynamic_cut.
silhouette is available for all cluster methods, but fixed_cut only applies to methods that create hierarchical trees (hclust, agnes and diana).
dynamic_cut requires the dynamicTreeCut package and can only be used with agnes and hclust.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_threshold The threshold level for pair-wise similarity that is required to form feature clusters with the fixed_cut method.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_metric Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and feature_similarity_metric therefore has the same options as cluster_similarity_metric: mcfadden_r2, cox_snell_r2, nagelkerke_r2, spearman, kendall and pearson.

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_cluster_method` The method used to perform clustering based on distance between samples. These are the same methods as for the `cluster_method` configuration parameter: `hclust`, `agnes`, `diana` and `pam`.
`none` cannot be used when extracting data for feature expressions.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_linkage_method` The method used for agglomerative clustering in `hclust` and `agnes`. These are the same methods as for the `cluster_linkage_method` configuration parameter: `average`, `single`, `complete`, `weighted`, and `ward`.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_similarity_metric` Metric to determine pairwise similarity between samples. Similarity is computed in the same manner as for clustering, but `sample_similarity_metric` has different options that are better suited to computing distance between samples instead of between features: `gower`, `euclidean`.
The underlying feature data is scaled to the $[0, 1]$ range (for numerical features) using the feature values across the samples. The normalisation parameters required can optionally be computed from feature data with the outer 5% (on both sides) of feature values trimmed or winsorised. To do so append `_trim` (trimming) or `_winsor` (winsorising) to the metric name. This reduces the effect of outliers somewhat.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`icc_type` String indicating the type of intraclass correlation coefficient (`1`, `2` or `3`) that should be used to compute robustness for features in repeated measurements during the evaluation of univariate importance. These types correspond to the types in Shrout and Fleiss (1979). If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.

`data_element` String indicating which data elements are to be extracted. Default is `all`, but specific elements can be specified to speed up computations if not all elements are to be computed. This is an internal parameter that is set by, e.g. the `export_model_vimp` method.

`sample_limit` (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. `list("sample_similarity"=100, "permutation_vimp"=1000)`.
This parameter can be set for the following data elements: `sample_similarity`, `shap`, `permutation_vimp`, and `ice_data`.

`n_important_features` (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. `list("ice_data"=10, "permutation_vimp"=5)`.
This parameter can be set for the following data elements: `ice_data`, `permutation_vimp`, and `shap`.

`detail_level` (*optional*) Sets the level at which results are computed and aggregated.

- `ensemble`: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.
A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.
If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possi-

ble. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

stratification_method (*optional*) Method for determining the stratification threshold for creating survival groups. The actual, model-dependent, threshold value is obtained from the development data, and can afterwards be used to perform stratification on validation data.
The following stratification methods are available:

- median (default): The median predicted value in the development cohort is used to stratify the samples into two risk groups. For predicted outcome values that build a continuous spectrum, the two risk groups in the development cohort will be roughly equal in size.
- mean: The mean predicted value in the development cohort is used to stratify the samples into two risk groups.
- mean_trim: As mean, but based on the set of predicted values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- mean_winsor: As mean, but based on the set of predicted values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- fixed: Samples are stratified based on the sample quantiles of the predicted values. These quantiles are defined using the stratification_threshold

parameter.

- optimised: Use maximally selected rank statistics to determine the optimal threshold (Lausen and Schumacher, 1992; Hothorn et al., 2003) to stratify samples into two optimally separated risk groups.

One or more stratification methods can be selected simultaneously.
This parameter is only relevant for survival outcomes.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

## Details

Data, such as model performance and calibration information, is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from object and dir_path are only used if object is not a familiarCollection object, or a path to one.

## Value

A list of data.tables (if dir_path is not provided), or nothing, as all data is exported to csv files.

---

# export_auc_data *Extract and export ROC and Precision-Recall curves.*

---

## Description

Extract and export ROC and Precision-Recall curves for models in a familiarCollection.

## Usage

```
export_auc_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_auc_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
```

```
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_auc_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to extract_auc_data, as_familiar_collection |

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20

bootstraps and computing the model performance of the ensemble model for each bootstrap.

- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.
A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.
If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a

named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`. This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name` Name of the collection.

## Details

Data is usually collected from a `familiarCollection` object. However, you can also provide one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

ROC curve data are exported for individual and ensemble models. For ensemble models, a credibility interval for the ROC curve is determined using bootstrapping for each metric. In case of multinomial outcomes, ROC-curves are computed for each class, using a one-against-all approach.

## Value

A list of data.tables (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_calibration_data`

*Extract and export calibration and goodness-of-fit tests.*

---

## Description

Extract and export calibration and goodness-of-fit tests for data in a familiarCollection.

## Usage

```
export_calibration_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_calibration_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_calibration_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `familiarCollection` object, or other other objects from which a `familiarCollection` can be extracted. See details for more information. |
| `dir_path` | Path to folder where extracted data should be saved. `NULL` will allow export as a structured list of data.tables. |
| `aggregate_results` | Flag that signifies whether results should be aggregated for export. |
| `export_collection` | (*optional*) Exports the collection if TRUE. |
| `...` | Arguments passed on to `extract_calibration_data`, `as_familiar_collection`<br>`data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.<br>`is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.<br>`cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.<br>`evaluation_times` One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. |

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

`ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:

- `median` (default): Use the median of the predicted values as the ensemble value for a sample.
- `mean`: Use the mean of the predicted values as the ensemble value for a sample.

`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.

`detail_level` (*optional*) Sets the level at which results are computed and aggregated.

- `ensemble`: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.

`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.

A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.

This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

## Details

Data is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

Calibration tests are performed based on expected (predicted) and observed outcomes. For all outcomes, calibration-at-the-large and calibration slopes are determined. Furthermore, for all but survival outcomes, a repeated, randomised grouping Hosmer-Lemeshow test is performed. For survival outcomes, the Nam-D'Agostino and Greenwood-Nam-D'Agostino tests are performed.

## Value

A list of data.tables (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_calibration_info`

*Extract and export calibration information.*

---

## Description

Extract and export calibration information (e.g. baseline survival) for data in a familiarCollection.

## Usage

```
export_calibration_info(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_calibration_info(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_calibration_info(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to as_familiar_collection<br>familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.<br>collection_name Name of the collection. |

### Details

Data is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from object and dir_path are only used if object is not a familiarCollection object, or a path to one.

Currently only baseline survival is exported as supporting calibration information. See export_calibration_data for export of direct assessment of calibration, including calibration and goodness-of-fit tests.

### Value

A data.table (if dir_path is not provided), or nothing, as all data is exported to csv files.

---

## export_confusion_matrix_data

*Extract and export confusion matrices.*

---

### Description

Extract and export confusion matrics for models in a familiarCollection.

### Usage

```
export_confusion_matrix_data(
  object,
  dir_path = NULL,
  export_collection = FALSE,
  ...
```

```
)

## S4 method for signature 'familiarCollection'
export_confusion_matrix_data(
  object,
  dir_path = NULL,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_confusion_matrix_data(
  object,
  dir_path = NULL,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `familiarCollection` object, or other other objects from which a `familiarCollection` can be extracted. See details for more information. |
| `dir_path` | Path to folder where extracted data should be saved. `NULL` will allow export as a structured list of data.tables. |
| `export_collection` | (*optional*) Exports the collection if TRUE. |
| `...` | Arguments passed on to `extract_confusion_matrix`, `as_familiar_collection`<br>`data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.<br>`is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.<br>`cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.<br>`ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:<br>• `median` (default): Use the median of the predicted values as the ensemble value for a sample.<br>• `mean`: Use the mean of the predicted values as the ensemble value for a sample.<br>`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.<br>`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.<br>`detail_level` (*optional*) Sets the level at which results are computed and aggregated. |

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.
A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

## Details

Data is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together

with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

Confusion matrices are exported for individual and ensemble models.

## Value

A list of data.tables (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_decision_curve_analysis_data` *Extract and export decision curve analysis data.*

---

## Description

Extract and export decision curve analysis data in a familiarCollection.

## Usage

```
export_decision_curve_analysis_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  ...
)

## S4 method for signature 'familiarCollection'
export_decision_curve_analysis_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  ...
)

## S4 method for signature 'ANY'
export_decision_curve_analysis_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  ...
)
```

### Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| ... | Arguments passed on to as_familiar_collection<br>familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.<br>collection_name Name of the collection. |

### Details

Data is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from object and dir_path are only used if object is not a familiarCollection object, or a path to one.

Decision curve analysis data is computed for categorical outcomes, i.e. binomial and multinomial, as well as survival outcomes.

### Value

A list of data.table (if dir_path is not provided), or nothing, as all data is exported to csv files.

---

## export_feature_expressions

*Extract and export feature expressions.*

---

### Description

Extract and export feature expressions for the features in a familiarCollection.

### Usage

```
export_feature_expressions(
  object,
  features = waiver(),
  dir_path = NULL,
  evaluation_time = waiver(),
  export_collection = FALSE,
  ...
```

```
)

## S4 method for signature 'familiarCollection'
export_feature_expressions(
  object,
  features = waiver(),
  dir_path = NULL,
  evaluation_time = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_feature_expressions(
  object,
  features = waiver(),
  dir_path = NULL,
  evaluation_time = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| features | Features that should be exported. If NULL or waiver(), all features exported (default). |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| evaluation_time | One or more time points that are used to create the outcome columns in expression plots. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarData objects. Only used for survival outcomes. |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to extract_feature_expression, as_familiar_collection, as_data_object<br>feature_similarity Table containing pairwise distance between sample. This is used to determine cluster information, and indicate which samples are similar. The table is created by the extract_sample_similarity method.<br>data A dataObject object, data.table or data.frame that constitutes the data that are assessed.<br>evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. |

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

`feature_cluster_method` The method used to perform clustering. These are the same methods as for the `cluster_method` configuration parameter: `none`, `hclust`, `agnes`, `diana` and `pam`.
`none` cannot be used when extracting data regarding mutual correlation or feature expressions.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`feature_linkage_method` The method used for agglomerative clustering in `hclust` and `agnes`. These are the same methods as for the `cluster_linkage_method` configuration parameter: `average`, `single`, `complete`, `weighted`, and `ward`.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`feature_similarity_metric` Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and `feature_similarity_metric` therefore has the same options as `cluster_similarity_metric`: `mcfadden_r2`, `cox_snell_r2`, `nagelkerke_r2`, `spearman`, `kendall` and `pearson`.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_cluster_method` The method used to perform clustering based on distance between samples. These are the same methods as for the `cluster_method` configuration parameter: `hclust`, `agnes`, `diana` and `pam`.
`none` cannot be used when extracting data for feature expressions.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_linkage_method` The method used for agglomerative clustering in `hclust` and `agnes`. These are the same methods as for the `cluster_linkage_method` configuration parameter: `average`, `single`, `complete`, `weighted`, and `ward`.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`sample_similarity_metric` Metric to determine pairwise similarity between samples. Similarity is computed in the same manner as for clustering, but `sample_similarity_metric` has different options that are better suited to computing distance between samples instead of between features: `gower`, `euclidean`.
The underlying feature data is scaled to the $[0, 1]$ range (for numerical features) using the feature values across the samples. The normalisation parameters required can optionally be computed from feature data with the outer 5% (on both sides) of feature values trimmed or winsorised. To do so append `_trim` (trimming) or `_winsor` (winsorising) to the metric name. This reduces the effect of outliers somewhat.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

check_stringency Specifies stringency of various checks. This is mostly:

- strict: default value used for summon_familiar. Thoroughly checks input data. Used internally for checking development data.
- external_warn: value used for extract_data and related methods. Less stringent checks, but will warn for possible issues. Used internally for checking data for evaluation and explanation.
- external: value used for external methods such as predict. Less stringent checks, particularly for identifier and outcome columns, which may be completely absent. Used internally for predict.

.no_features_required Internal flag to signify that data without features is allowed. Default: FALSE (most processing steps require features).

batch_id_column (**recommended**) Name of the column containing batch or cohort identifiers. This parameter is required if more than one dataset is provided, or if external validation is performed.

In familiar any row of data is organised by four identifiers:

- The batch identifier batch_id_column: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.
- The sample identifier sample_id_column: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.
- The series identifier series_id_column: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.
- The repetition identifier: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries for the same series of the same sample in the same batch that share the same outcome are encountered.

sample_id_column (**recommended**) Name of the column containing sample or subject identifiers. See batch_id_column above for more details.

If unset, every row will be identified as a single sample.

series_id_column (*optional*) Name of the column containing series identifiers, which distinguish between measurements that are part of a series for a single sample. See batch_id_column above for more details.

If unset, rows which share the same batch and sample identifiers but have a different outcome are assigned unique series identifiers.

development_batch_id (*optional*) One or more batch or cohort identifiers to constitute data sets for development. Defaults to all, or all minus the identifiers in validation_batch_id for external validation. Required if external validation is performed and validation_batch_id is not provided.

validation_batch_id (*optional*) One or more batch or cohort identifiers to constitute data sets for external validation. Defaults to all data sets except those in development_batch_id for external validation, or none if not. Required if development_batch_id is not provided.

outcome_name (*optional*) Name of the modelled outcome. This name will be used in figures created by familiar.
If not set, the column name in outcome_column will be used for binomial, multinomial, and continuous outcomes. For other outcomes (survival and competing_risk) no default is used.

outcome_column (**recommended**) Name of the column containing the outcome of interest. May be identified from a formula, if a formula is provided as an argument. Otherwise an error is raised. Note that survival and competing_risk outcome type outcomes require two columns that indicate the time-to-event or the time of last follow-up and the event status.

outcome_type (**recommended**) Type of outcome found in the outcome column. The outcome type determines many aspects of the overall process, e.g. the available variable importance methods and learners, but also the type of assessments that can be conducted to evaluate the resulting models. Implemented outcome types are:

- binomial: categorical outcome with 2 levels.
- multinomial: categorical outcome with 2 or more levels.
- continuous: general continuous numeric outcomes.
- survival: survival outcome for time-to-event data.

If not provided, the algorithm will attempt to obtain outcome_type from contents of the outcome column. This may lead to unexpected results, and we therefore advise to provide this information manually.
Note that competing_risk survival analysis are not fully supported, and is currently not a valid choice for outcome_type. The count outcome type was deprecated in version 2.0.0, and superseded by continuous.

class_levels (*optional*) Class levels for binomial or multinomial outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column.

event_indicator (**recommended**) Indicator for events in survival and competing_risk analyses. familiar will automatically recognise 1, true, t, y and yes as event indicators, including different capitalisations. If this parameter is set, it replaces the default values.

censoring_indicator (**recommended**) Indicator for right-censoring in survival and competing_risk analyses. familiar will automatically recognise 0, false, f, n, no as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values.

competing_risk_indicator (**recommended**) Indicator for competing risks in competing_risk analyses. There are no default values, and if unset, all values other than those specified by the event_indicator and censoring_indicator parameters are considered to indicate competing risks.

exclude_features (*optional*) Feature columns that will be removed from the data set. Cannot overlap with features in signature, novelty_features

or include_features.

include_features (*optional*) Feature columns that are specifically included in the data set. By default all features are included. Cannot overlap with exclude_features, but may overlap signature. Features in signature and novelty_features are always included. If both exclude_features and include_features are provided, include_features takes precedence, provided that there is no overlap between the two.

reference_method (*optional*) Method used to set reference levels for categorical features. There are several options:

- auto (default): Categorical features that are not explicitly set by the user, i.e. columns containing boolean values or characters, use the most frequent level as reference. Categorical features that are explicitly set, i.e. as factors, are used as is.
- always: Both automatically detected and user-specified categorical features have the reference level set to the most frequent level. Ordinal features are not altered, but are used as is.
- never: User-specified categorical features are used as is. Automatically detected categorical features are simply sorted, and the first level is then used as the reference level. This was the behaviour prior to familiar version 1.3.0.

## Details

All parameters aside from object and dir_path are only used if object is not a familiarCollection object, or a path to one.

Feature similarity data can be created from dataObject, or data.table objects. For data.table, see as_data_object for additional arguments.

## Value

A data.table (if dir_path is not provided), or nothing, as all data is exported to csv files.

---

# export_feature_similarity

*Extract and export mutual correlation between features.*

---

## Description

Extract and export mutual correlation between features in a familiarCollection.

## Usage

```
export_feature_similarity(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
```

```
  features = waiver(),
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  export_dendrogram = FALSE,
  export_ordered_data = FALSE,
  export_clustering = FALSE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_feature_similarity(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  features = waiver(),
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  export_dendrogram = FALSE,
  export_ordered_data = FALSE,
  export_clustering = FALSE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_feature_similarity(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  features = waiver(),
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  export_dendrogram = FALSE,
  export_ordered_data = FALSE,
  export_clustering = FALSE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| features | Features that should be considered for extracting information from. Typically called in external workflows, e.g. for plotting. Internally, i.e. from summon_familiar, this variable is not used. |
| feature_cluster_method | The method used to perform clustering. These are the same methods as for the cluster_method configuration parameter: none, hclust, agnes, diana and pam.<br>none cannot be used when extracting data regarding mutual correlation or feature expressions.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| feature_linkage_method | The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| feature_cluster_cut_method | The method used to divide features into separate clusters. The available methods are the same as for the cluster_cut_method configuration parameter: silhouette, fixed_cut and dynamic_cut.<br>silhouette is available for all cluster methods, but fixed_cut only applies to methods that create hierarchical trees (hclust, agnes and diana). dynamic_cut requires the dynamicTreeCut package and can only be used with agnes and hclust.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| feature_similarity_threshold | The threshold level for pair-wise similarity that is required to form feature clusters with the fixed_cut method.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| export_dendrogram | Add dendrogram in the data element objects. |
| export_ordered_data | Add feature label ordering to data in the data element objects. |
| export_clustering | Add clustering information to data. |

export_collection
: (*optional*) Exports the collection if TRUE.

...
: Arguments passed on to as_familiar_collection, as_data_object

familiar_data_names
: Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name
: Name of the collection.

data
: A data.frame or data.table, a path to such tables on a local or network drive, or a path to tabular data that may be converted to these formats.

check_stringency
: Specifies stringency of various checks. This is mostly:
  - strict: default value used for summon_familiar. Thoroughly checks input data. Used internally for checking development data.
  - external_warn: value used for extract_data and related methods. Less stringent checks, but will warn for possible issues. Used internally for checking data for evaluation and explanation.
  - external: value used for external methods such as predict. Less stringent checks, particularly for identifier and outcome columns, which may be completely absent. Used internally for predict.

.no_features_required
: Internal flag to signify that data without features is allowed. Default: FALSE (most processing steps require features).

batch_id_column
: (**recommended**) Name of the column containing batch or cohort identifiers. This parameter is required if more than one dataset is provided, or if external validation is performed.
  In familiar any row of data is organised by four identifiers:
  - The batch identifier batch_id_column: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.
  - The sample identifier sample_id_column: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.
  - The series identifier series_id_column: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.
  - The repetition identifier: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries for the same series of the same sample in the same batch that share the same outcome are encountered.

sample_id_column
: (**recommended**) Name of the column containing sample or subject identifiers. See batch_id_column above for more details.
  If unset, every row will be identified as a single sample.

series_id_column
: (*optional*) Name of the column containing series identifiers, which distinguish between measurements that are part of a series for a single sample. See batch_id_column above for more details.
  If unset, rows which share the same batch and sample identifiers but have a different outcome are assigned unique series identifiers.

development_batch_id (*optional*) One or more batch or cohort identifiers to constitute data sets for development. Defaults to all, or all minus the identifiers in validation_batch_id for external validation. Required if external validation is performed and validation_batch_id is not provided.

validation_batch_id (*optional*) One or more batch or cohort identifiers to constitute data sets for external validation. Defaults to all data sets except those in development_batch_id for external validation, or none if not. Required if development_batch_id is not provided.

outcome_name (*optional*) Name of the modelled outcome. This name will be used in figures created by familiar.

If not set, the column name in outcome_column will be used for binomial, multinomial, and continuous outcomes. For other outcomes (survival and competing_risk) no default is used.

outcome_column (**recommended**) Name of the column containing the outcome of interest. May be identified from a formula, if a formula is provided as an argument. Otherwise an error is raised. Note that survival and competing_risk outcome type outcomes require two columns that indicate the time-to-event or the time of last follow-up and the event status.

outcome_type (**recommended**) Type of outcome found in the outcome column. The outcome type determines many aspects of the overall process, e.g. the available variable importance methods and learners, but also the type of assessments that can be conducted to evaluate the resulting models. Implemented outcome types are:

- binomial: categorical outcome with 2 levels.
- multinomial: categorical outcome with 2 or more levels.
- continuous: general continuous numeric outcomes.
- survival: survival outcome for time-to-event data.

If not provided, the algorithm will attempt to obtain outcome_type from contents of the outcome column. This may lead to unexpected results, and we therefore advise to provide this information manually.

Note that competing_risk survival analysis are not fully supported, and is currently not a valid choice for outcome_type. The count outcome type was deprecated in version 2.0.0, and superseded by continuous.

class_levels (*optional*) Class levels for binomial or multinomial outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column.

event_indicator (**recommended**) Indicator for events in survival and competing_risk analyses. familiar will automatically recognise 1, true, t, y and yes as event indicators, including different capitalisations. If this parameter is set, it replaces the default values.

censoring_indicator (**recommended**) Indicator for right-censoring in survival and competing_risk analyses. familiar will automatically recognise 0, false, f, n, no as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values.

competing_risk_indicator (**recommended**) Indicator for competing risks in competing_risk analyses. There are no default values, and if unset, all

values other than those specified by the `event_indicator` and `censoring_indicator` parameters are considered to indicate competing risks.

`exclude_features` (*optional*) Feature columns that will be removed from the data set. Cannot overlap with features in `signature`, `novelty_features` or `include_features`.

`include_features` (*optional*) Feature columns that are specifically included in the data set. By default all features are included. Cannot overlap with `exclude_features`, but may overlap `signature`. Features in `signature` and `novelty_features` are always included. If both `exclude_features` and `include_features` are provided, `include_features` takes precedence, provided that there is no overlap between the two.

`reference_method` (*optional*) Method used to set reference levels for categorical features. There are several options:

- `auto` (default): Categorical features that are not explicitly set by the user, i.e. columns containing boolean values or characters, use the most frequent level as reference. Categorical features that are explicitly set, i.e. as factors, are used as is.
- `always`: Both automatically detected and user-specified categorical features have the reference level set to the most frequent level. Ordinal features are not altered, but are used as is.
- `never`: User-specified categorical features are used as is. Automatically detected categorical features are simply sorted, and the first level is then used as the reference level. This was the behaviour prior to familiar version 1.3.0.

## Details

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

Feature similarity data can be created from `dataObject`, or `data.table` objects. For `data.table`, see `as_data_object` for additional arguments.

## Value

A list containing a data.table (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_fs_vimp` *Extract and export feature selection variable importance.*

---

## Description

Extract and export feature selection variable importance from a familiarCollection.

## Usage

```
export_fs_vimp(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  aggregation_method = waiver(),
  rank_threshold = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_fs_vimp(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  aggregation_method = waiver(),
  rank_threshold = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_fs_vimp(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  aggregation_method = waiver(),
  rank_threshold = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `familiarCollection` object, or other other objects from which a `familiarCollection` can be extracted. See details for more information. |
| `dir_path` | Path to folder where extracted data should be saved. `NULL` will allow export as a structured list of data.tables. |
| `aggregate_results` | Flag that signifies whether results should be aggregated for export. |
| `aggregation_method` | (*optional*) The method used to aggregate variable importances over different data subsets, e.g. bootstraps. The following methods can be selected:<br>• `mean` (default): Use the mean rank of a feature over the subsets to determine the aggregated feature rank.<br>• `median`: Use the median rank of a feature over the subsets to determine the aggregated feature rank. |

- `best`: Use the best rank the feature obtained in any subset to determine the aggregated feature rank.
- `worst`: Use the worst rank the feature obtained in any subset to determine the aggregated feature rank.
- `stability`: Use the frequency of the feature being in the subset of highly ranked features as measure for the aggregated feature rank (Meinshausen and Buehlmann, 2010).
- `exponential`: Use a rank-weighted frequence of occurrence in the subset of highly ranked features as measure for the aggregated feature rank (Haury et al., 2011).
- `borda`: Use the borda count as measure for the aggregated feature rank (Wald et al., 2012).
- `enhanced_borda`: Use an occurrence frequency-weighted borda count as measure for the aggregated feature rank (Wald et al., 2012).
- `truncated_borda`: Use borda count computed only on features within the subset of highly ranked features.
- `enhanced_truncated_borda`: Apply both the enhanced borda method and the truncated borda method and use the resulting borda count as the aggregated feature rank.

`rank_threshold` (*optional*) The threshold used to define the subset of highly important features. If not set, this threshold is determined by maximising the variance in the occurrence value over all features over the subset size.

This parameter is only relevant for `stability`, `exponential`, `enhanced_borda`, `truncated_borda` and `enhanced_truncated_borda` methods.

`export_collection`

(*optional*) Exports the collection if TRUE.

`...` Arguments passed on to `as_familiar_collection`

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name` Name of the collection.

## Details

Data, such as model performance and calibration information, is usually collected from a `familiarCollection` object. However, you can also provide one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. Paths to the previous files can also be provided.

Unlike other export function, export using `familiarEnsemble` or `familiarModel` objects is not possible. This is because feature selection variable importance is not stored within `familiarModel` objects.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

Variable importance is based on the ranking produced by feature selection routines. In case feature selection was performed repeatedly, e.g. using bootstraps, feature ranks are first aggregated using the method defined by the `aggregation_method`, some of which require a `rank_threshold` to indicate a subset of most important features.

Information concerning highly similar features that form clusters is provided as well. This information is based on consensus clustering of the features. This clustering information is also used during aggregation to ensure that co-clustered features are only taken into account once.

## Value

A data.table (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_hyperparameters` *Extract and export model hyperparameters.*

---

## Description

Extract and export model hyperparameters from models in a familiarCollection.

## Usage

```
export_hyperparameters(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_hyperparameters(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_hyperparameters(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to as_familiar_collection<br>familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.<br>collection_name Name of the collection. |

## Details

Data, such as model performance and calibration information, is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from object and dir_path are only used if object is not a familiarCollection object, or a path to one.

Many model hyperparameters are optimised using sequential model-based optimisation. The extracted hyperparameters are those that were selected to construct the underlying models (familiarModel objects).

## Value

A data.table (if dir_path is not provided), or nothing, as all data is exported to csv files. In case of the latter, hyperparameters are summarised.

---

# export_ice_data *Extract and export individual conditional expectation data.*

---

## Description

Extract and export individual conditional expectation data.

## Usage

```
export_ice_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_ice_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_ice_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `familiarCollection` object, or other other objects from which a `familiarCollection` can be extracted. See details for more information. |
| `dir_path` | Path to folder where extracted data should be saved. `NULL` will allow export as a structured list of data.tables. |
| `aggregate_results` | Flag that signifies whether results should be aggregated for export. |
| `export_collection` | (*optional*) Exports the collection if TRUE. |
| `...` | Arguments passed on to `extract_ice`, `as_familiar_collection`<br>`features` Names of the feature or features (2) assessed simultaneously. By default `NULL`, which means that all features are assessed one-by-one.<br>`feature_x_range` When one or two features are defined using `features`, `feature_x_range` can be used to set the range of values for the first feature. For numeric features, a vector of two values is assumed to indicate a range from which `n_sample_points` are uniformly sampled. A vector of more than two values is interpreted as is, i.e. these represent the values to be sampled. For categorical features, values should represent a (sub)set of available levels. |

feature_y_range As feature_x_range, but for the second feature in case two features are defined.

n_sample_points Number of points used to sample continuous features.

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

sample_limit (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("sample_similarity"=100, "permutation_vimp"=1000).
This parameter can be set for the following data elements: sample_similarity, shap, permutation_vimp, and ice_data.

n_important_features (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("ice_data"=10, "permutation_vimp"=5).
This parameter can be set for the following data elements: ice_data, permutation_vimp, and shap.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of

model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.

- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.
A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.
If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

## Details

Data is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from object and dir_path are only used if object is not a familiarCollection object, or a path to one.

## Value

A list of data.tables (if dir_path is not provided), or nothing, as all data is exported to csv files.

---

# export_model_performance

*Extract and export metrics for model performance.*

---

## Description

Extract and export metrics for model performance of models in a familiarCollection.

## Usage

```
export_model_performance(
  object,
  dir_path = NULL,
  aggregate_results = FALSE,
```

```
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_model_performance(
  object,
  dir_path = NULL,
  aggregate_results = FALSE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_model_performance(
  object,
  dir_path = NULL,
  aggregate_results = FALSE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `familiarCollection` object, or other other objects from which a `familiarCollection` can be extracted. See details for more information. |
| `dir_path` | Path to folder where extracted data should be saved. `NULL` will allow export as a structured list of data.tables. |
| `aggregate_results` | Flag that signifies whether results should be aggregated for export. |
| `export_collection` | (*optional*) Exports the collection if TRUE. |
| `...` | Arguments passed on to `extract_performance`, `as_familiar_collection` |
| | `data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed. |
| | `is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`. |
| | `cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation. |
| | `evaluation_times` One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes. |
| | `ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are: |

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

metric One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.
A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").

This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

## Details

Data is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

Performance of individual and ensemble models is exported. For ensemble models, a credibility interval is determined using bootstrapping for each metric.

## Value

A list of data.tables (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_model_vimp` *Extract and export model-based variable importance.*

---

## Description

Extract and export model-based variable importance from a familiarCollection.

## Usage

```
export_model_vimp(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  aggregation_method = waiver(),
  rank_threshold = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_model_vimp(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  aggregation_method = waiver(),
  rank_threshold = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_model_vimp(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  aggregation_method = waiver(),
  rank_threshold = waiver(),
  export_collection = FALSE,
```

```
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| aggregation_method | (*optional*) The method used to aggregate variable importances over different data subsets, e.g. bootstraps. The following methods can be selected:<br>• mean (default): Use the mean rank of a feature over the subsets to determine the aggregated feature rank.<br>• median: Use the median rank of a feature over the subsets to determine the aggregated feature rank.<br>• best: Use the best rank the feature obtained in any subset to determine the aggregated feature rank.<br>• worst: Use the worst rank the feature obtained in any subset to determine the aggregated feature rank.<br>• stability: Use the frequency of the feature being in the subset of highly ranked features as measure for the aggregated feature rank (Meinshausen and Buehlmann, 2010).<br>• exponential: Use a rank-weighted frequence of occurrence in the subset of highly ranked features as measure for the aggregated feature rank (Haury et al., 2011).<br>• borda: Use the borda count as measure for the aggregated feature rank (Wald et al., 2012).<br>• enhanced_borda: Use an occurrence frequency-weighted borda count as measure for the aggregated feature rank (Wald et al., 2012).<br>• truncated_borda: Use borda count computed only on features within the subset of highly ranked features.<br>• enhanced_truncated_borda: Apply both the enhanced borda method and the truncated borda method and use the resulting borda count as the aggregated feature rank. |
| rank_threshold | (*optional*) The threshold used to define the subset of highly important features. If not set, this threshold is determined by maximising the variance in the occurrence value over all features over the subset size.<br>This parameter is only relevant for stability, exponential, enhanced_borda, truncated_borda and enhanced_truncated_borda methods. |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to as_familiar_collection |

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

## Details

Data, such as model performance and calibration information, is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from object and dir_path are only used if object is not a familiarCollection object, or a path to one.

Variable importance is based on the ranking produced by model-specific variable importance routines, e.g. permutation for random forests. If such a routine is absent, variable importance is based on the feature selection method that led to the features included in the model. In case multiple models (familiarModel objects) are combined, feature ranks are first aggregated using the method defined by the aggregation_method, some of which require a rank_threshold to indicate a subset of most important features.

Information concerning highly similar features that form clusters is provided as well. This information is based on consensus clustering of the features that were used in the signatures of the underlying models. This clustering information is also used during aggregation to ensure that co-clustered features are only taken into account once.

## Value

A data.table (if dir_path is not provided), or nothing, as all data is exported to csv files.

---

# export_partial_dependence_data

*Extract and export partial dependence data.*

---

## Description

Extract and export partial dependence data.

## Usage

```
export_partial_dependence_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)
```

```
## S4 method for signature 'familiarCollection'
export_partial_dependence_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_partial_dependence_data(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to extract_ice, as_familiar_collection<br>features Names of the feature or features (2) assessed simultaneously. By default NULL, which means that all features are assessed one-by-one.<br>feature_x_range When one or two features are defined using features, feature_x_range can be used to set the range of values for the first feature. For numeric features, a vector of two values is assumed to indicate a range from which n_sample_points are uniformly sampled. A vector of more than two values is interpreted as is, i.e. these represent the values to be sampled. For categorical features, values should represent a (sub)set of available levels.<br>feature_y_range As feature_x_range, but for the second feature in case two features are defined.<br>n_sample_points Number of points used to sample continuous features.<br>data A dataObject object, data.table or data.frame that constitutes the data that are assessed.<br>is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.<br>cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation. |

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

sample_limit (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("sample_similarity"=100, "permutation_vimp"=1000).
This parameter can be set for the following data elements: sample_similarity, shap, permutation_vimp, and ice_data.

n_important_features (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("ice_data"=10, "permutation_vimp"=5).
This parameter can be set for the following data elements: ice_data, permutation_vimp, and shap.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.
A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.
If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.

- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name` Name of the collection.

## Details

Data is usually collected from a `familiarCollection` object. However, you can also provide one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

## Value

A list of data.tables (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_permutation_vimp` *Extract and export permutation variable importance.*

---

## Description

Extract and export model-based variable importance from a familiarCollection.

## Usage

```
export_permutation_vimp(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_permutation_vimp(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
```

```
  ...
)

## S4 method for signature 'ANY'
export_permutation_vimp(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `familiarCollection` object, or other other objects from which a `familiarCollection` can be extracted. See details for more information. |
| `dir_path` | Path to folder where extracted data should be saved. `NULL` will allow export as a structured list of data.tables. |
| `aggregate_results` | Flag that signifies whether results should be aggregated for export. |
| `export_collection` | (*optional*) Exports the collection if TRUE. |
| `...` | Arguments passed on to `extract_permutation_vimp`, `as_familiar_collection` |

`data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.

`is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.

`cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.

`evaluation_times` One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

`ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:

- `median` (default): Use the median of the predicted values as the ensemble value for a sample.
- `mean`: Use the mean of the predicted values as the ensemble value for a sample.

`metric` One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

feature_cluster_method The method used to perform clustering. These are the same methods as for the cluster_method configuration parameter: none, hclust, agnes, diana and pam.
none cannot be used when extracting data regarding mutual correlation or feature expressions.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_linkage_method The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_cluster_cut_method The method used to divide features into separate clusters. The available methods are the same as for the cluster_cut_method configuration parameter: silhouette, fixed_cut and dynamic_cut.
silhouette is available for all cluster methods, but fixed_cut only applies to methods that create hierarchical trees (hclust, agnes and diana). dynamic_cut requires the dynamicTreeCut package and can only be used with agnes and hclust.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_threshold The threshold level for pair-wise similarity that is required to form feature clusters with the fixed_cut method.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_metric Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and feature_similarity_metric therefore has the same options as cluster_similarity_metric: mcfadden_r2, cox_snell_r2, nagelkerke_r2, spearman, kendall and pearson.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

sample_limit (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("sample_similarity"=100, "permutation_vimp"=1000).
This parameter can be set for the following data elements: sample_similarity, shap, permutation_vimp, and ice_data.

n_important_features (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("ice_data"=10, "permutation_vimp"=5).

This parameter can be set for the following data elements: ice_data, permutation_vimp, and shap.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.

hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.

A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").

This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.

If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar

may bootstrap the data to create them.

- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`. This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name` Name of the collection.

## Details

Data, such as permutation variable importance and calibration information, is usually collected from a `familiarCollection` object. However, you can also provide one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to the previously mentioned files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

Permutation Variable importance assesses the improvement in model performance due to a feature. For this purpose, the performance of the model is measured as normal, and is measured again with a dataset where the values of the feature in question have been randomly permuted. The difference between both performance measurements is the permutation variable importance.

In familiar, this basic concept is extended in several ways:

- Point estimates of variable importance are based on multiple (21) random permutations. The difference between model performance on the normal dataset and the median performance measurement of the randomly permuted datasets is used as permutation variable importance.
- Confidence intervals for the ensemble model are determined using bootstrap methods.

- Permutation variable importance is assessed for any metric specified using the `metric` argument.
- Permutation variable importance can take into account similarity between features and permute similar features simultaneously.

## Value

A data.table (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_prediction_data`

*Extract and export predicted values.*

---

## Description

Extract and export the values predicted by single and ensemble models in a familiarCollection.

## Usage

```
export_prediction_data(object, dir_path = NULL, export_collection = FALSE, ...)

## S4 method for signature 'ANY'
export_prediction_data(object, dir_path = NULL, export_collection = FALSE, ...)

## S4 method for signature 'familiarCollection'
export_prediction_data(object, dir_path = NULL, export_collection = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `object` | A `familiarCollection` object, or other other objects from which a `familiarCollection` can be extracted. See details for more information. |
| `dir_path` | Path to folder where extracted data should be saved. `NULL` will allow export as a structured list of data.tables. |
| `export_collection` | (*optional*) Exports the collection if TRUE. |
| `...` | Arguments passed on to `extract_predictions`, `as_familiar_collection`<br>`data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.<br>`is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.<br>`cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation. |

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.

A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

aggregate_results (*optional*) Flag that signifies whether results should be aggregated during evaluation. If estimation_type is bias_correction or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.
The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE, , "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

## Details

Data, such as model performance and calibration information, is usually collected from a `familiarCollection` object. However, you can also provide one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

Both single and ensemble predictions are exported.

## Value

A list of data.tables (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_risk_stratification_data` *Extract and export sample risk group stratification and associated tests.*

---

## Description

Extract and export sample risk group stratification and associated tests for data in a familiarCollection.

## Usage

```
export_risk_stratification_data(
  object,
  dir_path = NULL,
  export_strata = TRUE,
  time_range = NULL,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_risk_stratification_data(
  object,
  dir_path = NULL,
  export_strata = TRUE,
  time_range = NULL,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
```

```
export_risk_stratification_data(
  object,
  dir_path = NULL,
  export_strata = TRUE,
  time_range = NULL,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| export_strata | Flag that determines whether the raw data or strata are exported. |
| time_range | Time range for which strata should be created. If NULL, the full time range is used. |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to extract_risk_stratification_data, as_familiar_collection |

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.

- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.
A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

## Details

Data is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together

with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

Three tables are exported in a list:

- `data`: Contains the assigned risk group for a given sample, along with its reported survival time and censoring status.
- `hr_ratio`: Contains the hazard ratio between different risk groups.
- `logrank`: Contains the results from the logrank test between different risk groups.

## Value

A list of data.tables (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_risk_stratification_info` *Extract and export cut-off values for risk group stratification.*

---

## Description

Extract and export cut-off values for risk group stratification by models in a familiarCollection.

## Usage

```
export_risk_stratification_info(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_risk_stratification_info(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_risk_stratification_info(
  object,
  dir_path = NULL,
```

```
  aggregate_results = TRUE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to as_familiar_collection<br>familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.<br>collection_name Name of the collection. |

## Details

Data is usually collected from a familiarCollection object. However, you can also provide one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from object and dir_path are only used if object is not a familiarCollection object, or a path to one.

Stratification cut-off values are determined when creating a model, using one of several methods set by the stratification_method parameter. These values are then used to stratify samples in any new dataset. The available methods are:

- median (default): The median predicted value in the development cohort is used to stratify the samples into two risk groups.
- fixed: Samples are stratified based on the sample quantiles of the predicted values. These quantiles are defined using the stratification_threshold parameter.
- optimised: Use maximally selected rank statistics to determine the optimal threshold (Lausen and Schumacher, 1992; Hothorn et al., 2003) to stratify samples into two optimally separated risk groups.

## Value

A data.table (if dir_path is not provided), or nothing, as all data is exported to csv files.

## References

1. Lausen, B. & Schumacher, M. Maximally Selected Rank Statistics. Biometrics 48, 73 (1992).
2. Hothorn, T. & Lausen, B. On the exact distribution of maximally selected rank statistics. Comput. Stat. Data Anal. 43, 121–137 (2003).

---

`export_sample_similarity` *Extract and export mutual correlation between features.*

---

## Description

Extract and export mutual correlation between features in a familiarCollection.

## Usage

```
export_sample_similarity(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  sample_limit = waiver(),
  sample_cluster_method = waiver(),
  sample_linkage_method = waiver(),
  export_dendrogram = FALSE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_sample_similarity(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  sample_limit = waiver(),
  sample_cluster_method = waiver(),
  sample_linkage_method = waiver(),
  export_dendrogram = FALSE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_sample_similarity(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  sample_limit = waiver(),
```

```
  sample_cluster_method = waiver(),
  sample_linkage_method = waiver(),
  export_dendrogram = FALSE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| aggregate_results | Flag that signifies whether results should be aggregated for export. |
| sample_limit | (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.<br>This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("sample_similarity"=100, "permutation_vimp"=1000).<br>This parameter can be set for the following data elements: sample_similarity, shap, permutation_vimp, and ice_data. |
| sample_cluster_method | The method used to perform clustering based on distance between samples. These are the same methods as for the cluster_method configuration parameter: hclust, agnes, diana and pam.<br>none cannot be used when extracting data for feature expressions.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| sample_linkage_method | The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| export_dendrogram | Add dendrogram in the data element objects. |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to as_familiar_collection, as_data_object<br>familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.<br>collection_name Name of the collection.<br>data A data.frame or data.table, a path to such tables on a local or network drive, or a path to tabular data that may be converted to these formats.<br>check_stringency Specifies stringency of various checks. This is mostly: |

- `strict`: default value used for `summon_familiar`. Thoroughly checks input data. Used internally for checking development data.
- `external_warn`: value used for `extract_data` and related methods. Less stringent checks, but will warn for possible issues. Used internally for checking data for evaluation and explanation.
- `external`: value used for external methods such as `predict`. Less stringent checks, particularly for identifier and outcome columns, which may be completely absent. Used internally for `predict`.

`.no_features_required` Internal flag to signify that data without features is allowed. Default: FALSE (most processing steps require features).

`batch_id_column` (**recommended**) Name of the column containing batch or cohort identifiers. This parameter is required if more than one dataset is provided, or if external validation is performed.
In familiar any row of data is organised by four identifiers:

- The batch identifier `batch_id_column`: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.
- The sample identifier `sample_id_column`: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.
- The series identifier `series_id_column`: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.
- The repetition identifier: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries for the same series of the same sample in the same batch that share the same outcome are encountered.

`sample_id_column` (**recommended**) Name of the column containing sample or subject identifiers. See `batch_id_column` above for more details.
If unset, every row will be identified as a single sample.

`series_id_column` (*optional*) Name of the column containing series identifiers, which distinguish between measurements that are part of a series for a single sample. See `batch_id_column` above for more details.
If unset, rows which share the same batch and sample identifiers but have a different outcome are assigned unique series identifiers.

`development_batch_id` (*optional*) One or more batch or cohort identifiers to constitute data sets for development. Defaults to all, or all minus the identifiers in `validation_batch_id` for external validation. Required if external validation is performed and `validation_batch_id` is not provided.

`validation_batch_id` (*optional*) One or more batch or cohort identifiers to constitute data sets for external validation. Defaults to all data sets except those in `development_batch_id` for external validation, or none if not. Required if `development_batch_id` is not provided.

`outcome_name` (*optional*) Name of the modelled outcome. This name will be used in figures created by `familiar`.

If not set, the column name in `outcome_column` will be used for `binomial`, `multinomial`, and `continuous` outcomes. For other outcomes (`survival` and `competing_risk`) no default is used.

`outcome_column` (**recommended**) Name of the column containing the outcome of interest. May be identified from a formula, if a formula is provided as an argument. Otherwise an error is raised. Note that `survival` and `competing_risk` outcome type outcomes require two columns that indicate the time-to-event or the time of last follow-up and the event status.

`outcome_type` (**recommended**) Type of outcome found in the outcome column. The outcome type determines many aspects of the overall process, e.g. the available variable importance methods and learners, but also the type of assessments that can be conducted to evaluate the resulting models. Implemented outcome types are:

- `binomial`: categorical outcome with 2 levels.
- `multinomial`: categorical outcome with 2 or more levels.
- `continuous`: general continuous numeric outcomes.
- `survival`: survival outcome for time-to-event data.

If not provided, the algorithm will attempt to obtain outcome_type from contents of the outcome column. This may lead to unexpected results, and we therefore advise to provide this information manually.
Note that `competing_risk` survival analysis are not fully supported, and is currently not a valid choice for `outcome_type`. The `count` outcome type was deprecated in version 2.0.0, and superseded by `continuous`.

`class_levels` (*optional*) Class levels for `binomial` or `multinomial` outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column.

`event_indicator` (**recommended**) Indicator for events in `survival` and `competing_risk` analyses. `familiar` will automatically recognise `1`, `true`, `t`, `y` and `yes` as event indicators, including different capitalisations. If this parameter is set, it replaces the default values.

`censoring_indicator` (**recommended**) Indicator for right-censoring in `survival` and `competing_risk` analyses. `familiar` will automatically recognise `0`, `false`, `f`, `n`, `no` as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values.

`competing_risk_indicator` (**recommended**) Indicator for competing risks in `competing_risk` analyses. There are no default values, and if unset, all values other than those specified by the `event_indicator` and `censoring_indicator` parameters are considered to indicate competing risks.

`exclude_features` (*optional*) Feature columns that will be removed from the data set. Cannot overlap with features in `signature`, `novelty_features` or `include_features`.

`include_features` (*optional*) Feature columns that are specifically included in the data set. By default all features are included. Cannot overlap with `exclude_features`, but may overlap `signature`. Features in `signature` and `novelty_features` are always included. If both `exclude_features`

and include_features are provided, include_features takes precedence, provided that there is no overlap between the two.

reference_method (*optional*) Method used to set reference levels for categorical features. There are several options:

- auto (default): Categorical features that are not explicitly set by the user, i.e. columns containing boolean values or characters, use the most frequent level as reference. Categorical features that are explicitly set, i.e. as factors, are used as is.
- always: Both automatically detected and user-specified categorical features have the reference level set to the most frequent level. Ordinal features are not altered, but are used as is.
- never: User-specified categorical features are used as is. Automatically detected categorical features are simply sorted, and the first level is then used as the reference level. This was the behaviour prior to familiar version 1.3.0.

## Details

All parameters aside from object and dir_path are only used if object is not a familiarCollection object, or a path to one.

Sample similarity data can be created from dataObject, or data.table objects. For data.table, see as_data_object for additional arguments.

## Value

A list containing a data.table (if dir_path is not provided), or nothing, as all data is exported to csv files.

---

# export_shap *Extract and export individual conditional expectation data.*

---

## Description

Extract and export individual conditional expectation data.

## Usage

```
export_shap(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  feature_x = NULL,
  feature_y = NULL,
  ...
)
```

```
## S4 method for signature 'familiarCollection'
export_shap(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  feature_x = NULL,
  feature_y = NULL,
  ...
)

## S4 method for signature 'ANY'
export_shap(
  object,
  dir_path = NULL,
  aggregate_results = TRUE,
  export_collection = FALSE,
  feature_x = NULL,
  feature_y = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `familiarCollection` object, or other other objects from which a `familiarCollection` can be extracted. See details for more information. |
| `dir_path` | Path to folder where extracted data should be saved. `NULL` will allow export as a structured list of data.tables. |
| `aggregate_results` | Flag that signifies whether results should be aggregated for export. |
| `export_collection` | (*optional*) Exports the collection if TRUE. |
| `feature_x` | (*optional*) Feature(s) whose SHAP values are used for determining dependence. |
| `feature_y` | (*optional*) Feature(s) whose values are used to show interaction with the feature(s) in `feature_x`. |
| `...` | Arguments passed on to `as_familiar_collection`<br>`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.<br>`collection_name` Name of the collection. |

## Details

Data is usually collected from a `familiarCollection` object. However, you can also provide one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together

with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

## Value

A list of data.tables (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`export_univariate_analysis_data` *Extract and export univariate analysis data of features.*

---

## Description

Extract and export univariate analysis data of features for data in a familiarCollection.

## Usage

```
export_univariate_analysis_data(
  object,
  dir_path = NULL,
  p_adjustment_method = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
export_univariate_analysis_data(
  object,
  dir_path = NULL,
  p_adjustment_method = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
export_univariate_analysis_data(
  object,
  dir_path = NULL,
  p_adjustment_method = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| dir_path | Path to folder where extracted data should be saved. NULL will allow export as a structured list of data.tables. |
| p_adjustment_method | (*optional*) Indicates type of p-value that is shown. One of holm, hochberg, hommel, bonferroni, BH, BY, fdr, none, p_value or q_value for adjusted p-values, uncorrected p-values |
| export_collection | (*optional*) Exports the collection if TRUE. |
| ... | Arguments passed on to extract_univariate_analysis, as_familiar_collection |

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

feature_cluster_method The method used to perform clustering. These are the same methods as for the cluster_method configuration parameter: none, hclust, agnes, diana and pam.
none cannot be used when extracting data regarding mutual correlation or feature expressions.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_linkage_method The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_cluster_cut_method The method used to divide features into separate clusters. The available methods are the same as for the cluster_cut_method configuration parameter: silhouette, fixed_cut and dynamic_cut.
silhouette is available for all cluster methods, but fixed_cut only applies to methods that create hierarchical trees (hclust, agnes and diana).
dynamic_cut requires the dynamicTreeCut package and can only be used with agnes and hclust.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_threshold The threshold level for pair-wise similarity that is required to form feature clusters with the fixed_cut method.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_metric Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and feature_similarity_metric therefore has the same options as cluster_similarity_metric: mcfadden_r2, cox_snell_r2, nagelkerke_r2, spearman, kendall and pearson.

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`icc_type` String indicating the type of intraclass correlation coefficient (`1`, `2` or `3`) that should be used to compute robustness for features in repeated measurements during the evaluation of univariate importance. These types correspond to the types in Shrout and Fleiss (1979). If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name` Name of the collection.

## Details

Data is usually collected from a `familiarCollection` object. However, you can also provide one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to the previous files can also be provided.

All parameters aside from `object` and `dir_path` are only used if `object` is not a `familiarCollection` object, or a path to one.

Univariate analysis includes the computation of p-values, as well as robustness (in case of repeated measurements). p-values are derived from Wald's test.

## Value

A data.table (if `dir_path` is not provided), or nothing, as all data is exported to `csv` files.

---

`familiar` *familiar: Fully Automated Machine Learning with Interpretable Analysis of Results*

---

## Description

End-to-end, automated machine learning package for creating trustworthy and interpretable machine learning models. Familiar supports modelling of regression, categorical and time-to-event (survival) outcomes. Models created using familiar are self-containing, and their use does not require additional information such as baseline survival, feature clustering, or feature transformation and normalisation parameters. In addition, an novelty or out-of-distribution detector is trained simultaneously and contained with every model. Model performance, calibration, risk group stratification, (permutation) variable importance, individual conditional expectation, partial dependence, and more, are assessed automatically as part of the evaluation process and exported in tabular format and plotted, and may also be computed manually using export and plot functions. Where possible, metrics and values obtained during the evaluation process come with confidence intervals.

## Author(s)

**Maintainer**: Alex Zwanenburg <alexander.zwanenburg@nct-dresden.de> (ORCID)

Authors:

- Alex Zwanenburg <alexander.zwanenburg@nct-dresden.de> (ORCID)
- Steffen Löck

Other contributors:

- German Cancer Research Center (DKFZ) [copyright holder]
- Technische Universität Dresden [copyright holder]

## See Also

Useful links:

- https://github.com/oncoray/familiar
- Report bugs at https://github.com/oncoray/familiar/issues

---

`familiarCollection-class`

*Collection of familiar data.*

---

## Description

A familiarCollection object aggregates data from one or more familiarData objects.

## Slots

`name` Name of the collection.

`data_sets` Name of the individual underlying datasets.

`outcome_type` Outcome type for which the collection was created.

`outcome_info` Outcome information object, which contains information concerning the outcome, such as class levels.

`fs_vimp` collected for variable importance methods.

`model_vimp` Variable importance data collected from model-specific algorithms implemented by models created by familiar.

`permutation_vimp` Data collected for permutation variable importance.

`hyperparameters` Hyperparameters collected from created models.

`hyperparameter_data` Additional data concerning hyperparameters. This is currently not used yet.

`required_features` The set of features required for complete reproduction, i.e. with imputation.

`model_features` The set of features that are required for using the model, but without imputation.

learner Learning algorithm(s) used for data in the collection.

vimp_method Variable importance method(s) used for data in the collection.

prediction_data Model predictions for the data in the collection.

confusion_matrix Confusion matrix information for the data in the collection.

decision_curve_data Decision curve analysis data for the data in the collection.

calibration_info Calibration information, e.g. baseline survival in the development cohort.

calibration_data Model calibration data collected from data in the collection.

model_performance Collection of model performance data for data in the collection.

km_info Information concerning risk-stratification cut-off values for data in the collection.

km_data Kaplan-Meier survival data for data in the collection.

auc_data AUC-ROC and AUC-PR data for data in the collection.

ice_data Individual conditional expectation data for data in the collection. Partial dependence data are computed on the fly from these data.

shap_data SHAP values for features included in a model or ensemble of models.

univariate_analysis Univariate analysis results of data in the collection.

feature_expressions Feature expression values for data in the collection.

feature_similarity Feature similarity information for data in the collection.

sample_similarity Sample similarity information for data in the collection.

data_set_labels Labels for the different datasets in the collection. See get_data_set_names and set_data_set_names.

learner_labels Labels for the different learning algorithms used to create the collection. See get_learner_names and set_learner_names.

vimp_method_labels Labels for the different variable importance methods used to create the collection. See get_vimp_method_names and set_vimp_method_names.

feature_labels Labels for the features in this collection. See get_feature_names and set_feature_names.

km_group_labels Labels for the risk strata in this collection. See get_risk_group_names and set_risk_group_names.

class_labels Labels of the response variable. See get_class_names and set_class_names.

project_id Identifier of the project that generated this collection.

familiar_version Version of the familiar package.

familiarCollection objects collect data from one or more familiarData objects. This objects are important, as all plotting and export functions use it. The fact that one can supply familiarModel, familiarEnsemble and familiarData objects as arguments for these methods, is because familiar internally converts these into familiarCollection objects prior to executing the method.

`familiarData-class` *Dataset obtained after evaluating models on a dataset.*

## Description

A familiarData object is created by evaluating familiarEnsemble or familiarModel objects on a dataset. Multiple familiarData objects are aggregated in a familiarCollection object.

## Slots

`name` Name of the dataset, e.g. training or internal validation.

`outcome_type` Outcome type of the data used to create the object.

`outcome_info` Outcome information object, which contains additional information concerning the outcome, such as class levels.

`fs_vimp` Data collected for variable importance methods.

`model_vimp` Variable importance data collected from model-specific algorithms implemented by models created by familiar.

`permutation_vimp` Data collected for permutation variable importance.

`hyperparameters` Hyperparameters collected from created models.

`hyperparameter_data` Additional data concerning hyperparameters. This is currently not used yet.

`required_features` The set of features required for complete reproduction, i.e. with imputation.

`model_features` The set of features that are required for using the model or ensemble of models, but without imputation.

`learner` Learning algorithm used to create the model or ensemble of models.

`vimp_method` Method used to determine variable importance for the model or ensemble of models.

`pooling_table` Run table for the data underlying the familiarData object. Used internally.

`prediction_data` Model predictions for a model or ensemble of models for the underlying dataset.

`confusion_matrix` Confusion matrix for a model or ensemble of models, based on the underlying dataset.

`decision_curve_data` Decision curve analysis data for a model or ensemble of models, based on the underlying dataset.

`calibration_info` Calibration information, e.g. baseline survival in the development cohort.

`calibration_data` Calibration data for a model or ensemble of models, based on the underlying dataset.

`model_performance` Model performance data for a model or ensemble of models, based on the underlying dataset.

`km_info` Information concerning risk-stratification cut-off values..

`km_data` Kaplan-Meier survival data for a model or ensemble of models, based on the underlying dataset.

`auc_data` AUC-ROC and AUC-PR data for a model or ensemble of models, based on the underlying dataset.

`ice_data` Individual conditional expectation data for features included in a model or ensemble of models, based on the underlying dataset. Partial dependence data are computed on the fly from these data.

`shap_data` SHAP values for features included in a model or ensemble of models.

`univariate_analysis` Univariate analysis of the underlying dataset.

`feature_expressions` Feature expression values of the underlying dataset.

`feature_similarity` Feature similarity information of the underlying dataset.

`sample_similarity` Sample similarity information of the underlying dataset.

`project_id` Identifier of the project that generated the familiarData object.

`familiar_version` Version of the familiar package.

familiarData objects contain information obtained by evaluating a single model or single ensemble of models on a dataset.

---

`familiarDataElement-class`

*Data container for evaluation data.*

---

## Description

Most attributes of the familiarData object are objects of the familiarDataElement class. This (super-)class is used to allow for standardised aggregation and processing of evaluation data.

## Slots

`data` Evaluation data, typically a data.table or list.

`identifiers` Identifiers of the data, e.g. the generating model name, learner, etc.

`detail_level` Sets the level at which results are computed and aggregated.

- `ensemble`: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.

Some child classes do not use this parameter.

`estimation_type` Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

Some child classes do not use this parameter.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.

`bootstrap_ci_method` Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

`value_column` Identifies column(s) in the `data` attribute presenting values.

`grouping_column` Identifies column(s) in the `data` attribute presenting identifier columns for grouping during aggregation. Familiar will automatically assign items from the `identifiers` attribute to the data and this attribute when combining multiple familiarDataElements of the same (child) class.

`is_aggregated` Defines whether the object was aggregated.

## References

1. Efron, B. & Hastie, T. Computer Age Statistical Inference. (Cambridge University Press, 2016).

`familiarEnsemble-class`
*Ensemble of familiar models.*

## Description

A familiarEnsemble object contains one or more familiarModel objects.

## Slots

`name` Name of the familiarEnsemble object.

`model_list` List of attached familiarModel objects, or paths to these objects. Familiar attaches familiarModel objects when required.

`outcome_type` Outcome type of the data used to create the object.

`outcome_info` Outcome information object, which contains additional information concerning the outcome, such as class levels.

`data_column_info` Data information object containing information regarding identifier column names and outcome column names.

`learner` Learning algorithm used to create the models in the ensemble.

`vimp_method` Method used to determine variable importance for the models in the ensemble.

`feature_info` List of objects containing feature information, e.g., name, class levels, transformation, normalisation and clustering parameters.

`required_features` The set of features required for complete reproduction, i.e. with imputation.

`model_features` The combined set of features that is used to train the models in the ensemble,

`novelty_features` The combined set of features that is used to train all novelty detectors in the ensemble.

`data_id` Internal identifier for the dataset used to train or evaluate the ensemble.

`run_id` Internal identifier for the specific subset of the dataset used used to train or evaluate the ensemble.

`run_table` Run table for the data used to train the ensemble. Used internally.

`calibration_info` Calibration information, e.g. baseline survival in the development cohort.

`model_dir_path` Path to folder containing the familiarModel objects. Can be updated using the `update_model_dir_path` method.

`auto_detach` Flag used to determine whether models should be detached from the model after use, or not. Used internally.

`settings` A copy of the evaluation configuration parameters used at model creation. These are used as default parameters when evaluating the ensemble to create a familiarData object.

`project_id` Identifier of the project that generated the underlying familiarModel object(s).

`familiar_version` Version of the familiar package.

`familiarHyperparameterLearner-class`
*Hyperparameter learner.*

## Description

A familiarHyperparameterLearner object is a self-contained model that can be applied to predict optimisation scores for a set of hyperparameters.

## Details

Hyperparameter learners are used to infer the optimisation score for sets of hyperparameters. These are then used to either infer utility using acquisition functions or to generate summary scores to identify the optimal model.

## Slots

`name` Name of the familiarHyperparameterLearner object.

`learner` Algorithm used to create the hyperparameter learner.

`target_learner` Algorithm for which the hyperparameters are being learned.

`target_outcome_type` Outcome type of the learner for which hyperparameters are being modeled. Used to determine the target hyperparameters.

`optimisation_metric` One or metrics used to generate the optimisation score.

`optimisation_function` Function used to generate the optimisation score.

`model` The actual model trained using the specific algorithm, e.g. a isolation forest from the `isotree` package.

`target_hyperparameters` The names of the hyperparameters that are used to train the hyperparameter learner.

`project_id` Identifier of the project that generated the familiarHyperparameterLearner object.

`familiar_version` Version of the familiar package.

`package` Name of package(s) required to executed the hyperparameter learner itself, e.g. `laGP`.

`package_version` Version of the packages mentioned in the `package` attribute.

## familiarMetric-class *Model performance metric.*

### Description

Superclass for model performance objects.

### Slots

`metric` Performance metric.

`outcome_type` Type of outcome being predicted.

`name` Name of the performance metric.

`value_range` Range of the performance metric. Can be half-open.

`baseline_value` Value of the metric for trivial models, e.g. models that always predict the median value, the majority class, or the mean hazard, etc.

`higher_better` States whether higher metric values correspond to better predictive model performance (e.g. accuracy) or not (e.g. root mean squared error).

## familiarModel-class *Familiar model.*

### Description

A familiarModel object is a self-contained model that can be applied to generate predictions for a dataset. familiarModel objects form the parent class of learner-specific child classes.

### Slots

`name` Name of the familiarModel object.

`model` The actual model trained using a specific algorithm, e.g. a random forest from the `ranger` package, or a LASSO model from `glmnet`.

`outcome_type` Outcome type of the data used to create the object.

`outcome_info` Outcome information object, which contains additional information concerning the outcome, such as class levels.

`feature_info` List of objects containing feature information, e.g., name, class levels, transformation, normalisation and clustering parameters.

`data_column_info` Data information object containing information regarding identifier column names and outcome column names.

`hyperparameters` Set of hyperparameters used to train the model.

`hyperparameter_data` Information generated during hyperparameter optimisation.

`calibration_model` One or more models used to recalibrate the model output. Currently only used by some models.

`novelty_detector` A familiarNoveltyDetector object that can be used to detect out-of-distribution samples.

`learner` Learning algorithm used to create the model.

`vimp_method` Method used to determine variable importance for the model.

`vimp_table` Variable importance table or list of variable importance tables for the model.

`vimp_aggregation_method` Method used for aggregating variable importance tables if more than one is present.)

`vimp_rank_threshold` Threshold used for some variable importance aggregation methods.

`required_features` The set of features required for complete reproduction, i.e. with imputation.

`model_features` The set of features that is used to train the model,

`novelty_features` The set of features that is used to train all novelty detectors in the ensemble.

`calibration_info` Calibration information, e.g. baseline survival in the development cohort.

`km_info` Data concerning stratification into risk groups.

`data_id` Internal identifier for the dataset used to train the model.

`run_id` Internal identifier for the specific subset of the dataset used used to train the model.

`run_table` Run table for the data used to train the model. Used internally.

`settings` A copy of the evaluation configuration parameters used at model creation. These are used as default parameters when evaluating the model (technically, familiarEnsemble) to create a familiarData object.

`is_trimmed` Flag that indicates whether the model, stored in the `model` slot, has been trimmed.

`trimmed_function` List of functions whose output has been captured prior to trimming the model.

`messages` List of warning and error messages generated during training.

`project_id` Identifier of the project that generated the familiarModel object.

`familiar_version` Version of the familiar package.

`package` Name of package(s) required to executed the model itself, e.g. `ranger` or `glmnet`.

`package_version` Version of the packages mentioned in the `package` attribute.

---

`familiarNoveltyDetector-class`
*Novelty detector.*

---

## Description

A familiarNoveltyDetector object is a self-contained model that can be applied to generate out-of-distribution predictions for instances in a dataset.

## Details

Note that these objects do not contain any data concerning outcome, as this not relevant for (prospective) out-of-distribution detection.

## Slots

`name` Name of the familiarNoveltyDetector object.

`model` The actual novelty detector trained using a specific algorithm, e.g. a isolation forest from the `isotree` package.

`feature_info` List of objects containing feature information, e.g., name, class levels, transformation, normalisation and clustering parameters.

`data_column_info` Data information object containing information regarding identifier column names.

`hyperparameters` Set of hyperparameters used to train the detector.

`hyperparameter_data` Information generated during hyperparameter optimisation. Currently not used.

`calibration_model` Model used to convert raw output to statistical probability of being out-of-distribution. Currently not used.

`learner` Learning algorithm used to create the novelty detector.

`vimp_method` Method used to determine variable importance for the novelty detector.

`vimp_table` Variable importance table or list of variable importance tables for the model.

`vimp_aggregation_method` Method used for aggregating variable importance tables if more than one is present.)

`vimp_rank_threshold` Threshold used for some variable importance aggregation methods.

`required_features` The set of features required for complete reproduction, i.e. with imputation.

`model_features` The set of features that is used to train the detector.

`data_id` Internal identifier for the dataset used to train the detector.

`run_id` Internal identifier for the specific subset of the dataset used used to train the detector.

`run_table` Run table for the data used to train the detector. Used internally.

`is_trimmed` Flag that indicates whether the detector, stored in the `model` slot, has been trimmed.

`trimmed_function` List of functions whose output has been captured prior to trimming the model.

`messages` List of warning and error messages generated during training.

`project_id` Identifier of the project that generated the familiarNoveltyDetector object.

`familiar_version` Version of the familiar package.

`package` Name of package(s) required to executed the detector itself, e.g. `isotree`.

`package_version` Version of the packages mentioned in the `package` attribute.

## familiarVimpMethod-class *Variable importance method object.*

### Description

The familiarVimpMethod class is the parent class for all variable importance methods in familiar.

### Slots

`outcome_type` Outcome type of the data to be evaluated using the object.

`hyperparameters` Set of hyperparameters for the variable importance method.

`vimp_method` The character string indicating the variable importance method.

`vimp_aggregation_method` Method used for aggregating variable importance tables if more than one is present.)

`vimp_rank_threshold` Threshold used for some variable importance aggregation methods.

`multivariate` Flags whether the variable importance method is multivariate vs. univariate.

`outcome_info` Outcome information object, which contains additional information concerning the outcome, such as class levels.

`feature_info` List of objects containing feature information, e.g., name, class levels, transformation, normalisation and clustering parameters.

`required_features` The set of features to be assessed by the variable importance method.

`package` Name of the package(s) required to execute the variable importance method.

`run_table` Run table for the data to be assessed by the variable importance method. Used internally.

`project_id` Identifier of the project that generated the familiarVimpMethod object.

`familiar_version` Version of the familiar package used to create this object.

## featureInfo-class *Feature information object.*

### Description

A featureInfo object contains information for a single feature. This information is used to check data prospectively for consistency and for data preparation. These objects are, for instance, attached to a familiarModel object so that data can be pre-processed in the same way as the development data.

## Slots

`name` Name of the feature, which by default is the column name of the feature.

`set_descriptor` Character string describing the set to which the feature belongs. Currently not used.

`feature_type` Describes the feature type, i.e. `factor` or `numeric`.

`levels` The class levels of categorical features. This is used to check prospective datasets.

`ordered` Specifies whether the

`distribution` Five-number summary (numeric) or class frequency (categorical).

`data_id` Internal identifier for the dataset used to derive the feature information.

`run_id` Internal identifier for the specific subset of the dataset used to derive the feature information.

`in_signature` Specifies whether the feature is included in the model signature.

`in_novelty` Specifies whether the feature is included in the novelty detector.

`removed` Specifies whether the feature was removed during pre-processing.

`removed_unknown_type` Specifies whether the feature was removed during pre-processing because the type was neither factor nor numeric..

`removed_missing_values` Specifies whether the feature was removed during pre-processing because it contained too many missing values.

`removed_no_variance` Specifies whether the feature was removed during pre-processing because it did not contain more than 1 unique value.

`removed_low_variance` Specifies whether the feature was removed during pre-processing because the variance was too low. Requires applying `low_variance` as a `filter_method`.

`removed_low_robustness` Specifies whether the feature was removed during pre-processing because it lacks robustness. Requires applying `robustness` as a `filter_method`, as well as repeated measurement.

`removed_low_importance` Specifies whether the feature was removed during pre-processing because it lacks relevance. Requires applying `univariate_test` as a `filter_method`.

`fraction_missing` Specifies the fraction of missing values.

`robustness` Specifies robustness of the feature, if measured.

`univariate_importance` Specifies the univariate p-value of the feature, if measured.

`transformation_parameters` Details parameters for power transformation of numeric features.

`normalisation_parameters` Details parameters for (global) normalisation of numeric features.

`batch_normalisation_parameters` Details parameters for batch normalisation of numeric features.

`imputation_parameters` Details parameters or models for imputation of missing values.

`cluster_parameters` Details parameters for forming clusters with other features.

`required_features` Details features required for clustering or imputation.

`project_id` Identifier of the project that generated this collection.

`familiar_version` Version of the familiar package.

## featureInfoParameters-class

*Feature information parameters object.*

### Description

A featureInfo object contains information for a single feature. Some information, for example concerning clustering and transformation contains various parameters that allow for applying the data transformation correctly. These are stored in featureInfoParameters objects.

### Details

featureInfoParameters is normally a parent class for specific classes, such as featureInfoParameter-sTransformation.

### Slots

`name` Name of the feature, which by default is the column name of the feature. Typically used to correctly assign the data.

`complete` Flags whether the parameters have been completely set.

`familiar_version` Version of the familiar package.

## get_class_names,familiarCollection-method

*Get outcome class labels*

### Description

Outcome classes in familiarCollection objects can have custom names for export and plotting. This function retrieves the currently assigned names.

### Usage

```
## S4 method for signature 'familiarCollection'
get_class_names(x)
```

### Arguments

| | |
|---|---|
| `x` | A familiarCollection object. |

### Details

Labels convert internal class names to the requested label at export or when plotting. Labels can be changed using the `set_class_names` method.

### Value

An ordered array of class labels.

### See Also

- familiarCollection for information concerning the familiarCollection class.
- set_class_names for updating the name and ordering of classes.

---

## get_data_set_names,familiarCollection-method
*Get current name of datasets*

---

### Description

Datasets in familiarCollection objects can have custom names for export and plotting. This function retrieves the currently assigned names.

### Usage

```
## S4 method for signature 'familiarCollection'
get_data_set_names(x)
```

### Arguments

| | |
|---|---|
| x | A familiarCollection object. |

### Details

Labels convert internal naming of data sets to the requested label at export or when plotting. Labels can be changed using the set_data_set_names method.

### Value

An ordered array of dataset name labels.

### See Also

- familiarCollection for information concerning the familiarCollection class.
- set_data_set_names for updating the name of datasets and their ordering.

## get_feature_names,familiarCollection-method

*Get current feature labels*

### Description

Features in familiarCollection objects can have custom names for export and plotting. This function retrieves the currently assigned names.

### Usage

```
## S4 method for signature 'familiarCollection'
get_feature_names(x)
```

### Arguments

| | |
|---|---|
| x | A familiarCollection object. |

### Details

Labels convert internal naming of features to the requested label at export or when plotting. Labels can be changed using the `set_feature_names` method.

### Value

An ordered array of feature labels.

### See Also

- familiarCollection for information concerning the familiarCollection class.
- `set_feature_names` for updating the name and ordering of features.

## get_learner_names,familiarCollection-method

*Get current learner name labels*

### Description

Learners in familiarCollection objects can have custom names for export and plotting. This function retrieves the currently assigned names.

### Usage

```
## S4 method for signature 'familiarCollection'
get_learner_names(x)
```

### Arguments

| | |
|---|---|
| x | A familiarCollection object. |

### Details

Labels convert internal naming of learners to the requested label at export or when plotting. Labels can be changed using the `set_learner_names` method.

### Value

An ordered array of learner name labels.

### See Also

- familiarCollection for information concerning the familiarCollection class.
- `set_learner_names` for updating the name of learners and their ordering.

---

## `get_risk_group_names,familiarCollection-method` *Get current risk group labels*

---

### Description

Risk groups in familiarCollection objects can have custom names for export and plotting. This function retrieves the currently assigned names.

### Usage

```
## S4 method for signature 'familiarCollection'
get_risk_group_names(x)
```

### Arguments

| | |
|---|---|
| x | A familiarCollection object. |

### Details

Labels convert internal naming of risk groups to the requested label at export or when plotting. Labels can be changed using the `set_risk_group_names` method.

### Value

An ordered array of risk group labels.

### See Also

- familiarCollection for information concerning the familiarCollection class.
- `set_risk_group_names` for updating the name and ordering of risk groups.

## get_vimp_method_names,familiarCollection-method *Get current variable importance method name labels*

### Description

Variable importance methods in familiarCollection objects can have custom names for export and plotting. This function retrieves the currently assigned names.

### Usage

```
## S4 method for signature 'familiarCollection'
get_vimp_method_names(x)
```

### Arguments

| | |
|---|---|
| x | A familiarCollection object. |

### Details

Labels convert internal naming of variable importance methods to the requested label at export or when plotting. Labels can be changed using the set_vimp_method_names method.

### Value

An ordered array of variable importance method name labels.

### See Also

- familiarCollection for information concerning the familiarCollection class.
- set_vimp_method_names for updating the name of variable importance methods and their ordering.

## get_vimp_table *Extract variable importance table.*

### Description

This method retrieves and parses variable importance tables from their respective vimpTable objects.

## Usage

```
get_vimp_table(x, state = "ranked", ...)

## S4 method for signature 'list'
get_vimp_table(x, state = "ranked", ...)

## S4 method for signature 'character'
get_vimp_table(x, state = "ranked", ...)

## S4 method for signature 'vimpTable'
get_vimp_table(x, state = "ranked", ...)

## S4 method for signature 'NULL'
get_vimp_table(x, state = "ranked", ...)

## S4 method for signature 'experimentData'
get_vimp_table(x, state = "ranked", ...)

## S4 method for signature 'familiarModel'
get_vimp_table(x, state = "ranked", data = NULL, as_object = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | Variable importance (`vimpTable`) object, a list thereof, or one or more paths to these objects. This method extracts the variable importance table from such objects. |
| state | State of the returned variable importance table. This affects what contents are shown, and in which format. The variable importance table can be returned with the following states:<br>• `initial`: initial state, directly after the variable importance table is filled. The returned variable importance table shows the raw, un-processed data.<br>• `decoded`: depending on the variable importance method, the initial variable importance table may contain the scores of individual contrasts for categorical variables. When decoded, scores from all contrasts are aggregated to a single score for each feature.<br>• `declustered`: variable importance is determined from fully processed features, which includes clustering. This means that a single feature in the variable importance table may represent multiple original features. When a variable importance table has been declustered, all clusters have been turned into their constituent features.<br>• `ranked` (default): The scores have been used to create ranks, with lower ranks indicating better features.<br>Internally, the variable importance table will go through each state, i.e. an variable importance table in the initial state will be decoded, declustered and then ranked prior to returning the variable importance table. |
| ... | Unused arguments. |

| | |
|---|---|
| data | Internally used argument for use with familiarModel objects. |
| as_object | Internally used argument for use with familiarModel objects. |

## Value

A data.table with variable importance scores and, with state="ranked", the respective ranks.

---

# get_xml_config *Create an empty xml configuration file*

---

## Description

This function creates an empty configuration xml file in the directory specified by dir_path. This provides an alternative to the use of input arguments for familiar.

## Usage

```
get_xml_config(dir_path)
```

## Arguments

| | |
|---|---|
| dir_path | Path to the directory where the configuration file should be created. The directory should exist, and no file named config.xml should be present. |

## Value

Nothing. A file named config.xml is created in the directory indicated by dir_path.

## Examples

```
## Not run:
# Creates a config.xml file in the working directory
get_xml_config(dir_path=getwd())

## End(Not run)
```

## outcomeInfo-class *Outcome information object.*

### Description

An outcome information object stores data concerning an outcome. This is used to prospectively check data.

### Slots

`name` Name of the outcome, inherited from the original column name by default.

`outcome_type` Type of outcome.

`outcome_column` Name of the outcome column in data.

`levels` Specifies class levels of categorical outcomes.

`ordered` Specifies whether categorical outcomes are ordered.

`reference` Class level used as reference.

`time` Maximum time, as set by the `time_max` configuration parameter.

`censored` Censoring indicators for survival outcomes.

`event` Event indicators for survival outcomes.

`competing_risk` Indicators for competing risks in survival outcomes.

`distribution` Five-number summary (numeric outcomes), class frequency (categorical outcomes), or survival distributions.

`data_id` Internal identifier for the dataset used to derive the outcome information.

`run_id` Internal identifier for the specific subset of the dataset used to derive the outcome information.

`transformation_parameters` Parameters used for transforming a numeric outcomes. Currently unused.

`normalisation_parameters` Parameters used for normalising numeric outcomes. Currently unused.

`familiar_version` Version of the familiar package used to create this object.

## plot_auc_precision_recall_curve *Plot the precision-recall curve.*

### Description

This method creates precision-recall curves based on data in a familiarCollection object.

## Usage

```
plot_auc_precision_recall_curve(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_auc_precision_recall_curve(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_n_breaks = 5L,
```

```
  x_breaks = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_auc_precision_recall_curve(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | familiarCollection object, or one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to such files can also be provided. |

Additionally, some familiarData objects can be created from prediction tables (familiarDataElementPredictionTable). Other familiarData objects can be created from data (dataObject, or data.table). Please check *details* for more information.

draw (*optional*) Draws the plot if TRUE.

dir_path (*optional*) Path to the directory where the plots of receiver operating characteristic curves are saved to. Output is saved in the performance subdirectory. If NULL no figures are saved, but are returned instead.

split_by (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

color_by (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

ggtheme (*optional*) ggplot theme to use for plotting.

discrete_palette (*optional*) Palette for colouring the plot elements according to the groupings indicated by the color_by argument (if any). familiar has a default palette. Other palettes are supported by paletteer, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

x_n_breaks (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

x_breaks (*optional*) Break points on the x-axis of the plot.

y_n_breaks (*optional*) Number of breaks to show on the y-axis of the plot. y_n_breaks is used to determine the y_breaks argument in case it is unset.

y_breaks (*optional*) Break points on the y-axis of the plot.

conf_int_style (*optional*) Confidence interval style. See details for allowed styles.

conf_int_alpha (*optional*) Alpha value to determine transparency of confidence intervals or, alternatively, other plot elements with which the confidence interval overlaps. Only values between 0.0 (fully transparent) and 1.0 (fully opaque) are allowed.

width (*optional*) Width of the plot. A default value is derived from the number of facets.

height (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection (*optional*) Exports the collection if TRUE.

... Arguments passed on to as_familiar_collection, ggplot2::ggsave

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

## Details

This function generates area under the precision-recall curve plots.

Available splitting variables are: vimp_method, learner, data_set and positive_class. By default, the data is split by vimp_method and learner, with faceting by data_set and colouring by positive_class.

Available palettes for discrete_palette are those listed by grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. If not specified, a default palette based on palettes in Tableau are used. You may

also specify your own palette by using colour names listed by `grDevices::colors()` or through hexadecimal RGB strings.

Bootstrap confidence intervals of the ROC curve (if present) can be shown using various styles set by `conf_int_style`:

- `ribbon` (default): confidence intervals are shown as a ribbon with an opacity of `conf_int_alpha` around the point estimate of the ROC curve.
- `step` (default): confidence intervals are shown as a step function around the point estimate of the ROC curve.
- `none`: confidence intervals are not shown. The point estimate of the ROC curve is shown as usual.

Labelling methods such as `set_vimp_method_names` or `set_data_set_names` can be applied to the `familiarCollection` object to update labels, and order the output in the figure.

## Value

`NULL` or list of plot objects, if `dir_path` is `NULL`.

---

`plot_auc_roc_curve` *Plot the receiver operating characteristic curve.*

---

## Description

This method creates receiver operating characteristic curves based on data in a familiarCollection object.

## Usage

```
plot_auc_roc_curve(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
```

```
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_auc_roc_curve(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_auc_roc_curve(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
```

```
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | familiarCollection object, or one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some familiarData objects can be created from prediction tables (familiarDataElementPredictionTable). Other familiarData objects can be created from data (dataObject, or data.table). Please check *details* for more information. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where the plots of receiver operating characteristic curves are saved to. Output is saved in the performance subdirectory. If NULL no figures are saved, but are returned instead. |
| split_by | (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables. |
| color_by | (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables. |
| facet_by | (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The |

variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols
: (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

ggtheme
: (*optional*) ggplot theme to use for plotting.

discrete_palette
: (*optional*) Palette for colouring the plot elements according to the groupings indicated by the color_by argument (if any). familiar has a default palette. Other palettes are supported by paletteer, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label
: (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label
: (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label
: (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title
: (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title
: (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption
: (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

x_n_breaks
: (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

x_breaks
: (*optional*) Break points on the x-axis of the plot.

y_n_breaks
: (*optional*) Number of breaks to show on the y-axis of the plot. y_n_breaks is used to determine the y_breaks argument in case it is unset.

y_breaks
: (*optional*) Break points on the y-axis of the plot.

conf_int_style
: (*optional*) Confidence interval style. See details for allowed styles.

conf_int_alpha
: (*optional*) Alpha value to determine transparency of confidence intervals or, alternatively, other plot elements with which the confidence interval overlaps. Only values between 0.0 (fully transparent) and 1.0 (fully opaque) are allowed.

width
: (*optional*) Width of the plot. A default value is derived from the number of facets.

height
: (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units
: (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection
: (*optional*) Exports the collection if TRUE.

...
: Arguments passed on to as_familiar_collection, ggplot2::ggsave, extract_auc_data

  familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where

performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.

- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.

`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.

A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`. This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.

If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`. This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`aggregate_results` (*optional*) Flag that signifies whether results should be aggregated during evaluation. If `estimation_type` is `bias_correction` or `bc`, aggregation leads to a single bias-corrected estimate. If `estimation_type` is `bootstrap_confidence_interval` or `bci`, aggregation leads to a single

bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if `estimation_type` is `point`.
The default value is equal to `TRUE` except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with `detail_level` and `estimation_type`, a non-default `aggregate_results` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"=TRUE, , "model_performance"=FALSE)`. This parameter exists for the same elements as `estimation_type`.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

## Details

This function generates area under the ROC curve plots.

Available splitting variables are: `vimp_method`, `learner`, `data_set` and `positive_class`. By default, the data is split by `vimp_method` and `learner`, with faceting by `data_set` and colouring by `positive_class`.

Bootstrap confidence intervals of the ROC curve (if present) can be shown using various styles set by `conf_int_style`:

- `ribbon` (default): confidence intervals are shown as a ribbon with an opacity of `conf_int_alpha` around the point estimate of the ROC curve.
- `step` (default): confidence intervals are shown as a step function around the point estimate of the ROC curve.
- `none`: confidence intervals are not shown. The point estimate of the ROC curve is shown as usual.

Labelling methods such as `set_vimp_method_names` or `set_data_set_names` can be applied to the `familiarCollection` object to update labels, and order the output in the figure.

## Value

`NULL` or list of plot objects, if `dir_path` is `NULL`.

`plot_calibration_data` *Plot calibration figures.*

## Description

This method creates calibration plots from calibration data stored in a familiarCollection object. For this figures, the expected (predicted) values are plotted against the observed values. A well-calibrated model should be close to the identity line.

## Usage

```
plot_calibration_data(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  show_density = TRUE,
  show_calibration_fit = TRUE,
  show_goodness_of_fit = TRUE,
  density_plot_height = grid::unit(1, "cm"),
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

```
## S4 method for signature 'ANY'
plot_calibration_data(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  show_density = TRUE,
  show_calibration_fit = TRUE,
  show_goodness_of_fit = TRUE,
  density_plot_height = grid::unit(1, "cm"),
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_calibration_data(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
```

```
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  show_density = TRUE,
  show_calibration_fit = TRUE,
  show_goodness_of_fit = TRUE,
  density_plot_height = grid::unit(1, "cm"),
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | `familiarCollection` object, or one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some `familiarData` objects can be created from prediction tables (`familiarDataElementPredictionTable`). Other `familiarData` objects can be created from data (`dataObject`, or `data.table`). Please check *details* for more information. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where created calibration plots are saved to. Output is saved in the `calibration` subdirectory. If `NULL` no figures are saved, but are returned instead. |
| split_by | (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables. |

color_by (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

ggtheme (*optional*) ggplot theme to use for plotting.

discrete_palette (*optional*) Palette for colouring the plot elements according to the groupings indicated by the color_by argument (if any). familiar has a default palette. Other palettes are supported by paletteer, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

x_label_shared (*optional*) Sharing of x-axis labels between facets. One of three values:

- overall: A single label is placed at the bottom of the figure. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).
- column: A label is placed at the bottom of each column. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).
- individual: A label is placed below each facet plot. Tick text is kept.

y_label (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

y_label_shared (*optional*) Sharing of y-axis labels between facets. One of three values:

- overall: A single label is placed to the left of the figure. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).
- row: A label is placed to the left of each row. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).
- individual: A label is placed below each facet plot. Tick text is kept.

legend_label (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

x_range (*optional*) Value range for the x-axis.

x_n_breaks (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

`x_breaks` (*optional*) Break points on the x-axis of the plot.

`y_range` (*optional*) Value range for the y-axis.

`y_n_breaks` (*optional*) Number of breaks to show on the y-axis of the plot. `y_n_breaks` is used to determine the `y_breaks` argument in case it is unset.

`y_breaks` (*optional*) Break points on the y-axis of the plot.

`conf_int_style` (*optional*) Confidence interval style. See details for allowed styles.

`conf_int_alpha` (*optional*) Alpha value to determine transparency of confidence intervals or, alternatively, other plot elements with which the confidence interval overlaps. Only values between 0.0 (fully transparent) and 1.0 (fully opaque) are allowed.

`show_density` (*optional*) Show point density in top margin of the figure. If `color_by` is set, this information will not be shown.

`show_calibration_fit`
(*optional*) Specifies whether the calibration in the large and calibration slope are annotated in the plot. If `color_by` is set, this information will not be shown.

`show_goodness_of_fit`
(*optional*) Specifies whether a the results of goodness of fit tests are annotated in the plot. If `color_by` is set, this information will not be shown.

`density_plot_height`
(*optional*) Height of the density plot. The height is 1.5 cm by default. Height is expected to be grid unit (see `grid::unit`), which also allows for specifying relative heights. Will be ignored if `show_density` is `FALSE`.

`width` (*optional*) Width of the plot. A default value is derived from the number of facets.

`height` (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

`units` (*optional*) Plot size unit. Either `cm` (default), `mm` or `in`.

`export_collection`
(*optional*) Exports the collection if TRUE.

`...` Arguments passed on to `as_familiar_collection`, `ggplot2::ggsave`, `extract_calibration_data`

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name` Name of the collection.

`device` Device to use. Can either be a device function (e.g. `png`), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If `NULL` (default), the device is guessed based on the `filename` extension.

`scale` Multiplicative scaling factor.

`dpi` Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

`limitsize` When `TRUE` (the default), `ggsave()` will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.
A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

aggregate_results (*optional*) Flag that signifies whether results should be aggregated during evaluation. If estimation_type is bias_correction or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.
The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE,

, "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

## Details

This function generates a calibration plot for each model in each dataset. Any data used for calibration (e.g. baseline survival) is obtained during model creation.

Available splitting variables are: vimp_method, learner, data_set and evaluation_time (survival analysis only) and positive_class (multinomial endpoints only). By default, separate figures are created for each combination of vimp_method and learner, with facetting by data_set.

Calibration in survival analysis is performed at set time points so that survival probabilities can be computed from the model, and compared with observed survival probabilities. This is done differently depending on the underlying model. For Cox partial hazards regression models, the base survival (of the development samples) are used, whereas accelerated failure time models (e.g. Weibull) and survival random forests can be used to directly predict survival probabilities at a given time point. For survival analysis, evaluation_time is an additional facet variable (by default).

Calibration for multinomial endpoints is performed in a one-against-all manner. This yields calibration information for each individual class of the endpoint. For such endpoints, positive_class is an additional facet variable (by default).

Calibration plots have a density plot in the margin, which shows the density of the plotted points, ordered by the expected probability or value. For binomial and multinomial outcomes, the density for positive and negative classes are shown separately. Note that this information is only provided in when color_by is not used as a splitting variable (i.e. one calibration plot per facet).

Calibration plots are annotated with the intercept and the slope of a linear model fitted to the sample points. A well-calibrated model has an intercept close to 0.0 and a slope of 1.0. Intercept and slope are shown with their respective 95% confidence intervals. In addition, goodness-of-fit tests may be shown. For most endpoints these are based on the Hosmer-Lemeshow (HL) test, but for survival endpoints both the Nam-D'Agostino (ND) and the Greenwood-Nam-D'Agostino (GND) tests are shown. Note that this information is only annotated when color_by is not used as a splitting variable (i.e. one calibration plot per facet).

Labelling methods such as set_risk_group_names or set_data_set_names can be applied to the familiarCollection object to update labels, and order the output in the figure.

## Value

NULL or list of plot objects, if dir_path is NULL.

## References

1. Hosmer, D. W., Hosmer, T., Le Cessie, S. & Lemeshow, S. A comparison of goodness-of-fit tests for the logistic regression model. Stat. Med. 16, 965–980 (1997).
2. D'Agostino, R. B. & Nam, B.-H. Evaluation of the Performance of Survival Analysis Models: Discrimination and Calibration Measures. in Handbook of Statistics vol. 23 1–25 (Elsevier, 2003).
3. Demler, O. V., Paynter, N. P. & Cook, N. R. Tests of calibration and goodness-of-fit in the survival setting. Stat. Med. 34, 1659–1680 (2015).

---

# plot_confusion_matrix *Plot confusion matrix.*

---

## Description

This method creates confusion matrices based on data in a familiarCollection object.

## Usage

```
plot_confusion_matrix(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  rotate_x_tick_labels = waiver(),
  show_alpha = TRUE,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

```
## S4 method for signature 'ANY'
plot_confusion_matrix(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  rotate_x_tick_labels = waiver(),
  show_alpha = TRUE,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_confusion_matrix(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  rotate_x_tick_labels = waiver(),
  show_alpha = TRUE,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
```

)

## Arguments

| | |
|---|---|
| object | familiarCollection object, or one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some familiarData objects can be created from prediction tables (familiarDataElementPredictionTable). Other familiarData objects can be created from data (dataObject, or data.table). Please check *details* for more information. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where created confusion matrixes are saved to. Output is saved in the performance subdirectory. If NULL no figures are saved, but are returned instead. |
| split_by | (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables. |
| facet_by | (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables. |
| facet_wrap_cols | (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead. |
| ggtheme | (*optional*) ggplot theme to use for plotting. |
| discrete_palette | (*optional*) Palette Palette for colouring the cells of the confusion matrix. The colour depends on whether each cell of the confusion matrix is on the diagonal (observed outcome matched expected outcome) or not. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings. |
| x_label | (*optional*) Label to provide to the x-axis. If NULL, no label is shown. |
| y_label | (*optional*) Label to provide to the y-axis. If NULL, no label is shown. |
| legend_label | (*optional*) Label to provide to the legend. If NULL, the legend will not have a name. |
| plot_title | (*optional*) Label to provide as figure title. If NULL, no title is shown. |
| plot_sub_title | (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown. |

caption (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

rotate_x_tick_labels
(*optional*) Rotate tick labels on the x-axis by 90 degrees. Defaults to TRUE. Rotation of x-axis tick labels may also be controlled through the ggtheme. In this case, FALSE should be provided explicitly.

show_alpha (*optional*) Interpreting confusion matrices is made easier by setting the opacity of the cells. show_alpha takes the following values:

- none: Cell opacity is not altered. Diagonal and off-diagonal cells are completely opaque and transparent, respectively. Same as show_alpha=FALSE.
- by_class: Cell opacity is normalised by the number of instances for each observed outcome class in each confusion matrix.
- by_matrix (default): Cell opacity is normalised by the number of instances in the largest observed outcome class in each confusion matrix. Same as show_alpha=TRUE
- by_figure: Cell opacity is normalised by the number of instances in the largest observed outcome class across confusion matrices in different facets.
- by_all: Cell opacity is normalised by the number of instances in the largest observed outcome class across all confusion matrices.

width (*optional*) Width of the plot. A default value is derived from the number of facets.

height (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection
(*optional*) Exports the collection if TRUE.

... Arguments passed on to as_familiar_collection, ggplot2::ggsave, extract_confusion_matrix

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

`data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.

`is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.

`cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.

`ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:

- `median` (default): Use the median of the predicted values as the ensemble value for a sample.
- `mean`: Use the mean of the predicted values as the ensemble value for a sample.

`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.

`detail_level` (*optional*) Sets the level at which results are computed and aggregated.

- `ensemble`: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.

hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.
A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

## Details

This function generates area under the ROC curve plots.

Available splitting variables are: vimp_method, learner and data_set. By default, the data is split by vimp_method and learner, with facetting by data_set.

Labelling methods such as set_vimp_method_names or set_data_set_names can be applied to the familiarCollection object to update labels, and order the output in the figure.

## Value

NULL or list of plot objects, if dir_path is NULL.

---

# plot_decision_curve *Plot decision curves.*

---

## Description

This method creates decision curves based on data in a familiarCollection object.

## Usage

```
plot_decision_curve(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
```

```
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_decision_curve(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

```
## S4 method for signature 'familiarCollection'
plot_decision_curve(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | `familiarCollection` object, or one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some `familiarData` objects can be created from prediction tables (`familiarDataElementPredictionTable`). Other `familiarData` objects can be created from data (`dataObject`, or `data.table`). Please check *details* for more information. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where created decision curve plots are saved to. Output is saved in the `decision_curve_analysis` subdirectory. If `NULL`, figures are written to the folder, but are returned instead. |

split_by (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

color_by (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

ggtheme (*optional*) ggplot theme to use for plotting.

discrete_palette (*optional*) Palette for colouring the plot elements according to the groupings indicated by the color_by argument (if any). familiar has a default palette. Other palettes are supported by paletteer, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

x_range (*optional*) Value range for the x-axis.

x_n_breaks (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

x_breaks (*optional*) Break points on the x-axis of the plot.

y_range (*optional*) Value range for the y-axis.

y_n_breaks (*optional*) Number of breaks to show on the y-axis of the plot. y_n_breaks is used to determine the y_breaks argument in case it is unset.

y_breaks (*optional*) Break points on the y-axis of the plot.

conf_int_style (*optional*) Confidence interval style. See details for allowed styles.

conf_int_alpha (*optional*) Alpha value to determine transparency of confidence intervals or, alternatively, other plot elements with which the confidence interval overlaps. Only values between 0.0 (fully transparent) and 1.0 (fully opaque) are allowed.

`width` (*optional*) Width of the plot. A default value is derived from the number of facets.

`height` (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

`units` (*optional*) Plot size unit. Either `cm` (default), `mm` or `in`.

`export_collection`
(*optional*) Exports the collection if TRUE.

`...` Arguments passed on to `as_familiar_collection`, `ggplot2::ggsave`, `extract_decision_curve_dat`

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name` Name of the collection.

`device` Device to use. Can either be a device function (e.g. `png`), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If `NULL` (default), the device is guessed based on the `filename` extension.

`scale` Multiplicative scaling factor.

`dpi` Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

`limitsize` When `TRUE` (the default), `ggsave()` will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

`bg` Background colour. If `NULL`, uses the `plot.background` fill value from the plot theme.

`create.dir` Whether to create new directories if a non-existing directory is specified in the `filename` or `path` (`TRUE`) or return an error (`FALSE`, default). If `FALSE` and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

`data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.

`is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.

`cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.

`evaluation_times` One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

`ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:

- `median` (default): Use the median of the predicted values as the ensemble value for a sample.
- `mean`: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.

hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.

A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").

This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.

If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.

- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

aggregate_results (*optional*) Flag that signifies whether results should be aggregated during evaluation. If estimation_type is bias_correction or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.
The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE, , "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

## Details

This function generates plots for decision curves.

Available splitting variables are: vimp_method, learner, data_set and positive_class (categorical outcomes) or evaluation_time (survival outcomes). By default, the data is split by vimp_method and learner, with faceting by data_set and colouring by positive_class or evaluation_time.

Bootstrap confidence intervals of the decision curve (if present) can be shown using various styles set by `conf_int_style`:

- `ribbon` (default): confidence intervals are shown as a ribbon with an opacity of `conf_int_alpha` around the point estimate of the decision curve.
- `step` (default): confidence intervals are shown as a step function around the point estimate of the decision curve.
- `none`: confidence intervals are not shown. The point estimate of the decision curve is shown as usual.

Labelling methods such as `set_vimp_method_names` or `set_data_set_names` can be applied to the `familiarCollection` object to update labels, and order the output in the figure.

## Value

`NULL` or list of plot objects, if `dir_path` is `NULL`.

## References

1. Vickers, A. J. & Elkin, E. B. Decision curve analysis: a novel method for evaluating prediction models. Med. Decis. Making 26, 565–574 (2006).
2. Vickers, A. J., Cronin, A. M., Elkin, E. B. & Gonen, M. Extensions to decision curve analysis, a novel method for evaluating diagnostic tests, prediction models and molecular markers. BMC Med. Inform. Decis. Mak. 8, 53 (2008).
3. Vickers, A. J., van Calster, B. & Steyerberg, E. W. A simple, step-by-step guide to interpreting decision curve analysis. Diagn Progn Res 3, 18 (2019).

---

# plot_feature_similarity

*Plot heatmaps for pairwise similarity between features.*

---

## Description

This method creates a heatmap based on data stored in a `familiarCollection` object. Features in the heatmap are ordered so that more similar features appear together.

## Usage

```
plot_feature_similarity(
  object,
  features = waiver(),
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  draw = FALSE,
  dir_path = NULL,
```

```
  split_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  gradient_palette = NULL,
  gradient_palette_range = NULL,
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  y_range = NULL,
  y_n_breaks = 3L,
  y_breaks = NULL,
  rotate_x_tick_labels = waiver(),
  remove_feature_labels = FALSE,
  show_dendrogram = c("top", "right"),
  dendrogram_height = grid::unit(1.5, "cm"),
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_feature_similarity(
  object,
  features = waiver(),
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  gradient_palette = NULL,
  gradient_palette_range = NULL,
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
```

```
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  y_range = NULL,
  y_n_breaks = 3L,
  y_breaks = NULL,
  rotate_x_tick_labels = waiver(),
  remove_feature_labels = FALSE,
  show_dendrogram = c("top", "right"),
  dendrogram_height = grid::unit(1.5, "cm"),
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_feature_similarity(
  object,
  features = waiver(),
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  gradient_palette = NULL,
  gradient_palette_range = NULL,
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  y_range = NULL,
  y_n_breaks = 3L,
  y_breaks = NULL,
  rotate_x_tick_labels = waiver(),
  remove_feature_labels = FALSE,
  show_dendrogram = c("top", "right"),
```

```
  dendrogram_height = grid::unit(1.5, "cm"),
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| features | Features that should be considered for extracting information from. Typically called in external workflows, e.g. for plotting. Internally, i.e. from summon_familiar, this variable is not used. |
| feature_cluster_method | The method used to perform clustering. These are the same methods as for the cluster_method configuration parameter: none, hclust, agnes, diana and pam.<br>none cannot be used when extracting data regarding mutual correlation or feature expressions.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| feature_linkage_method | The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| feature_cluster_cut_method | The method used to divide features into separate clusters. The available methods are the same as for the cluster_cut_method configuration parameter: silhouette, fixed_cut and dynamic_cut.<br>silhouette is available for all cluster methods, but fixed_cut only applies to methods that create hierarchical trees (hclust, agnes and diana). dynamic_cut requires the dynamicTreeCut package and can only be used with agnes and hclust.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| feature_similarity_threshold | The threshold level for pair-wise similarity that is required to form feature clusters with the fixed_cut method.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| draw | (*optional*) Draws the plot if TRUE. |

dir_path (*optional*) Path to the directory where created performance plots are saved to. Output is saved in the feature_similarity subdirectory. If NULL no figures are saved, but are returned instead.

split_by (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

ggtheme (*optional*) ggplot theme to use for plotting.

gradient_palette (*optional*) Sequential or divergent palette used to colour the similarity or distance between features in a heatmap. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

gradient_palette_range (*optional*) Numerical range used to span the gradient. This should be a range of two values, e.g. c(0, 1). Lower or upper boundary can be unset by using NA. If not set, the full metric-specific range is used.

x_label (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

x_label_shared (*optional*) Sharing of x-axis labels between facets. One of three values:

- overall: A single label is placed at the bottom of the figure. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).
- column: A label is placed at the bottom of each column. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).
- individual: A label is placed below each facet plot. Tick text is kept.

y_label (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

y_label_shared (*optional*) Sharing of y-axis labels between facets. One of three values:

- overall: A single label is placed to the left of the figure. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).
- row: A label is placed to the left of each row. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).
- individual: A label is placed below each facet plot. Tick text is kept.

legend_label (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

y_range (*optional*) Value range for the y-axis.

y_n_breaks (*optional*) Number of breaks to show on the y-axis of the plot. y_n_breaks is used to determine the y_breaks argument in case it is unset.

y_breaks (*optional*) Break points on the y-axis of the plot.

rotate_x_tick_labels
(*optional*) Rotate tick labels on the x-axis by 90 degrees. Defaults to TRUE. Rotation of x-axis tick labels may also be controlled through the ggtheme. In this case, FALSE should be provided explicitly.

remove_feature_labels
(*optional*) If TRUE, feature labels are not shown. By default, feature labels are shown.

show_dendrogram
(*optional*) Show dendrogram around the main panel. Can be TRUE, FALSE, NULL, or a position, i.e. top, bottom, left and right. Up to two positions may be provided, but only as long as the dendrograms are not on opposite sides of the heatmap: top and bottom, and left and right cannot be used together.

A dendrogram can only be drawn from cluster methods that produce dendrograms, such as hclust. A dendrogram can for example not be constructed using the partitioning around medioids method (pam).

By default, a dendrogram is drawn to the top and right of the panel.

dendrogram_height
(*optional*) Height of the dendrogram. The height is 1.5 cm by default. Height is expected to be grid unit (see grid::unit), which also allows for specifying relative heights.

width (*optional*) Width of the plot. A default value is derived from the number of facets.

height (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection
(*optional*) Exports the collection if TRUE.

... Arguments passed on to as_familiar_collection, ggplot2::ggsave, extract_feature_similarity

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

feature_similarity_metric Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and feature_similarity_metric therefore has the same options as cluster_similarity_metric: mcfadden_r2, cox_snell_r2, nagelkerke_r2, spearman, kendall and pearson.

If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

aggregate_results (*optional*) Flag that signifies whether results should be aggregated during evaluation. If estimation_type is bias_correction

or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.
The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE, , "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

## Details

This function generates area under the ROC curve plots.

Available splitting variables are: vimp_method, learner, and data_set. By default, the data is split by vimp_method, learner and data_set, since the features may be ordered differently for each data set.

Note that similarity is determined based on the underlying data. Hence the ordering of features may differ between facets, and tick labels are maintained for each panel.

Labeling methods such as set_vimp_method_names or set_data_set_names can be applied to the familiarCollection object to update labels, and order the output in the figure.

This plot can be created from dataObject, or data.table objects. For data.table, see as_data_object for additional arguments.

## Value

NULL or list of plot objects, if dir_path is NULL.

plot_ice *Plot individual conditional expectation plots.*

## Description

This method creates individual conditional expectation plots based on data in a familiarCollection object.

## Usage

```
plot_ice(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  gradient_palette_range = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  novelty_range = NULL,
  value_scales = waiver(),
  novelty_scales = waiver(),
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  ice_default_alpha = 0.6,
  n_max_samples_shown = 50L,
  show_ice = TRUE,
  show_pd = TRUE,
  show_novelty = TRUE,
  anchor_values = NULL,
  width = waiver(),
  height = waiver(),
```

```
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_ice(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  gradient_palette_range = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  novelty_range = NULL,
  value_scales = waiver(),
  novelty_scales = waiver(),
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  ice_default_alpha = 0.6,
  n_max_samples_shown = 50L,
  show_ice = TRUE,
  show_pd = TRUE,
  show_novelty = TRUE,
  anchor_values = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

```
## S4 method for signature 'familiarCollection'
plot_ice(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  gradient_palette_range = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  novelty_range = NULL,
  value_scales = waiver(),
  novelty_scales = waiver(),
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  ice_default_alpha = 0.6,
  n_max_samples_shown = 50L,
  show_ice = TRUE,
  show_pd = TRUE,
  show_novelty = TRUE,
  anchor_values = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | `familiarCollection` object, or one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to such files |

can also be provided.
Additionally, some familiarData objects can be created from prediction tables (familiarDataElementPredictionTable). Other familiarData objects can be created from data (dataObject, or data.table). Please check *details* for more information.

draw (*optional*) Draws the plot if TRUE.

dir_path (*optional*) Path to the directory where created individual conditional expectation plots are saved to. Output is saved in the explanation subdirectory. If NULL, figures are written to the folder, but are returned instead.

split_by (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

color_by (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols
(*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

ggtheme (*optional*) ggplot theme to use for plotting.

discrete_palette
(*optional*) Palette for colouring plot elements indicated by the color_by argument (if any). For 2D individual conditional expectation plots without novelty, the initial colour determines the colour of the points indicating sample values. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

gradient_palette
(*optional*) Sequential or divergent palette used to colour the raster in 2D individual conditional expectation or partial dependence plots. This argument is not used for 1D plots. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

gradient_palette_range
(*optional*) Numerical range used to span the gradient for 2D plots. This should be a range of two values, e.g. c(0, 1). By default, values are determined from

the data, dependent on the value_scales parameter. This parameter is ignored for 1D plots.

x_label (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

x_range (*optional*) Value range for the x-axis.

x_n_breaks (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

x_breaks (*optional*) Break points on the x-axis of the plot.

y_range (*optional*) Value range for the y-axis.

y_n_breaks (*optional*) Number of breaks to show on the y-axis of the plot. y_n_breaks is used to determine the y_breaks argument in case it is unset.

y_breaks (*optional*) Break points on the y-axis of the plot.

novelty_range (*optional*) Numerical range used to span the range of novelty values. This determines the size of the bubbles in 2D, and transparency of lines in 1D. This should be a range of two values, e.g. c(0, 1). By default, values are determined from the data, dependent on the value_scales parameter. This parameter is ignored if show_novelty=FALSE.

value_scales (*optional*) Sets scaling of predicted values. This parameter has several options:

- fixed (default): The value axis for all features will have the same range.
- feature: The value axis for each feature will have the same range. This option is unavailable for 2D plots.
- figure: The value axis for all facets in a figure will have the same range.
- facet: Each facet has its own range. This option is unavailable for 2D plots.

For 1D plots, this option is ignored if the y_range is provided, whereas for 2D it is ignored if the gradient_palette_range is provided.

novelty_scales (*optional*) Sets scaling of novelty values, similar to the value_scales parameter, but with more limited options:

- fixed (default): The novelty will have the same range for all features.
- figure: The novelty will have the same range for all facets in a figure.

conf_int_style (*optional*) Confidence interval style. See details for allowed styles.

conf_int_alpha (*optional*) Alpha value to determine transparency of confidence intervals or, alternatively, other plot elements with which the confidence interval overlaps. Only values between 0.0 (fully transparent) and 1.0 (fully opaque) are allowed.

ice_default_alpha

(*optional*) Default transparency (value) of sample lines in an 1D plot. When novelty is shown, this is the transparency corresponding to the least novel points. The confidence interval alpha values is scaled by this value.

`n_max_samples_shown`
: (*optional*) Maximum number of samples shown in an individual conditional expectation plot. Defaults to 50. These samples are randomly picked from the samples present in the ICE data, but the same samples are consistently picked. Partial dependence is nonetheless computed from all available samples.

`show_ice`
: (*optional*) Sets whether individual conditional expectation plots should be created.

`show_pd`
: (*optional*) Sets whether partial dependence plots should be created. Note that if an anchor is set for a particular feature, its partial dependence cannot be shown.

`show_novelty`
: (*optional*) Sets whether novelty is shown in plots.

`anchor_values`
: (*optional*) A single value or a named list or array of values that are used to centre the individual conditional expectation plot. A single value is valid if and only if only a single feature is assessed. Otherwise, values Has no effect if the plot is not shown, i.e. `show_ice=FALSE`. A partial dependence plot cannot be shown for those features.

`width`
: (*optional*) Width of the plot. A default value is derived from the number of facets.

`height`
: (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

`units`
: (*optional*) Plot size unit. Either `cm` (default), `mm` or `in`.

`export_collection`
: (*optional*) Exports the collection if TRUE.

`...`
: Arguments passed on to `export_ice_data`, `ggplot2::ggsave`, `extract_ice`

  `aggregate_results` Flag that signifies whether results should be aggregated for export.

  `device` Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If `NULL` (default), the device is guessed based on the `filename` extension.

  `scale` Multiplicative scaling factor.

  `dpi` Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

  `limitsize` When `TRUE` (the default), `ggsave()` will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

  `bg` Background colour. If `NULL`, uses the `plot.background` fill value from the plot theme.

  `create.dir` Whether to create new directories if a non-existing directory is specified in the `filename` or `path` (`TRUE`) or return an error (`FALSE`, default). If `FALSE` and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

  `features` Names of the feature or features (2) assessed simultaneously. By default `NULL`, which means that all features are assessed one-by-one.

feature_x_range When one or two features are defined using features, feature_x_range can be used to set the range of values for the first feature. For numeric features, a vector of two values is assumed to indicate a range from which n_sample_points are uniformly sampled. A vector of more than two values is interpreted as is, i.e. these represent the values to be sampled. For categorical features, values should represent a (sub)set of available levels.

feature_y_range As feature_x_range, but for the second feature in case two features are defined.

n_sample_points Number of points used to sample continuous features.

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

sample_limit (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("sample_similarity"=100, "permutation_vimp"=1000).
This parameter can be set for the following data elements: sample_similarity, shap, permutation_vimp, and ice_data.

n_important_features (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("ice_data"=10, "permutation_vimp"=5).
This parameter can be set for the following data elements: ice_data, permutation_vimp, and shap.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.
A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016).

The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`. This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

## Details

This function generates individual conditional expectation plots. These plots come in two varieties, namely 1D and 2D. 1D plots show the predicted value as function of a single feature, whereas 2D plots show the predicted value as a function of two features.

Available splitting variables are: `feature_x`, `feature_y` (2D only), `vimp_method`, `learner`, `data_set` and `positive_class` (categorical outcomes) or `evaluation_time` (survival outcomes). By default, for 1D ICE plots the data are split by `feature_x`, `vimp_method` and `learner`, with faceting by `data_set`, `positive_class` or `evaluation_time`. If only partial dependence is shown, `positive_class` and `evaluation_time` are used to set colours instead. For 2D plots, by default the data are split by `feature_x`, `vimp_method` and `learner`, with faceting by `data_set`, `positive_class` or `evaluation_time`. The `color_by` argument cannot be used with 2D plots, and attempting to do so causes an error. Attempting to specify `feature_x` or `feature_y` for `color_by` will likewise result in an error, as multiple features cannot be shown in the same facet.

Bootstrap confidence intervals of the partial dependence plots can be shown using various styles set by `conf_int_style`:

- `ribbon` (default): confidence intervals are shown as a ribbon with an opacity of `conf_int_alpha` around the point estimate of the partial dependence.
- `step` (default): confidence intervals are shown as a step function around the point estimate of the partial dependence.
- `none`: confidence intervals are not shown. The point estimate of the partial dependence is shown as usual.

Note that when bootstrap confidence intervals were computed, they were also computed for individual samples in individual conditional expectation plots. To avoid clutter, only point estimates for individual samples are shown.

Labelling methods such as `set_vimp_method_names` or `set_data_set_names` can be applied to the `familiarCollection` object to update labels, and order the output in the figure.

## Value

`NULL` or list of plot objects, if `dir_path` is `NULL`.

---

`plot_kaplan_meier` *Plot Kaplan-Meier survival curves.*

---

## Description

This function creates Kaplan-Meier survival curves from stratification data stored in a familiarCollection object.

## Usage

```
plot_kaplan_meier(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  linetype_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  combine_legend = TRUE,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = "time",
  x_label_shared = "column",
  y_label = "survival probability",
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = c(0, 1),
  y_n_breaks = 5L,
  y_breaks = NULL,
  confidence_level = NULL,
```

```
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  censoring = TRUE,
  censor_shape = "plus",
  show_logrank = TRUE,
  show_survival_table = TRUE,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_kaplan_meier(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  linetype_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  combine_legend = TRUE,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = "time",
  x_label_shared = "column",
  y_label = "survival probability",
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = c(0, 1),
  y_n_breaks = 5L,
  y_breaks = NULL,
  confidence_level = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  censoring = TRUE,
  censor_shape = "plus",
  show_logrank = TRUE,
  show_survival_table = TRUE,
  width = waiver(),
```

```
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_kaplan_meier(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  linetype_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  combine_legend = TRUE,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = "time",
  x_label_shared = "column",
  y_label = "survival probability",
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = c(0, 1),
  y_n_breaks = 5L,
  y_breaks = NULL,
  confidence_level = NULL,
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  censoring = TRUE,
  censor_shape = "plus",
  show_logrank = TRUE,
  show_survival_table = TRUE,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | familiarCollection object, or one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some familiarData objects can be created from prediction tables (familiarDataElementPredictionTable). Other familiarData objects can be created from data (dataObject, or data.table). Please check *details* for more information. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where created figures are saved to. Output is saved in the stratification subdirectory. If NULL no figures are saved, but are returned instead. |
| split_by | (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables. |
| color_by | (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables. |
| linetype_by | (*optional*) Variables that are used to determine the linetype of lines in a plot. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. Sett details for available variables. |
| facet_by | (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables. |
| facet_wrap_cols | (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead. |
| combine_legend | (*optional*) Flag to indicate whether the same legend is to be shared by multiple aesthetics, such as those specified by color_by and linetype_by arguments. |
| ggtheme | (*optional*) ggplot theme to use for plotting. |
| discrete_palette | (*optional*) Palette for colouring the plot elements according to the groupings indicated by the color_by argument (if any). familiar has a default palette. Other palettes are supported by paletteer, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings. |
| x_label | (*optional*) Label to provide to the x-axis. If NULL, no label is shown. |

`x_label_shared` (*optional*) Sharing of x-axis labels between facets. One of three values:

- `overall`: A single label is placed at the bottom of the figure. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).
- `column`: A label is placed at the bottom of each column. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).
- `individual`: A label is placed below each facet plot. Tick text is kept.

`y_label` (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

`y_label_shared` (*optional*) Sharing of y-axis labels between facets. One of three values:

- `overall`: A single label is placed to the left of the figure. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).
- `row`: A label is placed to the left of each row. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).
- `individual`: A label is placed below each facet plot. Tick text is kept.

`legend_label` (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

`plot_title` (*optional*) Label to provide as figure title. If NULL, no title is shown.

`plot_sub_title` (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

`caption` (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

`x_range` (*optional*) Value range for the x-axis.

`x_n_breaks` (*optional*) Number of breaks to show on the x-axis of the plot. `x_n_breaks` is used to determine the `x_breaks` argument in case it is unset.

`x_breaks` (*optional*) Break points on the x-axis of the plot.

`y_range` (*optional*) Value range for the y-axis.

`y_n_breaks` (*optional*) Number of breaks to show on the y-axis of the plot. `y_n_breaks` is used to determine the `y_breaks` argument in case it is unset.

`y_breaks` (*optional*) Break points on the y-axis of the plot.

`confidence_level` (*optional*) Confidence level for the strata in the plot.

`conf_int_style` (*optional*) Confidence interval style. See details for allowed styles.

`conf_int_alpha` (*optional*) Alpha value to determine transparency of confidence intervals or, alternatively, other plot elements with which the confidence interval overlaps. Only values between 0.0 (fully transparent) and 1.0 (fully opaque) are allowed.

`censoring` (*optional*) Flag to indicate whether censored samples should be indicated on the survival curve.

`censor_shape` (*optional*) Shape used to indicate censored samples on the survival curve. Available shapes are documented in the `ggplot2` vignette *Aesthetic specifications*. By default a plus shape is used.

`show_logrank` (*optional*) Specifies whether the results of a logrank test to assess differences between the risk strata is annotated in the plot. A log-rank test can only be shown when `color_by` and `linestyle_by` are either unset, or only contain `group`.

`show_survival_table`
: (*optional*) Specifies whether a survival table is shown below the Kaplan-Meier survival curves. Survival in the risk strata is assessed for each of the breaks in `x_breaks`.

`width`
: (*optional*) Width of the plot. A default value is derived from the number of facets.

`height`
: (*optional*) Height of the plot. A default value is derived from number of facets and the inclusion of survival tables.

`units`
: (*optional*) Plot size unit. Either `cm` (default), `mm` or `in`.

`export_collection`
: (*optional*) Exports the collection if TRUE.

`...`
: Arguments passed on to `as_familiar_collection`, `ggplot2::ggsave`, `extract_risk_stratificatio`

  `familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

  `collection_name` Name of the collection.

  `device` Device to use. Can either be a device function (e.g. `png`), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If `NULL` (default), the device is guessed based on the `filename` extension.

  `scale` Multiplicative scaling factor.

  `dpi` Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

  `limitsize` When `TRUE` (the default), `ggsave()` will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

  `bg` Background colour. If `NULL`, uses the `plot.background` fill value from the plot theme.

  `create.dir` Whether to create new directories if a non-existing directory is specified in the `filename` or `path` (`TRUE`) or return an error (`FALSE`, default). If `FALSE` and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

  `data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.

  `is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.

  `cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.

  `ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:

  - `median` (default): Use the median of the predicted values as the ensemble value for a sample.
  - `mean`: Use the mean of the predicted values as the ensemble value for a sample.

`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.

`detail_level` (*optional*) Sets the level at which results are computed and aggregated.

- `ensemble`: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.
A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.
If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

### Details

This function generates a Kaplan-Meier survival plot based on risk group stratification by the learners.

`familiar` does not determine what units the x-axis has or what kind of survival the y-axis represents. It is therefore recommended to provide `x_label` and `y_label` arguments.

Available splitting variables are: `vimp_method`, `learner`, `data_set`, `group` and `stratification_method`. By default, separate figures are created for each combination of `vimp_method` and `learner`, with faceting by `data_set`, colouring of the strata in each individual plot by `group`.

Greenwood confidence intervals of the Kaplan-Meier curve can be shown using various styles set by `conf_int_style`:

- `ribbon` (default): confidence intervals are shown as a ribbon with an opacity of `conf_int_alpha` around the point estimate of the Kaplan-Meier curve.
- `step` (default): confidence intervals are shown as a step function around the point estimate of the Kaplan-Meier curve.
- `none`: confidence intervals are not shown. The point estimate of the ROC curve is shown as usual.

Labelling methods such as `set_risk_group_names` or `set_data_set_names` can be applied to the `familiarCollection` object to update labels, and order the output in the figure.

### Value

`NULL` or list of plot objects, if `dir_path` is `NULL`.

---

## plot_model_performance

*Plot model performance.*

---

### Description

This method creates plots that show model performance from the data stored in a familiarCollection object. This method may create several types of plots, as determined by `plot_type`.

### Usage

```
plot_model_performance(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
```

```
  plot_type = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  gradient_palette_range = waiver(),
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  rotate_x_tick_labels = waiver(),
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  annotate_performance = NULL,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_model_performance(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  plot_type = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  gradient_palette_range = waiver(),
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  rotate_x_tick_labels = waiver(),
  y_range = NULL,
  y_n_breaks = 5L,
```

```
  y_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  annotate_performance = NULL,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_model_performance(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  plot_type = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  gradient_palette_range = waiver(),
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  rotate_x_tick_labels = waiver(),
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  annotate_performance = NULL,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | familiarCollection object, or one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to such files |

can also be provided.

Additionally, some familiarData objects can be created from prediction tables (familiarDataElementPredictionTable). Other familiarData objects can be created from data (dataObject, or data.table). Please check *details* for more information.

draw (*optional*) Draws the plot if TRUE.

dir_path (*optional*) Path to the directory where created performance plots are saved to. Output is saved in the performance subdirectory. If NULL no figures are saved, but are returned instead.

split_by (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

x_axis_by (*optional*) Variable plotted along the x-axis of a plot. The variable cannot overlap with variables provided to the split_by and y_axis_by arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables.

y_axis_by (*optional*) Variable plotted along the y-axis of a plot. The variable cannot overlap with variables provided to the split_by and x_axis_by arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables.

color_by (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

plot_type (*optional*) Type of plot to draw. This is one of heatmap (draws a heatmap), barplot (draws a barplot with confidence intervals), boxplot (draws a boxplot) and violinplot (draws a violin plot). Defaults to violinplot.

The choice for plot_type affects several other arguments, e.g. color_by is not used for heatmap and y_axis_by is only used by heatmap.

ggtheme (*optional*) ggplot theme to use for plotting.

discrete_palette (*optional*) Palette for colouring plot elements indicated by the color_by argument (if any). Only used if plot_type is not heatmap. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

gradient_palette
: (*optional*) Sequential or divergent palette used to color the raster in `heatmap` plots. This argument is not used for other `plot_type` value. `familiar` has a default palette. Other palettes are supported by the `paletteer` package, `grDevices::palette.pals()` (requires R >= 4.0.0), `grDevices::hcl.pals()` (requires R >= 3.6.0) and `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` and `cm.colors`, which correspond to the palettes of the same name in `grDevices`. You may also specify your own palette by providing a vector of colour names listed by `grDevices::colors()` or through hexadecimal RGB strings.

gradient_palette_range
: (*optional*) Numerical range used to span the gradient. This should be a range of two values, e.g. `c(0, 1)`. Lower or upper boundary can be unset by using `NA`. If not set, the full metric-specific range is used.

x_label
: (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label
: (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label
: (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title
: (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title
: (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption
: (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

rotate_x_tick_labels
: (*optional*) Rotate tick labels on the x-axis by 90 degrees. Defaults to `TRUE`. Rotation of x-axis tick labels may also be controlled through the `ggtheme`. In this case, `FALSE` should be provided explicitly.

y_range
: (*optional*) Value range for the y-axis.

y_n_breaks
: (*optional*) Number of breaks to show on the y-axis of the plot. `y_n_breaks` is used to determine the `y_breaks` argument in case it is unset.

y_breaks
: (*optional*) Break points on the y-axis of the plot.

width
: (*optional*) Width of the plot. A default value is derived from the number of facets.

height
: (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units
: (*optional*) Plot size unit. Either `cm` (default), `mm` or `in`.

annotate_performance
: (*optional*) Indicates whether performance in heatmaps should be annotated with text. Can be `none`, `value` (default), or `value_ci` (median value plus 95% credibility intervals).

export_collection
: (*optional*) Exports the collection if TRUE.

...
: Arguments passed on to `extract_performance`, `as_familiar_collection`, `ggplot2::ggsave`

  `data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.

`is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.

`cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.

`evaluation_times` One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

`ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:

- `median` (default): Use the median of the predicted values as the ensemble value for a sample.
- `mean`: Use the mean of the predicted values as the ensemble value for a sample.

`metric` One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`verbose` Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.

`detail_level` (*optional*) Sets the level at which results are computed and aggregated.

- `ensemble`: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a

respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.
A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

aggregate_results (*optional*) Flag that signifies whether results should be aggregated during evaluation. If estimation_type is bias_correction or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.
The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE, , "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the

confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name` Name of the collection.

`device` Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If `NULL` (default), the device is guessed based on the `filename` extension.

`scale` Multiplicative scaling factor.

`dpi` Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

`limitsize` When `TRUE` (the default), `ggsave()` will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

`bg` Background colour. If `NULL`, uses the `plot.background` fill value from the plot theme.

`create.dir` Whether to create new directories if a non-existing directory is specified in the `filename` or `path` (`TRUE`) or return an error (`FALSE`, default). If `FALSE` and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

## Details

This function plots model performance based on empirical bootstraps, using various plot representations.

Available splitting variables are: `vimp_method`, `learner`, `data_set`, `evaluation_time` (survival outcome only) and `metric`. The default for `heatmap` is to split by `metric`, facet by `data_set` and `evaluation_time`, position `learner` along the x-axis and `vimp_method` along the y-axis. The `color_by` argument is not used. The only valid options for `x_axis_by` and `y_axis_by` are `learner` and `vimp_method`.

For other plot types (`barplot`, `boxplot` and `violinplot`), depends on the number of learners and feature selection methods:

- *one feature selection method and one learner*: the default is to split by `metric`, and have `data_set` along the x-axis.

- *one feature selection and multiple learners*: the default is to split by `metric`, facet by `data_set` and have `learner` along the x-axis.
- *multiple feature selection methods and one learner*: the default is to split by `metric`, facet by `data_set` and have `vimp_method` along the x-axis.
- *multiple feature selection methods and learners*: the default is to split by `metric`, facet by `data_set`, colour by `vimp_method` and have `learner` along the x-axis.

If applicable, additional faceting is performed for `evaluation_time`.

Labeling methods such as `set_vimp_method_names` or `set_data_set_names` can be applied to the `familiarCollection` object to update labels, and order the output in the figure.

## Value

`NULL` or list of plot objects, if `dir_path` is `NULL`.

---

`plot_pd` *Plot partial dependence.*

---

## Description

This method creates partial dependence plots based on data in a familiarCollection object.

## Usage

```
plot_pd(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  gradient_palette_range = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
```

```
  y_breaks = NULL,
  novelty_range = NULL,
  value_scales = waiver(),
  novelty_scales = waiver(),
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  show_novelty = TRUE,
  anchor_values = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_pd(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  gradient_palette_range = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  novelty_range = NULL,
  value_scales = waiver(),
  novelty_scales = waiver(),
  conf_int_style = c("ribbon", "step", "none"),
  conf_int_alpha = 0.2,
  show_novelty = TRUE,
  anchor_values = NULL,
  width = waiver(),
```

```
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | `familiarCollection` object, or one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some `familiarData` objects can be created from prediction tables (`familiarDataElementPredictionTable`). Other `familiarData` objects can be created from data (`dataObject`, or `data.table`). Please check *details* for more information. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where created individual conditional expectation plots are saved to. Output is saved in the `explanation` subdirectory. If `NULL`, figures are written to the folder, but are returned instead. |
| split_by | (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables. |
| color_by | (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the `split_by` argument, but may overlap with other arguments. See details for available variables. |
| facet_by | (*optional*) Variables used to determine how and if facets of each figure appear. In case the `facet_wrap_cols` argument is `NULL`, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the `split_by` argument, but may overlap with other arguments. See details for available variables. |
| facet_wrap_cols | (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead. |
| ggtheme | (*optional*) `ggplot` theme to use for plotting. |
| discrete_palette | (*optional*) Palette for colouring plot elements indicated by the `color_by` argument (if any). For 2D individual conditional expectation plots without novelty, the initial colour determines the colour of the points indicating sample values. `familiar` has a default palette. Other palettes are supported by the `paletteer` package, `grDevices::palette.pals()` (requires R >= 4.0.0), `grDevices::hcl.pals()` (requires R >= 3.6.0) and `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` and `cm.colors`, which correspond to the palettes of the same name in `grDevices`. You may also specify your own palette by providing a vector of colour names listed by `grDevices::colors()` or through hexadecimal RGB strings. |

gradient_palette
: (*optional*) Sequential or divergent palette used to colour the raster in 2D individual conditional expectation or partial dependence plots. This argument is not used for 1D plots. `familiar` has a default palette. Other palettes are supported by the `paletteer` package, `grDevices::palette.pals()` (requires R >= 4.0.0), `grDevices::hcl.pals()` (requires R >= 3.6.0) and `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` and `cm.colors`, which correspond to the palettes of the same name in `grDevices`. You may also specify your own palette by providing a vector of colour names listed by `grDevices::colors()` or through hexadecimal RGB strings.

gradient_palette_range
: (*optional*) Numerical range used to span the gradient for 2D plots. This should be a range of two values, e.g. `c(0, 1)`. By default, values are determined from the data, dependent on the `value_scales` parameter. This parameter is ignored for 1D plots.

x_label
: (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label
: (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label
: (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title
: (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title
: (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption
: (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

x_range
: (*optional*) Value range for the x-axis.

x_n_breaks
: (*optional*) Number of breaks to show on the x-axis of the plot. `x_n_breaks` is used to determine the `x_breaks` argument in case it is unset.

x_breaks
: (*optional*) Break points on the x-axis of the plot.

y_range
: (*optional*) Value range for the y-axis.

y_n_breaks
: (*optional*) Number of breaks to show on the y-axis of the plot. `y_n_breaks` is used to determine the `y_breaks` argument in case it is unset.

y_breaks
: (*optional*) Break points on the y-axis of the plot.

novelty_range
: (*optional*) Numerical range used to span the range of novelty values. This determines the size of the bubbles in 2D, and transparency of lines in 1D. This should be a range of two values, e.g. `c(0, 1)`. By default, values are determined from the data, dependent on the `value_scales` parameter. This parameter is ignored if `show_novelty=FALSE`.

value_scales
: (*optional*) Sets scaling of predicted values. This parameter has several options:

  - `fixed` (default): The value axis for all features will have the same range.
  - `feature`: The value axis for each feature will have the same range. This option is unavailable for 2D plots.
  - `figure`: The value axis for all facets in a figure will have the same range.
  - `facet`: Each facet has its own range. This option is unavailable for 2D plots.

  For 1D plots, this option is ignored if the `y_range` is provided, whereas for 2D it is ignored if the `gradient_palette_range` is provided.

`novelty_scales` (*optional*) Sets scaling of novelty values, similar to the `value_scales` parameter, but with more limited options:

- `fixed` (default): The novelty will have the same range for all features.
- `figure`: The novelty will have the same range for all facets in a figure.

`conf_int_style` (*optional*) Confidence interval style. See details for allowed styles.

`conf_int_alpha` (*optional*) Alpha value to determine transparency of confidence intervals or, alternatively, other plot elements with which the confidence interval overlaps. Only values between 0.0 (fully transparent) and 1.0 (fully opaque) are allowed.

`show_novelty` (*optional*) Sets whether novelty is shown in plots.

`anchor_values` (*optional*) A single value or a named list or array of values that are used to centre the individual conditional expectation plot. A single value is valid if and only if only a single feature is assessed. Otherwise, values Has no effect if the plot is not shown, i.e. `show_ice=FALSE`. A partial dependence plot cannot be shown for those features.

`width` (*optional*) Width of the plot. A default value is derived from the number of facets.

`height` (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

`units` (*optional*) Plot size unit. Either `cm` (default), `mm` or `in`.

`export_collection` (*optional*) Exports the collection if TRUE.

`...` Arguments passed on to `export_ice_data`, `ggplot2::ggsave`, `extract_ice`

`aggregate_results` Flag that signifies whether results should be aggregated for export.

`device` Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If `NULL` (default), the device is guessed based on the `filename` extension.

`scale` Multiplicative scaling factor.

`dpi` Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

`limitsize` When `TRUE` (the default), `ggsave()` will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

`bg` Background colour. If `NULL`, uses the `plot.background` fill value from the plot theme.

`create.dir` Whether to create new directories if a non-existing directory is specified in the `filename` or `path` (`TRUE`) or return an error (`FALSE`, default). If `FALSE` and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

`features` Names of the feature or features (2) assessed simultaneously. By default `NULL`, which means that all features are assessed one-by-one.

feature_x_range When one or two features are defined using features, feature_x_range can be used to set the range of values for the first feature. For numeric features, a vector of two values is assumed to indicate a range from which n_sample_points are uniformly sampled. A vector of more than two values is interpreted as is, i.e. these represent the values to be sampled. For categorical features, values should represent a (sub)set of available levels.

feature_y_range As feature_x_range, but for the second feature in case two features are defined.

n_sample_points Number of points used to sample continuous features.

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

sample_limit (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("sample_similarity"=100, "permutation_vimp"=1000).
This parameter can be set for the following data elements: sample_similarity, shap, permutation_vimp, and ice_data.

n_important_features (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("ice_data"=10, "permutation_vimp"=5).
This parameter can be set for the following data elements: ice_data, permutation_vimp, and shap.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.
A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016).

The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`. This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

## Details

This function generates partial dependence plots. These plots come in two varieties, namely 1D and 2D. 1D plots show the predicted value as function of a single feature, whereas 2D plots show the predicted value as a function of two features.

Available splitting variables are: `feature_x`, `feature_y` (2D only), `vimp_method`, `learner`, `data_set` and `positive_class` (categorical outcomes) or `evaluation_time` (survival outcomes). By default, for 1D ICE plots the data are split by `feature_x`, `vimp_method` and `learner`, with faceting by `data_set`, `positive_class` or `evaluation_time`. If only partial dependence is shown, `positive_class` and `evaluation_time` are used to set colours instead. For 2D plots, by default the data are split by `feature_x`, `vimp_method` and `learner`, with faceting by `data_set`, `positive_class` or `evaluation_time`. The `color_by` argument cannot be used with 2D plots, and attempting to do so causes an error. Attempting to specify `feature_x` or `feature_y` for `color_by` will likewise result in an error, as multiple features cannot be shown in the same facet.

The splitting variables indicated by `color_by` are coloured according to the `discrete_palette` parameter. This parameter is therefore only used for 1D plots. Available palettes for `discrete_palette` and `gradient_palette` are those listed by `grDevices::palette.pals()` (requires R >= 4.0.0), `grDevices::hcl.pals()` (requires R >= 3.6.0) and `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` and `cm.colors`, which correspond to the palettes of the same name in `grDevices`. If not specified, a default palette based on palettes in Tableau are used. You may also specify your own palette by using colour names listed by `grDevices::colors()` or through hexadecimal RGB strings.

Bootstrap confidence intervals of the partial dependence plots can be shown using various styles set by `conf_int_style`:

- ribbon (default): confidence intervals are shown as a ribbon with an opacity of conf_int_alpha around the point estimate of the partial dependence.
- step (default): confidence intervals are shown as a step function around the point estimate of the partial dependence.
- none: confidence intervals are not shown. The point estimate of the partial dependence is shown as usual.

Labelling methods such as set_vimp_method_names or set_data_set_names can be applied to the familiarCollection object to update labels, and order the output in the figure.

## Value

NULL or list of plot objects, if dir_path is NULL.

---

plot_permutation_variable_importance
*Plot permutation variable importance.*

---

## Description

This function plots the data on permutation variable importance stored in a familiarCollection object.

## Usage

```
plot_permutation_variable_importance(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = "feature",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  conf_int_style = c("point_line", "line", "bar_line", "none"),
  conf_int_alpha = 0.2,
```

```
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_permutation_variable_importance(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  y_label = "feature",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  conf_int_style = c("point_line", "line", "bar_line", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_permutation_variable_importance(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
```

```
  x_label = waiver(),
  y_label = "feature",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  conf_int_style = c("point_line", "line", "bar_line", "none"),
  conf_int_alpha = 0.2,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | familiarCollection object, or one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some familiarData objects can be created from prediction tables (familiarDataElementPredictionTable). Other familiarData objects can be created from data (dataObject, or data.table). Please check *details* for more information. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where created figures are saved to. Output is saved in the variable_importance subdirectory. If NULL no figures are saved, but are returned instead. |
| split_by | (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables. |
| color_by | (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables. |
| facet_by | (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables. |
| facet_wrap_cols | (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead. |

ggtheme (*optional*) ggplot theme to use for plotting.

discrete_palette
(*optional*) Palette for colouring the plot elements according to the groupings indicated by the color_by argument (if any). familiar has a default palette. Other palettes are supported by paletteer, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

limit_n_features
(*optional*) The number of features that should be included in the plot. Only the most important features are shown. By default, the number of features is not limited.

x_range (*optional*) Value range for the x-axis.

x_n_breaks (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

x_breaks (*optional*) Break points on the x-axis of the plot.

conf_int_style (*optional*) Confidence interval style. See details for allowed styles.

conf_int_alpha (*optional*) Alpha value to determine transparency of confidence intervals or, alternatively, other plot elements with which the confidence interval overlaps. Only values between 0.0 (fully transparent) and 1.0 (fully opaque) are allowed.

width (*optional*) Width of the plot. A default value is derived from the number of facets.

height (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection
(*optional*) Exports the collection if TRUE.

... Arguments passed on to as_familiar_collection, ggplot2::ggsave, extract_permutation_vimp

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

metric One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_cluster_method The method used to perform clustering. These are the same methods as for the cluster_method configuration parameter: none, hclust, agnes, diana and pam.

none cannot be used when extracting data regarding mutual correlation or feature expressions.

If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_linkage_method The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.

If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_cluster_cut_method The method used to divide features into separate clusters. The available methods are the same as for the cluster_cut_method configuration parameter: silhouette, fixed_cut and dynamic_cut. silhouette is available for all cluster methods, but fixed_cut only applies to methods that create hierarchical trees (hclust, agnes and diana). dynamic_cut requires the dynamicTreeCut package and can only be used with agnes and hclust.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_threshold The threshold level for pair-wise similarity that is required to form feature clusters with the fixed_cut method.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_metric Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and feature_similarity_metric therefore has the same options as cluster_similarity_metric: mcfadden_r2, cox_snell_r2, nagelkerke_r2, spearman, kendall and pearson.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

sample_limit (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("sample_similarity"=100, "permutation_vimp"=1000).
This parameter can be set for the following data elements: sample_similarity, shap, permutation_vimp, and ice_data.

n_important_features (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.
This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("ice_data"=10, "permutation_vimp"=5).
This parameter can be set for the following data elements: ice_data, permutation_vimp, and shap.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the en-

semble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.

- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.
A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.
If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`aggregate_results` (*optional*) Flag that signifies whether results should be aggregated during evaluation. If `estimation_type` is `bias_correction`

or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.
The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE, , "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

## Details

This function generates a horizontal barplot that lists features by the estimated model improvement over that of a dataset where the respective feature is randomly permuted.

The following splitting variables are available for split_by, color_by and facet_by:

- vimp_method: variable importance methods.
- learner: learners.
- data_set: data sets.
- metric: the model performance metrics.
- evaluation_time: the evaluation times (survival outcomes only).
- similarity_threshold: the similarity threshold used to identify groups of features to permute simultaneously.

By default, the data is split by vimp_method, learner and metric, faceted by data_set and evaluation_time, and coloured by similarity_threshold.

Labelling methods such as set_vimp_method_names or set_feature_names can be applied to the familiarCollection object to update labels, and order the output in the figure.

Bootstrap confidence intervals (if present) can be shown using various styles set by conf_int_style:

- point_line (default): confidence intervals are shown as lines, on which the point estimate is likewise shown.
- line (default): confidence intervals are shown as lines, but the point estimate is not shown.
- bar_line: confidence intervals are shown as lines, with the point estimate shown as a bar plot with the opacity of conf_int_alpha.
- none: confidence intervals are not shown. The point estimate is shown as a bar plot.

For metrics where lower values indicate better model performance, more negative permutation variable importance values indicate features that are more important. Because this may cause confusion, values obtained for these metrics are mirrored around 0.0 for plotting (but not any tabular data export).

## Value

NULL or list of plot objects, if dir_path is NULL.

---

plot_sample_clustering

*Plot heatmaps for pairwise similarity between features.*

---

## Description

This method creates a heatmap based on data stored in a familiarCollection object. Features in the heatmap are ordered so that more similar features appear together.

## Usage

```
plot_sample_clustering(
  object,
  features = waiver(),
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  sample_cluster_method = waiver(),
  sample_linkage_method = waiver(),
  sample_limit = waiver(),
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  gradient_palette = NULL,
  gradient_palette_range = waiver(),
  outcome_palette = NULL,
```

```
  outcome_palette_range = waiver(),
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  outcome_legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 3L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 3L,
  y_breaks = NULL,
  rotate_x_tick_labels = waiver(),
  remove_feature_labels = FALSE,
  remove_sample_labels = FALSE,
  show_feature_dendrogram = TRUE,
  show_sample_dendrogram = TRUE,
  show_normalised_data = TRUE,
  show_outcome = TRUE,
  dendrogram_height = grid::unit(1.5, "cm"),
  outcome_height = grid::unit(0.3, "cm"),
  evaluation_times = waiver(),
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  verbose = TRUE,
  ...
)

## S4 method for signature 'ANY'
plot_sample_clustering(
  object,
  features = waiver(),
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  sample_cluster_method = waiver(),
  sample_linkage_method = waiver(),
  sample_limit = waiver(),
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
```

```
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  gradient_palette = NULL,
  gradient_palette_range = waiver(),
  outcome_palette = NULL,
  outcome_palette_range = waiver(),
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  outcome_legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 3L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 3L,
  y_breaks = NULL,
  rotate_x_tick_labels = waiver(),
  remove_feature_labels = FALSE,
  remove_sample_labels = FALSE,
  show_feature_dendrogram = TRUE,
  show_sample_dendrogram = TRUE,
  show_normalised_data = TRUE,
  show_outcome = TRUE,
  dendrogram_height = grid::unit(1.5, "cm"),
  outcome_height = grid::unit(0.3, "cm"),
  evaluation_times = waiver(),
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  verbose = TRUE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_sample_clustering(
  object,
  features = waiver(),
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  sample_cluster_method = waiver(),
  sample_linkage_method = waiver(),
```

```
  sample_limit = waiver(),
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  gradient_palette = NULL,
  gradient_palette_range = waiver(),
  outcome_palette = NULL,
  outcome_palette_range = waiver(),
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  outcome_legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 3L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 3L,
  y_breaks = NULL,
  rotate_x_tick_labels = waiver(),
  remove_feature_labels = FALSE,
  remove_sample_labels = FALSE,
  show_feature_dendrogram = TRUE,
  show_sample_dendrogram = TRUE,
  show_normalised_data = TRUE,
  show_outcome = TRUE,
  dendrogram_height = grid::unit(1.5, "cm"),
  outcome_height = grid::unit(0.3, "cm"),
  evaluation_times = waiver(),
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  verbose = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| features | Features that should be considered for extracting information from. Typically called in external workflows, e.g. for plotting. Internally, i.e. from summon_familiar, this variable is not used. |
| feature_cluster_method | The method used to perform clustering. These are the same methods as for the cluster_method configuration parameter: none, hclust, agnes, diana and pam.<br>none cannot be used when extracting data regarding mutual correlation or feature expressions.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| feature_linkage_method | The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| sample_cluster_method | The method used to perform clustering based on distance between samples. These are the same methods as for the cluster_method configuration parameter: hclust, agnes, diana and pam.<br>none cannot be used when extracting data for feature expressions.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| sample_linkage_method | The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| sample_limit | (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.<br>This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. list("sample_similarity"=100, "permutation_vimp"=1000).<br>This parameter can be set for the following data elements: sample_similarity, shap, permutation_vimp, and ice_data. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where created performance plots are saved to. Output is saved in the feature_similarity subdirectory. If NULL no figures are saved, but are returned instead. |

split_by (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

x_axis_by (*optional*) Variable plotted along the x-axis of a plot. The variable cannot overlap with variables provided to the split_by and y_axis_by arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables.

y_axis_by (*optional*) Variable plotted along the y-axis of a plot. The variable cannot overlap with variables provided to the split_by and x_axis_by arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

ggtheme (*optional*) ggplot theme to use for plotting.

gradient_palette (*optional*) Sequential or divergent palette used to colour the similarity or distance between features in a heatmap. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

gradient_palette_range (*optional*) Numerical range used to span the gradient. This should be a range of two values, e.g. c(0, 1). Lower or upper boundary can be unset by using NA. If not set, the full metric-specific range is used.

outcome_palette (*optional*) Sequential (continuous outcomes) or qualitative (other outcome types) palette used to show outcome values. This argument is ignored if the outcome is not shown.

outcome_palette_range (*optional*) Numerical range used to span the gradient of numeric (continuous) outcome values. This argument is ignored for other outcome types or if the outcome is not shown.

x_label (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

x_label_shared (*optional*) Sharing of x-axis labels between facets. One of three values:

- overall: A single label is placed at the bottom of the figure. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).

- `column`: A label is placed at the bottom of each column. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).
- `individual`: A label is placed below each facet plot. Tick text is kept.

`y_label` (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

`y_label_shared` (*optional*) Sharing of y-axis labels between facets. One of three values:

- `overall`: A single label is placed to the left of the figure. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).
- `row`: A label is placed to the left of each row. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).
- `individual`: A label is placed below each facet plot. Tick text is kept.

`legend_label` (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

`outcome_legend_label` (*optional*) Label to provide to the legend for outcome data. If NULL, the legend will not have a name. By default, `class`, `value` and `event` are used for `binomial` and `multinomial`, `continuous`, and `survival` outcome types, respectively.

`plot_title` (*optional*) Label to provide as figure title. If NULL, no title is shown.

`plot_sub_title` (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

`caption` (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

`x_range` (*optional*) Value range for the x-axis.

`x_n_breaks` (*optional*) Number of breaks to show on the x-axis of the plot. `x_n_breaks` is used to determine the `x_breaks` argument in case it is unset.

`x_breaks` (*optional*) Break points on the x-axis of the plot.

`y_range` (*optional*) Value range for the y-axis.

`y_n_breaks` (*optional*) Number of breaks to show on the y-axis of the plot. `y_n_breaks` is used to determine the `y_breaks` argument in case it is unset.

`y_breaks` (*optional*) Break points on the y-axis of the plot.

`rotate_x_tick_labels` (*optional*) Rotate tick labels on the x-axis by 90 degrees. Defaults to `TRUE`. Rotation of x-axis tick labels may also be controlled through the `ggtheme`. In this case, `FALSE` should be provided explicitly.

`remove_feature_labels` (*optional*) If `TRUE`, feature labels are not shown. By default, feature labels are shown.

`remove_sample_labels` (*optional*) If `TRUE`, sample labels are not shown. By default, feature labels are shown.

`show_feature_dendrogram` (*optional*) Show feature dendrogram around the main panel. Can be `TRUE`, `FALSE`, `NULL`, or a position, i.e. `top`, `bottom`, `left` and `right`.

If a position is specified, it should be appropriate with regard to the `x_axis_by` or `y_axis_by` argument. If `x_axis_by` is `sample` (default), the only valid positions are `top` (default) and `bottom`. Alternatively, if `y_axis_by` is `feature`, the only valid positions are `right` (default) and `left`.

A dendrogram can only be drawn from cluster methods that produce dendograms, such as hclust. A dendogram can for example not be constructed using the partioning around medioids method (pam).

show_sample_dendrogram

(*optional*) Show sample dendrogram around the main panel. Can be TRUE, FALSE, NULL, or a position, i.e. top, bottom, left and right.

If a position is specified, it should be appropriate with regard to the x_axis_by or y_axis_by argument. If y_axis_by is sample (default), the only valid positions are right (default) and left. Alternatively, if x_axis_by is sample, the only valid positions are top (default) and bottom.

A dendrogram can only be drawn from cluster methods that produce dendograms, such as hclust. A dendogram can for example not be constructed using the partioning around medioids method (pam).

show_normalised_data

(*optional*) Flag that determines whether the data shown in the main heatmap is normalised using the same settings as within the analysis (fixed; default), using a standardisation method (set_normalisation) that is applied separately to each dataset, or not at all (none), which shows the data at the original scale, albeit with batch-corrections.

Categorial variables are plotted to span 90% of the entire numerical value range, i.e. the levels of categorical variables with 2 levels are represented at 5% and 95% of the range, with 3 levels at 5%, 50%, and 95%, etc.

show_outcome (*optional*) Show outcome column(s) or row(s) in the graph. Can be TRUE, FALSE, NULL or a poistion, i.e. top, bottom, left and right.

If a position is specified, it should be appropriate with regard to the x_axis_by or y_axis_by argument. If y_axis_by is sample (default), the only valid positions are left (default) and right. Alternatively, if x_axis_by is sample, the only valid positions are top (default) and bottom.

The outcome data will be drawn between the main panel and the sample dendrogram (if any).

dendrogram_height

(*optional*) Height of the dendrogram. The height is 1.5 cm by default. Height is expected to be grid unit (see grid::unit), which also allows for specifying relative heights.

outcome_height (*optional*) Height of an outcome data column/row. The height is 0.3 cm by default. Height is expected to be a grid unit (see grid::unit), which also allows for specifying relative heights. In case of survival outcome data with multipe evaluation_times, this height is multiplied by the number of time points.

evaluation_times

(*optional*) Times at which the event status of time-to-event survival outcomes are determined. Only used for survival outcome. If not specified, the values used when creating the underlying familiarData objects are used.

width (*optional*) Width of the plot. A default value is derived from the number of facets.

height (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection
(*optional*) Exports the collection if TRUE.

verbose Flag to indicate whether feedback should be provided for the plotting.

... Arguments passed on to as_familiar_collection, ggplot2::ggsave, extract_feature_expressior

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

feature_similarity Table containing pairwise distance between sample. This is used to determine cluster information, and indicate which samples are similar. The table is created by the extract_sample_similarity method.

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

feature_similarity_metric Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and feature_similarity_metric therefore has the same options as cluster_similarity_metric: mcfadden_r2, cox_snell_r2, nagelkerke_r2, spearman, kendall and pearson.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

sample_similarity_metric Metric to determine pairwise similarity between samples. Similarity is computed in the same manner as for clustering, but sample_similarity_metric has different options that are better suited to computing distance between samples instead of between features: gower, euclidean.
The underlying feature data is scaled to the $[0, 1]$ range (for numerical features) using the feature values across the samples. The normalisation parameters required can optionally be computed from feature data with the outer 5% (on both sides) of feature values trimmed or winsorised. To do

so append `_trim` (trimming) or `_winsor` (winsorising) to the metric name. This reduces the effect of outliers somewhat.
If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`message_indent` Number of indentation steps for messages shown during computation and extraction of various data elements.

## Details

This function generates area under the ROC curve plots.

Available splitting variables are: `vimp_method`, `learner`, and `data_set`. By default, the data is split by `vimp_method` and `learner` and `data_set`, since the number of samples will typically differ between data sets, even for the same variable importance method and learner.

The `x_axis_by` and `y_axis_by` arguments determine what data are shown along which axis. Each argument takes one of `feature` and `sample`, and both arguments should be unique. By default, features are shown along the x-axis and samples along the y-axis.

Note that similarity is determined based on the underlying data. Hence the ordering of features may differ between facets, and tick labels are maintained for each panel.

Labeling methods such as `set_vimp_method_names` or `set_data_set_names` can be applied to the `familiarCollection` object to update labels, and order the output in the figure.

This plot can be created from `dataObject`, or `data.table` objects. For `data.table`, see `as_data_object` for additional arguments.

## Value

`NULL` or list of plot objects, if `dir_path` is `NULL`.

---

# plot_shap_dependence *Create SHAP dependence plot.*

---

## Description

This method creates a SHAP dependence plot that shows the dependence of the SHAP value of a feature against its value.

## Usage

```
plot_shap_dependence(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
```

```
  gradient_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  shap_feature = NULL,
  interaction_feature = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_shap_dependence(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  shap_feature = NULL,
  interaction_feature = NULL,
  width = waiver(),
```

```
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_shap_dependence(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  shap_feature = NULL,
  interaction_feature = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | `familiarCollection` object, or one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some `familiarData` objects can be created from prediction tables (`familiarDataElementPredictionTable`). Other `familiarData` objects can be created from data (`dataObject`, or `data.table`). Please check *details* for more information. |

`draw` (*optional*) Draws the plot if TRUE.

`dir_path` (*optional*) Path to the directory where created SHAP dependence plots are saved to. Output is saved in the `explanation` subdirectory. If `NULL` no figures are saved, but are returned instead.

`split_by` (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

`facet_by` (*optional*) Variables used to determine how and if facets of each figure appear. In case the `facet_wrap_cols` argument is `NULL`, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the `split_by` argument, but may overlap with other arguments. See details for available variables.

`facet_wrap_cols` (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

`ggtheme` (*optional*) `ggplot` theme to use for plotting.

`discrete_palette` (*optional*) Divergent or sequential palette used to colour the elements of dependence plots for interactions with another **categorical** feature. `familiar` has a default palette. Other palettes are supported by the `paletteer` package, `grDevices::palette.pals()` (requires R >= 4.0.0), `grDevices::hcl.pals()` (requires R >= 3.6.0) and `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` and `cm.colors`, which correspond to the palettes of the same name in `grDevices`. You may also specify your own palette by providing a vector of colour names listed by `grDevices::colors()` or through hexadecimal RGB strings.

If no `interaction_feature` is set, or is a numerical feature, the gradient palette is not used.

`gradient_palette` (*optional*) Divergent or sequential palette used to colour the elements of dependence plots for interactions with another **numeric** feature. `familiar` has a default palette. Other palettes are supported by the `paletteer` package, `grDevices::palette.pals()` (requires R >= 4.0.0), `grDevices::hcl.pals()` (requires R >= 3.6.0) and `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` and `cm.colors`, which correspond to the palettes of the same name in `grDevices`. You may also specify your own palette by providing a vector of colour names listed by `grDevices::colors()` or through hexadecimal RGB strings.

If no `interaction_feature` is set, or is a categorical feature, the gradient palette is not used.

`x_label` (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

`y_label` (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

`legend_label` (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

`plot_title` (*optional*) Label to provide as figure title. If NULL, no title is shown.

`plot_sub_title` (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

`caption` (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

x_range (*optional*) Value range for the x-axis.

x_n_breaks (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

x_breaks (*optional*) Break points on the x-axis of the plot.

y_range (*optional*) Value range for the y-axis.

y_n_breaks (*optional*) Number of breaks to show on the y-axis of the plot. y_n_breaks is used to determine the y_breaks argument in case it is unset.

y_breaks (*optional*) Break points on the y-axis of the plot.

shap_feature (*optional*) Feature(s) whose SHAP values are used for creating the SHAP dependence plot.

interaction_feature
(*optional*) Feature(s) whose values are used to colour points of the shap_feature.

width (*optional*) Width of the plot. A default value is derived from the number of facets.

height (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection
(*optional*) Exports the collection if TRUE.

... Arguments passed on to extract_performance, as_familiar_collection, ggplot2::ggsave

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

metric One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.

hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.

A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").

This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.

If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.

- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

aggregate_results (*optional*) Flag that signifies whether results should be aggregated during evaluation. If estimation_type is bias_correction or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.
The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE, , "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

## Details

This function creates SHAP dependence plots, which show how the marginal contributions of a feature to the predicted value depend on its value.

Available splitting variables are: vimp_method, learner, data_set, evaluation_time (survival outcome only) and positive_class (categorical outcomes). The default is to facet by evaluation_time or positive_class, and split by vimp_method, learner and data_set. color_by is not used.

Labelling methods such as set_vimp_method_names or set_learner_names can be applied to the familiarCollection object to update labels, and order the output in the figure.

## Value

NULL or list of plot objects, if dir_path is NULL.

---

# plot_shap_force *Create SHAP force plot*

---

## Description

This method creates plots that show stacked SHAP force values obtained from the data stored in a familiarCollection object.

## Usage

```
plot_shap_force(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
```

```
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  highlight_feature = NULL,
  sample_order = "prediction",
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_shap_force(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  highlight_feature = NULL,
  sample_order = "prediction",
  width = waiver(),
```

```
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_shap_force(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  x_label = waiver(),
  x_label_shared = "column",
  y_label = waiver(),
  y_label_shared = "row",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  highlight_feature = NULL,
  sample_order = "prediction",
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | `familiarCollection` object, or one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some `familiarData` objects can be created from prediction tables (`familiarDataElementPredictionTable`). Other `familiarData` objects can be created from data (`dataObject`, or `data.table`). Please check *details* for more information. |

draw (*optional*) Draws the plot if TRUE.

dir_path (*optional*) Path to the directory where created SHAP force plots are saved to. Output is saved in the explanation subdirectory. If NULL no figures are saved, but are returned instead.

split_by (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

x_axis_by (*optional*) Variable plotted along the x-axis of a plot. The variable cannot overlap with variables provided to the split_by and y_axis_by arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables.

y_axis_by (*optional*) Variable plotted along the y-axis of a plot. The variable cannot overlap with variables provided to the split_by and x_axis_by arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

ggtheme (*optional*) ggplot theme to use for plotting.

discrete_palette (*optional*) Discrete palette used to colour the elements of force plots. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0). You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

x_label_shared (*optional*) Sharing of x-axis labels between facets. One of three values:

- overall: A single label is placed at the bottom of the figure. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).
- column: A label is placed at the bottom of each column. Tick text (but not the ticks themselves) is removed for all but the bottom facet plot(s).
- individual: A label is placed below each facet plot. Tick text is kept.

y_label (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

y_label_shared (*optional*) Sharing of y-axis labels between facets. One of three values:

- overall: A single label is placed to the left of the figure. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).
- row: A label is placed to the left of each row. Tick text (but not the ticks themselves) is removed for all but the left-most facet plot(s).

- `individual`: A label is placed below each facet plot. Tick text is kept.

`legend_label` (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

`plot_title` (*optional*) Label to provide as figure title. If NULL, no title is shown.

`plot_sub_title` (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

`caption` (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

`y_range` (*optional*) Value range for the y-axis.

`y_n_breaks` (*optional*) Number of breaks to show on the y-axis of the plot. `y_n_breaks` is used to determine the `y_breaks` argument in case it is unset.

`y_breaks` (*optional*) Break points on the y-axis of the plot.

`highlight_feature` (*optional*) Name of one or more features that should be highlighted in the force plot.

`sample_order` (*optional*) Ordering of samples, one of:

- `prediction`: samples are ordered by increasing predicted value. Sample order between facets may differ.
- `original`: samples retain the original ordering. Sample order between facets is consistent.

`width` (*optional*) Width of the plot. A default value is derived from the number of facets.

`height` (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

`units` (*optional*) Plot size unit. Either `cm` (default), `mm` or `in`.

`export_collection` (*optional*) Exports the collection if TRUE.

`...` Arguments passed on to `extract_performance`, `as_familiar_collection`, `ggplot2::ggsave`

`data` A `dataObject` object, `data.table` or `data.frame` that constitutes the data that are assessed.

`is_pre_processed` Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the `data` argument is a `data.table` or `data.frame`.

`cl` Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallellisation.

`evaluation_times` One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects. Only used for `survival` outcomes.

`ensemble_method` Method for ensembling predictions from models for the same sample. Available methods are:

- `median` (default): Use the median of the predicted values as the ensemble value for a sample.

- mean: Use the mean of the predicted values as the ensemble value for a sample.

metric One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.

hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.

A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid"). This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.

If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`. This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`aggregate_results` (*optional*) Flag that signifies whether results should be aggregated during evaluation. If `estimation_type` is `bias_correction` or `bc`, aggregation leads to a single bias-corrected estimate. If `estimation_type` is `bootstrap_confidence_interval` or `bci`, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if `estimation_type` is `point`.
The default value is equal to `TRUE` except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with `detail_level` and `estimation_type`, a non-default `aggregate_results` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"=TRUE, , "model_performance"=FALSE)`. This parameter exists for the same elements as `estimation_type`.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

## Details

This function plots model performance based on empirical bootstraps, using various plot representations.

Available splitting variables are: vimp_method, learner, data_set, evaluation_time (survival outcome only) and positive_class (categorical outcomes). The default for is to facet by evaluation_time or positive_class, and split by vimp_method, learner and data_set. color_by is not used.

Labelling methods such as set_vimp_method_names or set_learner_names can be applied to the familiarCollection object to update labels, and order the output in the figure.

## Value

NULL or list of plot objects, if dir_path is NULL.

---

# plot_shap_summary *Plot SHAP summary.*

---

## Description

This method creates plots that show a summary of SHAP values obtained from the data stored in a familiarCollection object.

## Usage

```
plot_shap_summary(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  plot_type = NULL,
  value_representation = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_shap_summary(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  plot_type = NULL,
  value_representation = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
```

```
  gradient_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_shap_summary(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  plot_type = NULL,
  value_representation = NULL,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
```

```
  ...
)
```

## Arguments

| | |
|---|---|
| object | familiarCollection object, or one or more familiarData objects, that will be internally converted to a familiarCollection object. It is also possible to provide a familiarEnsemble or one or more familiarModel objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some familiarData objects can be created from prediction tables (familiarDataElementPredictionTable). Other familiarData objects can be created from data (dataObject, or data.table). Please check *details* for more information. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where created SHAP summary plots are saved to. Output is saved in the explanation subdirectory. If NULL no figures are saved, but are returned instead. |
| split_by | (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables. |
| x_axis_by | (*optional*) Variable plotted along the x-axis of a plot. The variable cannot overlap with variables provided to the split_by and y_axis_by arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables. |
| y_axis_by | (*optional*) Variable plotted along the y-axis of a plot. The variable cannot overlap with variables provided to the split_by and x_axis_by arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables. |
| color_by | (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables. |
| facet_by | (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables. |
| facet_wrap_cols | (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead. |
| plot_type | (*optional*) Type of plot to draw. This is one of swarmplot (draws a beeswarm plot), barplot (draws a barplot), boxplot (draws a boxplot) and violinplot (draws a violin plot). Defaults to boxplot if a single SHAP value is available for each feature, and swarmplot otherwise.<br>The choice for plot_type affects several other arguments. |

value_representation
: (*optional*) Indicates how SHAP values are represented, with the following options:
  - raw (default for swarmplot, boxplot, and violinplot plot types): uses SHAP values as they are.
  - abs: uses absolute value of SHAP values.
  - abs_mean (default for barplot plot type): uses mean absolute value of SHAP values. Only used by barplot.
  - abs_max: uses maximum absolute value of SHAP values. Only used by barplot.
  - abs_min: uses minimum absolute value of SHAP values. Only used by barplot.

  If abs_mean, abs_max or abs_min are chosen, plot_type automatically switches to barplot.

ggtheme
: (*optional*) ggplot theme to use for plotting.

discrete_palette
: (*optional*) Palette for colouring plot elements indicated by the color_by argument (if any). Only used if plot_type is not swarmplot. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

gradient_palette
: (*optional*) Sequential or divergent palette used to colour the points the raster in the default swarmplot plots. This argument is not used for other plot_type value. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label
: (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label
: (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label
: (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title
: (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title
: (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption
: (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

limit_n_features
: (*optional*) The number of features that should be included in the plot. Only the most important features are shown. By default, the number of features is not limited.

x_range
: (*optional*) Value range for the x-axis.

x_n_breaks (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

x_breaks (*optional*) Break points on the x-axis of the plot.

width (*optional*) Width of the plot. A default value is derived from the number of facets.

height (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection (*optional*) Exports the collection if TRUE.

... Arguments passed on to extract_performance, as_familiar_collection, ggplot2::ggsave

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

metric One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.

- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- `model`: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For `ensemble` and `model` these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.
A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.
If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`.

This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

aggregate_results (*optional*) Flag that signifies whether results should be aggregated during evaluation. If estimation_type is bias_correction or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.
The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE, , "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default).

If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

## Details

This function creates SHAP summary plots, which provide an overview of marginal contributions of feature values to the predicted values.

Available splitting variables are: vimp_method, learner, data_set, evaluation_time (survival outcome only) and positive_class (categorical outcomes). The default for value_representation = "raw" is to facet by evaluation_time or positive_class, and split by vimp_method and learner. color_by is not used. The default for other value_representation is to color_by evaluation_time or positive_class, and split by vimp_method, learner and data_set.

Labelling methods such as set_vimp_method_names or set_learner_names can be applied to the familiarCollection object to update labels, and order the output in the figure.

## Value

NULL or list of plot objects, if dir_path is NULL.

---

# plot_shap_waterfall *Create SHAP waterfall plot*

---

## Description

This method creates plots that show a waterfall of SHAP values obtained from the data stored in a familiarCollection object.

## Usage

```
plot_shap_waterfall(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  gradient_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
```

```
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_shap_waterfall(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  gradient_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_shap_waterfall(
  object,
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  x_axis_by = NULL,
  y_axis_by = NULL,
  facet_by = NULL,
```

```
  facet_wrap_cols = NULL,
  ggtheme = NULL,
  gradient_palette = NULL,
  x_label = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | `familiarCollection` object, or one or more `familiarData` objects, that will be internally converted to a `familiarCollection` object. It is also possible to provide a `familiarEnsemble` or one or more `familiarModel` objects together with the data from which data is computed prior to export. Paths to such files can also be provided.<br>Additionally, some `familiarData` objects can be created from prediction tables (`familiarDataElementPredictionTable`). Other `familiarData` objects can be created from data (`dataObject`, or `data.table`). Please check *details* for more information. |
| draw | (*optional*) Draws the plot if TRUE. |
| dir_path | (*optional*) Path to the directory where created plots are saved to. Output is saved in the `explanation` subdirectory. If `NULL` no figures are saved, but are returned instead. |
| split_by | (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables. |
| x_axis_by | (*optional*) Variable plotted along the x-axis of a plot. The variable cannot overlap with variables provided to the `split_by` and `y_axis_by` arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables. |
| y_axis_by | (*optional*) Variable plotted along the y-axis of a plot. The variable cannot overlap with variables provided to the `split_by` and `x_axis_by` arguments (if used), but may overlap with other arguments. Only one variable is allowed for this argument. See details for available variables. |

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

ggtheme (*optional*) ggplot theme to use for plotting.

gradient_palette (*optional*) Divergent palette used to colour the elements of waterfall plots. familiar has a default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

limit_n_features (*optional*) The number of features that should be included in the plot. Only the most important features are shown. By default, the number of features is not limited.

x_range (*optional*) Value range for the x-axis.

x_n_breaks (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

x_breaks (*optional*) Break points on the x-axis of the plot.

width (*optional*) Width of the plot. A default value is derived from the number of facets.

height (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection (*optional*) Exports the collection if TRUE.

... Arguments passed on to extract_performance, as_familiar_collection, ggplot2::ggsave

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

is_pre_processed Flag that indicates whether the data was already pre-processed externally, e.g. normalised and clustered. Only used if the data argument is a data.table or data.frame.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

evaluation_times One or more time points that are used for in analysis of survival problems when data has to be assessed at a set time, e.g. calibration. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. Only used for survival outcomes.

ensemble_method Method for ensembling predictions from models for the same sample. Available methods are:

- median (default): Use the median of the predicted values as the ensemble value for a sample.
- mean: Use the mean of the predicted values as the ensemble value for a sample.

metric One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics. If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

detail_level (*optional*) Sets the level at which results are computed and aggregated.

- ensemble: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- hybrid (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to ensemble where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.
- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a

respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For `hybrid`, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using `hybrid` are at least as wide as those for `ensemble`. `hybrid` offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
`hybrid` is generally computationally less expensive then `ensemble`, which in turn is somewhat less expensive than `model`.
A non-default `detail_level` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="ensemble", "model_performance"="hybrid")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, `prediction_data` and `confusion_matrix`.
If results are computed from 10 samples or fewer, `ensemble` is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for `hybrid` or `model`.

`estimation_type` (*optional*) Sets the type of estimation that should be possible. This has the following options:

- `point`: Point estimates.
- `bias_correction` or `bc`: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and `familiar` may bootstrap the data to create them.
- `bootstrap_confidence_interval` or `bci` (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the `confidence_level` parameter, and `familiar` may bootstrap the data to create them.

As with `detail_level`, a non-default `estimation_type` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"="bci", "model_performance"="point")`.
This parameter can be set for the following data elements: `auc_data`, `decision_curve_analyis`, `model_performance`, `permutation_vimp`, `ice_data`, and `prediction_data`.

`aggregate_results` (*optional*) Flag that signifies whether results should be aggregated during evaluation. If `estimation_type` is `bias_correction` or `bc`, aggregation leads to a single bias-corrected estimate. If `estimation_type` is `bootstrap_confidence_interval` or `bci`, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if `estimation_type` is `point`.
The default value is equal to `TRUE` except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with `detail_level` and `estimation_type`, a non-default `aggregate_results` parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. `list("auc_data"=TRUE, , "model_performance"=FALSE)`. This parameter exists for the same elements as `estimation_type`.

`confidence_level` (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the

confidence intervals bootstrap estimation, `familiar` uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is `0.95`.

`bootstrap_ci_method` (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- `percentile` (default): Confidence intervals obtained using the percentile method.
- `bc`: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

`familiar_data_names` Names of the dataset(s). Only used if the `object` parameter is one or more `familiarData` objects.

`collection_name` Name of the collection.

`device` Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If `NULL` (default), the device is guessed based on the `filename` extension.

`scale` Multiplicative scaling factor.

`dpi` Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

`limitsize` When `TRUE` (the default), `ggsave()` will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

`bg` Background colour. If `NULL`, uses the `plot.background` fill value from the plot theme.

`create.dir` Whether to create new directories if a non-existing directory is specified in the `filename` or `path` (`TRUE`) or return an error (`FALSE`, default). If `FALSE` and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

## Details

This function creates SHAP waterfall plots, which show the individual marginal contributions of feature values to the predicted value.

Available splitting variables are: `vimp_method`, `learner`, `data_set`, `evaluation_time` (survival outcome only) and `positive_class` (categorical outcomes), `sample_id`. The default for is to facet by `evaluation_time` or `positive_class`, and split by `vimp_method`, `learner`, `data_set`, and `sample_id`. `color_by` is not used.

Labelling methods such as `set_vimp_method_names` or `set_learner_names` can be applied to the `familiarCollection` object to update labels, and order the output in the figure.

## Value

`NULL` or list of plot objects, if `dir_path` is `NULL`.

plot_univariate_importance *Plot univariate importance.*

## Description

This function plots the univariate analysis data stored in a familiarCollection object.

## Usage

```
plot_univariate_importance(
  object,
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  draw = FALSE,
  dir_path = NULL,
  p_adjustment_method = waiver(),
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  show_cluster = TRUE,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = waiver(),
  x_label = waiver(),
  y_label = "feature",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  significance_level_shown = 0.05,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  verbose = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
```

```
plot_univariate_importance(
  object,
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  draw = FALSE,
  dir_path = NULL,
  p_adjustment_method = waiver(),
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  show_cluster = TRUE,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = waiver(),
  x_label = waiver(),
  y_label = "feature",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  significance_level_shown = 0.05,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  verbose = TRUE,
  export_collection = FALSE,
  ...
)

## S4 method for signature 'familiarCollection'
plot_univariate_importance(
  object,
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  draw = FALSE,
  dir_path = NULL,
  p_adjustment_method = waiver(),
  split_by = NULL,
  color_by = NULL,
```

```
  facet_by = NULL,
  facet_wrap_cols = NULL,
  show_cluster = TRUE,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = waiver(),
  x_label = waiver(),
  y_label = "feature",
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  x_range = NULL,
  x_n_breaks = 5L,
  x_breaks = NULL,
  significance_level_shown = 0.05,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  verbose = TRUE,
  export_collection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiarCollection object, or other other objects from which a familiarCollection can be extracted. See details for more information. |
| feature_cluster_method | The method used to perform clustering. These are the same methods as for the cluster_method configuration parameter: none, hclust, agnes, diana and pam.<br>none cannot be used when extracting data regarding mutual correlation or feature expressions.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| feature_linkage_method | The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.<br>If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects. |
| feature_cluster_cut_method | The method used to divide features into separate clusters. The available methods are the same as for the cluster_cut_method configuration parameter: silhouette, fixed_cut and dynamic_cut. |

silhouette is available for all cluster methods, but fixed_cut only applies to methods that create hierarchical trees (hclust, agnes and diana). dynamic_cut requires the dynamicTreeCut package and can only be used with agnes and hclust.

If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

feature_similarity_threshold

The threshold level for pair-wise similarity that is required to form feature clusters with the fixed_cut method.

If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

draw (*optional*) Draws the plot if TRUE.

dir_path (*optional*) Path to the directory where created figures are saved to. Output is saved in the variable_importance subdirectory. If NULL no figures are saved, but are returned instead.

p_adjustment_method

(*optional*) Indicates type of p-value that is shown. One of holm, hochberg, hommel, bonferroni, BH, BY, fdr, none, p_value or q_value for adjusted p-values, uncorrected p-values

split_by (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

color_by (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols

(*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

show_cluster (*optional*) Show which features were clustered together.

ggtheme (*optional*) ggplot theme to use for plotting.

discrete_palette

(*optional*) Palette for colouring the plot elements according to the groupings indicated by the color_by argument (if any). familiar has a default palette. Other palettes are supported by paletteer, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

gradient_palette
: (*optional*) Palette to use for filling the bars in case the color_by argument is not set. The bars are then coloured according to their importance. By default, no gradient is used, and the bars are not filled according to importance. Use NULL to fill the bars using the default palette in familiar. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label
: (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

y_label
: (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label
: (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title
: (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title
: (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption
: (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

limit_n_features
: (*optional*) The number of features that should be included in the plot. Only the most important features are shown. By default, the number of features is not limited.

x_range
: (*optional*) Value range for the x-axis.

x_n_breaks
: (*optional*) Number of breaks to show on the x-axis of the plot. x_n_breaks is used to determine the x_breaks argument in case it is unset.

x_breaks
: (*optional*) Break points on the x-axis of the plot.

significance_level_shown
: Position(s) to draw vertical lines indicating a significance level, e.g. 0.05. Can be NULL to not draw anything.

width
: (*optional*) Width of the plot. A default value is derived from the number of facets.

height
: (*optional*) Height of the plot. A default value is derived from the number of features and the number of facets.

units
: (*optional*) Plot size unit. Either cm (default), mm or in.

verbose
: Flag to indicate whether feedback should be provided for the plotting.

export_collection
: (*optional*) Exports the collection if TRUE.

...
: Arguments passed on to as_familiar_collection, ggplot2::ggsave, extract_univariate_analysi

  familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

  collection_name Name of the collection.

  device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

data A dataObject object, data.table or data.frame that constitutes the data that are assessed.

cl Cluster created using the parallel package. This cluster is then used to speed up computation through parallellisation.

feature_similarity_metric Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and feature_similarity_metric therefore has the same options as cluster_similarity_metric: mcfadden_r2, cox_snell_r2, nagelkerke_r2, spearman, kendall and pearson.
If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

icc_type String indicating the type of intraclass correlation coefficient (1, 2 or 3) that should be used to compute robustness for features in repeated measurements during the evaluation of univariate importance. These types correspond to the types in Shrout and Fleiss (1979). If not provided explicitly, this parameter is read from settings used at creation of the underlying familiarModel objects.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

## Details

This function generates a horizontal barplot with the length of the bars corresponding to the 10-logarithm of the (multiple-testing corrected) p-value.

Features are assessed univariately using one-sample location t-tests after fitting a suitable regression model. The fitted model coefficient and the covariance matrix are then used to compute a p-value.

The following splitting variables are available for split_by, color_by and facet_by:

- vimp_method: variable importance methods
- learner: learners
- data_set: data sets

Unlike for plots of feature ranking in initial variable importance computation and after modelling (as assessed by model-specific routines), clusters of features are now found during creation of underlying `familiarData` objects, instead of through consensus clustering. Hence, clustering results may differ due to differences in the underlying datasets.

Labelling methods such as `set_vimp_method_names` or `set_feature_names` can be applied to the `familiarCollection` object to update labels, and order the output in the figure.

## Value

`NULL` or list of plot objects, if `dir_path` is `NULL`.

---

`plot_variable_importance` *Plot variable importance scores of features during feature selection or after training a model.*

---

## Description

This function plots variable importance based data obtained during feature selection or after training a model, which are stored in a `familiarCollection` object.

## Usage

```
plot_variable_importance(
  object,
  type,
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  aggregation_method = waiver(),
  rank_threshold = waiver(),
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  show_cluster = TRUE,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = waiver(),
  x_label = "feature",
  rotate_x_tick_labels = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
```

```
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

## S4 method for signature 'ANY'
plot_variable_importance(
  object,
  type,
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  aggregation_method = waiver(),
  rank_threshold = waiver(),
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  show_cluster = TRUE,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = waiver(),
  x_label = "feature",
  rotate_x_tick_labels = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
```

```
  ...
)

## S4 method for signature 'familiarCollection'
plot_variable_importance(
  object,
  type,
  feature_cluster_method = waiver(),
  feature_linkage_method = waiver(),
  feature_cluster_cut_method = waiver(),
  feature_similarity_threshold = waiver(),
  aggregation_method = waiver(),
  rank_threshold = waiver(),
  draw = FALSE,
  dir_path = NULL,
  split_by = NULL,
  color_by = NULL,
  facet_by = NULL,
  facet_wrap_cols = NULL,
  show_cluster = TRUE,
  ggtheme = NULL,
  discrete_palette = NULL,
  gradient_palette = waiver(),
  x_label = "feature",
  rotate_x_tick_labels = waiver(),
  y_label = waiver(),
  legend_label = waiver(),
  plot_title = waiver(),
  plot_sub_title = waiver(),
  caption = NULL,
  limit_n_features = waiver(),
  y_range = NULL,
  y_n_breaks = 5L,
  y_breaks = NULL,
  width = waiver(),
  height = waiver(),
  units = waiver(),
  export_collection = FALSE,
  ...
)

plot_feature_selection_occurrence(...)

plot_feature_selection_variable_importance(...)

plot_model_signature_occurrence(...)

plot_model_signature_variable_importance(...)
```

## Arguments

`object` A `familiarCollection` object, or other other objects from which a `familiarCollection` can be extracted. See details for more information.

`type` Determine what variable importance should be shown. Can be `feature_selection` or `model` for the variable importance after the feature selection step and after the model training step, respectively.

`feature_cluster_method`
The method used to perform clustering. These are the same methods as for the `cluster_method` configuration parameter: `none`, `hclust`, `agnes`, `diana` and `pam`.

`none` cannot be used when extracting data regarding mutual correlation or feature expressions.

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`feature_linkage_method`
The method used for agglomerative clustering in `hclust` and `agnes`. These are the same methods as for the `cluster_linkage_method` configuration parameter: `average`, `single`, `complete`, `weighted`, and `ward`.

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`feature_cluster_cut_method`
The method used to divide features into separate clusters. The available methods are the same as for the `cluster_cut_method` configuration parameter: `silhouette`, `fixed_cut` and `dynamic_cut`.

`silhouette` is available for all cluster methods, but `fixed_cut` only applies to methods that create hierarchical trees (`hclust`, `agnes` and `diana`). `dynamic_cut` requires the `dynamicTreeCut` package and can only be used with `agnes` and `hclust`.

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`feature_similarity_threshold`
The threshold level for pair-wise similarity that is required to form feature clusters with the `fixed_cut` method.

If not provided explicitly, this parameter is read from settings used at creation of the underlying `familiarModel` objects.

`aggregation_method`
(*optional*) The method used to aggregate variable importances over different data subsets, e.g. bootstraps. The following methods can be selected:

- `mean` (default): Use the mean rank of a feature over the subsets to determine the aggregated feature rank.
- `median`: Use the median rank of a feature over the subsets to determine the aggregated feature rank.
- `best`: Use the best rank the feature obtained in any subset to determine the aggregated feature rank.

- worst: Use the worst rank the feature obtained in any subset to determine the aggregated feature rank.
- stability: Use the frequency of the feature being in the subset of highly ranked features as measure for the aggregated feature rank (Meinshausen and Buehlmann, 2010).
- exponential: Use a rank-weighted frequence of occurrence in the subset of highly ranked features as measure for the aggregated feature rank (Haury et al., 2011).
- borda: Use the borda count as measure for the aggregated feature rank (Wald et al., 2012).
- enhanced_borda: Use an occurrence frequency-weighted borda count as measure for the aggregated feature rank (Wald et al., 2012).
- truncated_borda: Use borda count computed only on features within the subset of highly ranked features.
- enhanced_truncated_borda: Apply both the enhanced borda method and the truncated borda method and use the resulting borda count as the aggregated feature rank.

rank_threshold (*optional*) The threshold used to define the subset of highly important features. If not set, this threshold is determined by maximising the variance in the occurrence value over all features over the subset size.

This parameter is only relevant for stability, exponential, enhanced_borda, truncated_borda and enhanced_truncated_borda methods.

draw (*optional*) Draws the plot if TRUE.

dir_path (*optional*) Path to the directory where created figures are saved to. Output is saved in the variable_importance subdirectory. If NULL no figures are saved, but are returned instead.

split_by (*optional*) Splitting variables. This refers to column names on which datasets are split. A separate figure is created for each split. See details for available variables.

color_by (*optional*) Variables used to determine fill colour of plot objects. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_by (*optional*) Variables used to determine how and if facets of each figure appear. In case the facet_wrap_cols argument is NULL, the first variable is used to define columns, and the remaing variables are used to define rows of facets. The variables cannot overlap with those provided to the split_by argument, but may overlap with other arguments. See details for available variables.

facet_wrap_cols (*optional*) Number of columns to generate when facet wrapping. If NULL, a facet grid is produced instead.

show_cluster (*optional*) Show which features were clustered together. Currently not available in combination with variable importance obtained during feature selection.

ggtheme (*optional*) ggplot theme to use for plotting.

discrete_palette
: (*optional*) Palette for colouring the plot elements according to the groupings indicated by the color_by argument (if any). familiar has a default palette. Other palettes are supported by paletteer, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

gradient_palette
: (*optional*) Palette for filling bars if the color_by argument is not set. By default, bars are not coloured. If gradient_palette is set, the palette will colour bars according to feature importance. Use NULL to fill the bars using the familiar default palette. Other palettes are supported by the paletteer package, grDevices::palette.pals() (requires R >= 4.0.0), grDevices::hcl.pals() (requires R >= 3.6.0) and rainbow, heat.colors, terrain.colors, topo.colors and cm.colors, which correspond to the palettes of the same name in grDevices. You may also specify your own palette by providing a vector of colour names listed by grDevices::colors() or through hexadecimal RGB strings.

x_label
: (*optional*) Label to provide to the x-axis. If NULL, no label is shown.

rotate_x_tick_labels
: (*optional*) Rotate tick labels on the x-axis by 90 degrees. Defaults to TRUE. Rotation of x-axis tick labels may also be controlled through the ggtheme. In this case, FALSE should be provided explicitly.

y_label
: (*optional*) Label to provide to the y-axis. If NULL, no label is shown.

legend_label
: (*optional*) Label to provide to the legend. If NULL, the legend will not have a name.

plot_title
: (*optional*) Label to provide as figure title. If NULL, no title is shown.

plot_sub_title
: (*optional*) Label to provide as figure subtitle. If NULL, no subtitle is shown.

caption
: (*optional*) Label to provide as figure caption. If NULL, no caption is shown.

limit_n_features
: (*optional*) The number of features that should be included in the plot. Only the most important features are shown. By default, the number of features is not limited.

y_range
: (*optional*) Value range for the y-axis.

y_n_breaks
: (*optional*) Number of breaks to show on the y-axis of the plot. y_n_breaks is used to determine the y_breaks argument in case it is unset.

y_breaks
: (*optional*) Break points on the y-axis of the plot.

width
: (*optional*) Width of the plot. A default value is derived from the number of facets and the number of features.

height
: (*optional*) Height of the plot. A default value is derived from number of facets, and the length of the longest feature name (if rotate_x_tick_labels is TRUE).

units
: (*optional*) Plot size unit. Either cm (default), mm or in.

export_collection
: (*optional*) Exports the collection if TRUE.

... Arguments passed on to as_familiar_collection, ggplot2::ggsave, extract_fs_vimp

familiar_data_names Names of the dataset(s). Only used if the object parameter is one or more familiarData objects.

collection_name Name of the collection.

device Device to use. Can either be a device function (e.g. png), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only). If NULL (default), the device is guessed based on the filename extension.

scale Multiplicative scaling factor.

dpi Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). Only applies when converting pixel units, as is typical for raster output types.

limitsize When TRUE (the default), ggsave() will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.

bg Background colour. If NULL, uses the plot.background fill value from the plot theme.

create.dir Whether to create new directories if a non-existing directory is specified in the filename or path (TRUE) or return an error (FALSE, default). If FALSE and run in an interactive session, a prompt will appear asking to create a new directory when necessary.

verbose Flag to indicate whether feedback should be provided on the computation and extraction of various data elements.

message_indent Number of indentation steps for messages shown during computation and extraction of various data elements.

## Details

This function generates a barplot based on variable importance of features.

The only allowed values for split_by, color_by or facet_by are vimp_method and learner, but note that learner has no effect when plotting variable importance of features acquired during feature selection.

Labeling methods such as set_feature_names or set_vimp_method_names can be applied to the familiarCollection object to update labels, and order the output in the figure.

## Value

NULL or list of plot objects, if dir_path is NULL.

---

# precompute_data_assignment

*Pre-compute data assignment*

---

## Description

Creates data assignment.

## Usage

```
precompute_data_assignment(
  formula = NULL,
  data = NULL,
  experiment_data = NULL,
  cl = NULL,
  experimental_design = "fs+mb",
  verbose = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| formula | An R formula. The formula can only contain feature names and dot (.). The * and +1 operators are not supported as these refer to columns that are not present in the data set.<br>Use of the formula interface is optional. |
| data | A data.table object, a data.frame object, list containing multiple data.table or data.frame objects, or paths to data files.<br>data should be provided if no file paths are provided to the data_files argument. If both are provided, only data will be used.<br>All data is expected to be in wide format, and ideally has a sample identifier (see sample_id_column), batch identifier (see cohort_column) and outcome columns (see outcome_column).<br>In case paths are provided, the data should be stored as csv, rds or RData files. See documentation for the data_files argument for more information. |
| experiment_data | Experimental data may provided in the form of the output of precompute_data_assignment, precompute_feature_info or precompute_vimp. This allows for warm-starting experiments. |
| cl | Cluster created using the parallel package. This cluster is then used to speed up computation through parallelisation. When a cluster is not provided, parallelisation is performed by setting up a cluster on the local machine.<br>This parameter has no effect if the parallel argument is set to FALSE. |
| experimental_design | (**required**) Defines what the experiment looks like, e.g. cv(bt(fs,20)+mb,3,2) for 2 times repeated 3-fold cross-validation with nested variable importance computation on 20 bootstraps and model-building. The basic workflow components are:<br>• fs: (optional) variable importance computation step. If not explicitly declared, feature selection will be done just in time for hyperparameter optimisation.<br>• mb: (required) model building step.<br>• ev: (optional) external validation. If validation batches or cohorts are present in the dataset (data), these should be indicated in the validation_batch_id argument. |

The different components are linked using +.

Different subsampling methods can be used in conjunction with the basic workflow components:

- bs(x,n): (stratified) .632 bootstrap, with n the number of bootstraps. In contrast to bt, feature pre-processing parameters and hyperparameter optimisation are conducted on individual bootstraps.
- bt(x,n): (stratified) .632 bootstrap, with n the number of bootstraps. Unlike bs and other subsampling methods, no separate pre-processing parameters or optimised hyperparameters will be determined for each bootstrap.
- cv(x,n,p): (stratified) n-fold cross-validation, repeated p times. Pre-processing parameters are determined for each iteration.
- lv(x): leave-one-out-cross-validation. Pre-processing parameters are determined for each iteration.
- ip(x): imbalance partitioning for addressing class imbalances on the data set. Pre-processing parameters are determined for each partition. The number of partitions generated depends on the imbalance correction method (see the imbalance_correction_method parameter).

As shown in the example above, sampling algorithms can be nested.

Though neither variable importance is determined nor models are learned within precompute_data_assignment, the corresponding elements are still required to prevent issues when using the resulting experimentData object to warm-start the experiments.

The simplest valid experimental design is fs+mb. This is the default in precompute_data_assignment, and will simply assign all instances to the training set.

verbose — Indicates verbosity of the results. Default is TRUE, and all messages and warnings are returned.

... — Arguments passed on to .parse_experiment_settings, .parse_setup_settings, .parse_preprocessing_settings

batch_id_column (**recommended**) Name of the column containing batch or cohort identifiers. This parameter is required if more than one dataset is provided, or if external validation is performed.
In familiar any row of data is organised by four identifiers:

- The batch identifier batch_id_column: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.
- The sample identifier sample_id_column: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.
- The series identifier series_id_column: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.
- The repetition identifier: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries

for the same series of the same sample in the same batch that share the same outcome are encountered.

`sample_id_column` (**recommended**) Name of the column containing sample or subject identifiers. See `batch_id_column` above for more details.
If unset, every row will be identified as a single sample.

`series_id_column` (*optional*) Name of the column containing series identifiers, which distinguish between measurements that are part of a series for a single sample. See `batch_id_column` above for more details.
If unset, rows which share the same batch and sample identifiers but have a different outcome are assigned unique series identifiers.

`development_batch_id` (*optional*) One or more batch or cohort identifiers to constitute data sets for development. Defaults to all, or all minus the identifiers in `validation_batch_id` for external validation. Required if external validation is performed and `validation_batch_id` is not provided.

`validation_batch_id` (*optional*) One or more batch or cohort identifiers to constitute data sets for external validation. Defaults to all data sets except those in `development_batch_id` for external validation, or none if not. Required if `development_batch_id` is not provided.

`outcome_name` (*optional*) Name of the modelled outcome. This name will be used in figures created by `familiar`.
If not set, the column name in `outcome_column` will be used for `binomial`, `multinomial`, and `continuous` outcomes. For other outcomes (`survival` and `competing_risk`) no default is used.

`outcome_column` (**recommended**) Name of the column containing the outcome of interest. May be identified from a formula, if a formula is provided as an argument. Otherwise an error is raised. Note that `survival` and `competing_risk` outcome type outcomes require two columns that indicate the time-to-event or the time of last follow-up and the event status.

`outcome_type` (**recommended**) Type of outcome found in the outcome column. The outcome type determines many aspects of the overall process, e.g. the available variable importance methods and learners, but also the type of assessments that can be conducted to evaluate the resulting models.
Implemented outcome types are:

- `binomial`: categorical outcome with 2 levels.
- `multinomial`: categorical outcome with 2 or more levels.
- `continuous`: general continuous numeric outcomes.
- `survival`: survival outcome for time-to-event data.

If not provided, the algorithm will attempt to obtain outcome_type from contents of the outcome column. This may lead to unexpected results, and we therefore advise to provide this information manually.
Note that `competing_risk` survival analysis are not fully supported, and is currently not a valid choice for `outcome_type`. The `count` outcome type was deprecated in version 2.0.0, and superseded by `continuous`.

`class_levels` (*optional*) Class levels for `binomial` or `multinomial` outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column.

event_indicator (**recommended**) Indicator for events in survival and competing_risk analyses. familiar will automatically recognise 1, true, t, y and yes as event indicators, including different capitalisations. If this parameter is set, it replaces the default values.

censoring_indicator (**recommended**) Indicator for right-censoring in survival and competing_risk analyses. familiar will automatically recognise 0, false, f, n, no as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values.

competing_risk_indicator (**recommended**) Indicator for competing risks in competing_risk analyses. There are no default values, and if unset, all values other than those specified by the event_indicator and censoring_indicator parameters are considered to indicate competing risks.

signature (*optional*) One or more names of feature columns that are considered part of a specific signature. Features specified here will always be used for modelling. Ranking of variable importances has no effect for these features.

novelty_features (*optional*) One or more names of feature columns that should be included for the purpose of novelty detection.

exclude_features (*optional*) Feature columns that will be removed from the data set. Cannot overlap with features in signature, novelty_features or include_features.

include_features (*optional*) Feature columns that are specifically included in the data set. By default all features are included. Cannot overlap with exclude_features, but may overlap signature. Features in signature and novelty_features are always included. If both exclude_features and include_features are provided, include_features takes precedence, provided that there is no overlap between the two.

reference_method (*optional*) Method used to set reference levels for categorical features. There are several options:

- auto (default): Categorical features that are not explicitly set by the user, i.e. columns containing boolean values or characters, use the most frequent level as reference. Categorical features that are explicitly set, i.e. as factors, are used as is.
- always: Both automatically detected and user-specified categorical features have the reference level set to the most frequent level. Ordinal features are not altered, but are used as is.
- never: User-specified categorical features are used as is. Automatically detected categorical features are simply sorted, and the first level is then used as the reference level. This was the behaviour prior to familiar version 1.3.0.

imbalance_correction_method (*optional*) Type of method used to address class imbalances. Available options are:

- full_undersampling (default): All data will be used in an ensemble fashion. The full minority class will appear in each partition, but majority classes are undersampled until all data have been used.
- random_undersampling: Randomly undersamples majority classes. This is useful in cases where full undersampling would lead to the

formation of many models due major overrepresentation of the largest class.

This parameter is only used in combination with imbalance partitioning in the experimental design, and `ip` should therefore appear in the string that defines the design.

`imbalance_n_partitions` (*optional*) Number of times random undersampling should be repeated. 10 undersampled subsets with balanced classes are formed by default.

`iteration_seed` (*optional*) Integer seed used in sampling algorithms specified by the `experimental_design` argument. This allows for creating the same sample assignments across different experiments – of course provided that the same dataset is used. By default a random seed is used.

`parallel` (*optional*) Enable parallel processing. Defaults to `TRUE`. When set to `FALSE`, this disables all parallel processing, regardless of specific parameters such as `parallel_preprocessing`. However, when `parallel` is `TRUE`, parallel processing of different parts of the workflow can be disabled by setting respective flags to `FALSE`.

`parallel_nr_cores` (*optional*) Number of cores available for parallelisation. Defaults to 2. This setting does nothing if parallelisation is disabled.

`restart_cluster` (*optional*) Restart nodes used for parallel computing to free up memory prior to starting a parallel process. Note that it does take time to set up the clusters. Therefore setting this argument to `TRUE` may impact processing speed. This argument is ignored if `parallel` is `FALSE` or the cluster was initialised outside of familiar. Default is `FALSE`, which causes the clusters to be initialised only once.

`cluster_type` (*optional*) Selection of the cluster type for parallel processing. Available types are the ones supported by the parallel package that is part of the base R distribution: `psock` (default), `fork`, `mpi`, `nws`, `sock`. In addition, `none` is available, which also disables parallel processing.

`backend_type` (*optional*) Selection of the backend for distributing copies of the data. This backend ensures that only a single master copy is kept in memory. This limits memory usage during parallel processing.
Several backend options are available, notably `socket_server`, and `none` (default). `socket_server` is based on the callr package and R sockets, comes with `familiar` and is available for any OS. `none` uses the package environment of familiar to store data, and is available for any OS. However, `none` requires copying of data to any parallel process, and has a larger memory footprint.

`server_port` (*optional*) Integer indicating the port on which the socket server or RServe process should communicate. Defaults to port 6311. Note that ports 0 to 1024 and 49152 to 65535 cannot be used.

`feature_max_fraction_missing` (*optional*) Numeric value between `0.0` and `0.95` that determines the meximum fraction of missing values that still allows a feature to be included in the data set. All features with a missing value fraction over this threshold are not processed further. The default value is `0.30`.

`sample_max_fraction_missing` (*optional*) Numeric value between `0.0` and

`0.95` that determines the maximum fraction of missing values that still allows a sample to be included in the data set. All samples with a missing value fraction over this threshold are excluded and not processed further. The default value is `0.30`.

`filter_method` (*optional*) One or methods used to reduce dimensionality of the data set by removing irrelevant or poorly reproducible features.
Several method are available:

- `none` (default): None of the features will be filtered.
- `low_variance`: Features with a variance below the `low_var_minimum_variance_threshold` are filtered. This can be useful to filter, for example, genes that are not differentially expressed.
- `univariate_test`: Features undergo a univariate regression using an outcome-appropriate regression model. The p-value of the model coefficient is collected. Features with coefficient p-value above the `univariate_test_threshold` are subsequently filtered.
- `robustness`: Features that are not sufficiently robust according to the intraclass correlation coefficient are filtered. Use of this method requires that repeated measurements are present in the data set, i.e. there should be entries for which the sample and cohort identifiers are the same.

More than one method can be used simultaneously. Features with singular values are always filtered, as these do not contain information.

`univariate_test_threshold` (*optional*) Numeric value between `1.0` and `0.0` that determines which features are irrelevant and will be filtered by the `univariate_test`. p-values are compared to this threshold. All features with values above the threshold are filtered. The default value is `0.20`.

`univariate_test_threshold_metric` (*optional*) Metric used with the to compare the `univariate_test_threshold` against. The following metric can be chosen:

- `p_value` (default): The unadjusted p-value of each feature is used for to filter features.

`univariate_test_max_feature_set_size` (*optional*) Maximum size of the feature set after the univariate test. P or q values of features are compared against the threshold, but if the resulting data set would be larger than this setting, only the most relevant features up to the desired feature set size are selected.
The default value is `NULL`, which causes features to be filtered based on their relevance only.

`low_var_minimum_variance_threshold` (required, if used) Numeric value that determines which features will be filtered by the `low_variance` method. The variance of each feature is computed and compared to the threshold. If it is below the threshold, the feature is removed.
This parameter has no default value and should be set if `low_variance` is used.

`low_var_max_feature_set_size` (*optional*) Maximum size of the feature set after filtering features with a low variance. All features are first compared against `low_var_minimum_variance_threshold`. If the resulting feature

set would be larger than specified, only the most strongly varying features will be selected, up to the desired size of the feature set.
The default value is `NULL`, which causes features to be filtered based on their variance only.

`robustness_icc_type` (*optional*) String indicating the type of intraclass correlation coefficient (`1`, `2` or `3`) that should be used to compute robustness for features in repeated measurements. These types correspond to the types in Shrout and Fleiss (1979). The default value is `1`.

`robustness_threshold_metric` (*optional*) String indicating which specific intraclass correlation coefficient (ICC) metric should be used to filter features. This should be one of:

- `icc`: The estimated ICC value itself.
- `icc_low` (default): The estimated lower limit of the 95% confidence interval of the ICC, as suggested by Koo and Li (2016).
- `icc_panel`: The estimated ICC value over the panel average, i.e. the ICC that would be obtained if all repeated measurements were averaged.
- `icc_panel_low`: The estimated lower limit of the 95% confidence interval of the panel ICC.

`robustness_threshold_value` (*optional*) The intraclass correlation coefficient value that is as threshold. The default value is `0.70`.

`transformation_method` (*optional*) The transformation method used to change the distribution of the data to be more normal-like. The following methods are available:

- `none`: This disables transformation of features.
- `yeo_johnson`: Transformation using the location and scale invariant version of the Yeo-Johnson transformation (Yeo and Johnson, 2000; Zwanenburg and Löck, 2026).
- `yeo_johnson_robust` (default): A robust version of `yeo_johnson`. This method is less sensitive to outliers.
- `yeo_johnson_conventional`: As `yeo_johnson`, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Yeo and Johnson (2001).
- `box_cox`: Transformation using the location and scale invariant version of the Box-Cox transformation (Box and Cox, 1964; Zwanenburg and Löck, 2026).
- `box_cox_robust`: A robust version of `yeo_johnson`. This method is less sensitive to outliers.
- `box_cox_conventional`: As `box_cox`, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Box and Cox (1964). This method requires strictly positive feature values.

Transformation requires the `power.transform` package. Only features that contain numerical data are transformed. Transformation parameters obtained in development data are stored within `featureInfo` objects for later use with validation data sets.

`transformation_optimisation_criterion` (*optional*) Transformation parameters are optimised using a criterion, conventionally maximum-likelihood-estimation. `power.transform` implements multiple optimisation criteria, of which the following are available:

- `mle` (default): Optimisation using maximum likelihood estimation.
- `cramer_von_mises`: Optimisation using the Cramér-von Mises criterion. Zwanenburg and Löck (2026) found that this criterion was relatively robust against outliers.

`transformation_gof_test_p_value` (*optional*) Not all transformations will lead to features that are roughly normally distributed. Zwanenburg and Löck (2026) established a empirical goodness-of-fit test for central normality. This parameter sets the significance for rejecting the null-hypothesis that a feature distribution is centrally normal. When the null-hypothesis is rejected, no transformation is performed. The default value is `NULL`, which disables the test.

`normalisation_method` (*optional*) The normalisation method used to improve the comparability between numerical features that may have very different scales. The following normalisation methods can be chosen:

- `none`: This disables feature normalisation.
- `standardisation`: Features are normalised by subtraction of their mean values and division by their standard deviations. This causes every feature to be have a center value of 0.0 and standard deviation of 1.0.
- `standardisation_trim`: As `standardisation`, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- `standardisation_winsor`: As `standardisation`, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- `standardisation_robust` (default): A robust version of `standardisation` that relies on computing Huber's M-estimators for location and scale.
- `normalisation`: Features are normalised by subtraction of their minimum values and division by their ranges. This maps all feature values to a $[0, 1]$ interval.
- `normalisation_trim`: As `normalisation`, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- `normalisation_winsor`: As `normalisation`, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- `quantile`: Features are normalised by subtraction of their median values and division by their interquartile range.
- `mean_centering`: Features are centered by substracting the mean, but do not undergo rescaling.

Only features that contain numerical data are normalised. Normalisation parameters obtained in development data are stored within `featureInfo` objects for later use with validation data sets.

batch_normalisation_method (*optional*) The method used for batch normalisation. Available methods are:

- none (default): This disables batch normalisation of features.
- standardisation: Features within each batch are normalised by subtraction of the mean value and division by the standard deviation in each batch.
- standardisation_trim: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- standardisation_winsor: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- standardisation_robust: A robust version of standardisation that relies on computing Huber's M-estimators for location and scale within each batch.
- normalisation: Features within each batch are normalised by subtraction of their minimum values and division by their range in each batch. This maps all feature values in each batch to a $[0, 1]$ interval.
- normalisation_trim: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- normalisation_winsor: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- quantile: Features in each batch are normalised by subtraction of the median value and division by the interquartile range of each batch.
- mean_centering: Features in each batch are centered on 0.0 by substracting the mean value in each batch, but are not rescaled.
- combat_parametric: Batch adjustments using parametric empirical Bayes (Johnson et al, 2007). combat_p leads to the same method.
- combat_non_parametric: Batch adjustments using non-parametric empirical Bayes (Johnson et al, 2007). combat_np and combat lead to the same method. Note that we reduced complexity from O($n^2$) to O($n$) by only computing batch adjustment parameters for each feature on a subset of 50 randomly selected features, instead of all features.

Only features that contain numerical data are normalised using batch normalisation. Batch normalisation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets, in case the validation data is from the same batch.
If validation data contains data from unknown batches, normalisation parameters are separately determined for these batches.
Note that for both empirical Bayes methods, the batch effect is assumed to produce results across the features. This is often true for things such as gene expressions, but the assumption may not hold generally.
When performing batch normalisation, it is moreover important to check that differences between batches or cohorts are not related to the studied endpoint.

`imputation_method` (*optional*) Method used for imputing missing feature values. Two methods are implemented:

- `simple`: Simple replacement of a missing value by the median value (for numeric features) or the modal value (for categorical features).
- `lasso`: Imputation of missing value by lasso regression (using `glmnet`) based on information contained in other features.

`simple` imputation precedes `lasso` imputation to ensure that any missing values in predictors required for `lasso` regression are resolved. The `lasso` estimate is then used to replace the missing value.

The default value depends on the number of features in the dataset. If the number is lower than 100, `lasso` is used by default, and `simple` otherwise.

Only single imputation is performed. Imputation models and parameters are stored within `featureInfo` objects for later use with validation data sets.

`cluster_method` (*optional*) Clustering is performed to identify and replace redundant features, for example those that are highly correlated. Such features do not carry much additional information and may be removed or replaced instead (Park et al., 2007; Tolosi and Lengauer, 2011).

The cluster method determines the algorithm used to form the clusters. The following cluster methods are implemented:

- `none`: No clustering is performed.
- `hclust` (default): Hierarchical agglomerative clustering. If the `fastcluster` package is installed, `fastcluster::hclust` is used (Muellner 2013), otherwise `stats::hclust` is used.
- `agnes`: Hierarchical clustering using agglomerative nesting (Kaufman and Rousseeuw, 1990). This algorithm is similar to `hclust`, but uses the `cluster::agnes` implementation.
- `diana`: Divisive analysis hierarchical clustering. This method uses divisive instead of agglomerative clustering (Kaufman and Rousseeuw, 1990). `cluster::diana` is used.
- `pam`: Partioning around medioids. This partitions the data into $k$ clusters around medioids (Kaufman and Rousseeuw, 1990). $k$ is selected using the `silhouette` metric. `pam` is implemented using the `cluster::pam` function.

Clusters and cluster information is stored within `featureInfo` objects for later use with validation data sets. This enables reproduction of the same clusters as formed in the development data set.

`cluster_linkage_method` (*optional*) Linkage method used for agglomerative clustering in `hclust` and `agnes`. The following linkage methods can be used:

- `average` (default): Average linkage.
- `single`: Single linkage.
- `complete`: Complete linkage.
- `weighted`: Weighted linkage, also known as McQuitty linkage.
- `ward`: Linkage using Ward's minimum variance method.

`diana` and `pam` do not require a linkage method.

`cluster_cut_method` (*optional*) The method used to define the actual clusters. The following methods can be used:

- `silhouette`: Clusters are formed based on the silhouette score (Rousseeuw, 1987). The average silhouette score is computed from 2 to $n$ clusters, with $n$ the number of features. Clusters are only formed if the average silhouette exceeds 0.50, which indicates reasonable evidence for structure. This procedure may be slow if the number of features is large (>100s).
- `fixed_cut`: Clusters are formed by cutting the hierarchical tree at the point indicated by the `cluster_similarity_threshold`, e.g. where features in a cluster have an average Spearman correlation of 0.90. `fixed_cut` is only available for `agnes`, `diana` and `hclust`.
- `dynamic_cut`: Dynamic cluster formation using the cutting algorithm in the `dynamicTreeCut` package. This package should be installed to select this option. `dynamic_cut` can only be used with `agnes` and `hclust`.

The default options are `silhouette` for partioning around medioids (`pam`) and `fixed_cut` otherwise.

`cluster_similarity_metric` (*optional*) Clusters are formed based on feature similarity. All features are compared in a pair-wise fashion to compute similarity, for example correlation. The resulting similarity grid is converted into a distance matrix that is subsequently used for clustering. The following metrics are supported to compute pairwise similarities:

- `mutual_information` (default): normalised mutual information.
- `mcfadden_r2`: McFadden's pseudo R-squared (McFadden, 1974).
- `cox_snell_r2`: Cox and Snell's pseudo R-squared (Cox and Snell, 1989).
- `nagelkerke_r2`: Nagelkerke's pseudo R-squared (Nagelkerke, 1991).
- `spearman`: Spearman's rank order correlation.
- `kendall`: Kendall rank correlation.
- `pearson`: Pearson product-moment correlation.

The pseudo R-squared metrics can be used to assess similarity between mixed pairs of numeric and categorical features, as these are based on the log-likelihood of regression models. In `familiar`, the more informative feature is used as the predictor and the other feature as the reponse variable. In numeric-categorical pairs, the numeric feature is considered to be more informative and is thus used as the predictor. In categorical-categorical pairs, the feature with most levels is used as the predictor.

In case any of the classical correlation coefficients (`pearson`, `spearman` and `kendall`) are used with (mixed) categorical features, the categorical features are one-hot encoded and the mean correlation over all resulting pairs is used as similarity.

`cluster_similarity_threshold` (*optional*) The threshold level for pair-wise similarity that is required to form clusters using `fixed_cut`. This should be a numerical value between 0.0 and 1.0. Note however, that a reasonable threshold value depends strongly on the similarity metric. The following are the default values used:

- mcfadden_r2 and mutual_information: 0.30
- cox_snell_r2 and nagelkerke_r2: 0.75
- spearman, kendall and pearson: 0.90

Alternatively, if the fixed cut method is not used, this value determines whether any clustering should be performed, because the data may not contain highly similar features. The default values in this situation are:

- mcfadden_r2 and mutual_information: 0.25
- cox_snell_r2 and nagelkerke_r2: 0.40
- spearman, kendall and pearson: 0.70

The threshold value is converted to a distance (1-similarity) prior to cutting hierarchical trees.

cluster_representation_method (*optional*) Method used to determine how the information of co-clustered features is summarised and used to represent the cluster. The following methods can be selected:

- best_predictor (default): The feature with the highest importance according to univariate regression with the outcome is used to represent the cluster.
- medioid: The feature closest to the cluster center, i.e. the feature that is most similar to the remaining features in the cluster, is used to represent the feature.
- mean: A meta-feature is generated by averaging the feature values for all features in a cluster. This method aligns all features so that all features will be positively correlated prior to averaging. Should a cluster contain one or more categorical features, the medioid method will be used instead, as averaging is not possible. Note that if this method is chosen, the normalisation_method parameter should be one of standardisation, standardisation_trim, standardisation_winsor or quantile.‘

If the pam cluster method is selected, only the medioid method can be used. In that case 1 medioid is used by default.

parallel_preprocessing (*optional*) Enable parallel processing for the preprocessing workflow. Defaults to TRUE. When set to FALSE, this will disable the use of parallel processing while preprocessing, regardless of the settings of the parallel parameter. parallel_preprocessing is ignored if parallel=FALSE.

## Details

This is a thin wrapper around summon_familiar, and functions like it, but automatically skips computation of variable importance, learning and subsequent evaluation steps.

The function returns an experimentData object, which can be used to warm-start other experiments by providing it to the experiment_data argument.

## Value

An experimentData object.

precompute_feature_info *Pre-compute feature information*

## Description

Creates data assignment and subsequently extracts feature information such as normalisation and clustering parameters.

## Usage

```
precompute_feature_info(
  formula = NULL,
  data = NULL,
  experiment_data = NULL,
  cl = NULL,
  experimental_design = "fs+mb",
  verbose = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| formula | An R formula. The formula can only contain feature names and dot (.). The * and +1 operators are not supported as these refer to columns that are not present in the data set.<br>Use of the formula interface is optional. |
| data | A data.table object, a data.frame object, list containing multiple data.table or data.frame objects, or paths to data files.<br>data should be provided if no file paths are provided to the data_files argument. If both are provided, only data will be used.<br>All data is expected to be in wide format, and ideally has a sample identifier (see sample_id_column), batch identifier (see cohort_column) and outcome columns (see outcome_column).<br>In case paths are provided, the data should be stored as csv, rds or RData files. See documentation for the data_files argument for more information. |
| experiment_data | Experimental data may provided in the form of the output of precompute_data_assignment, precompute_feature_info or precompute_vimp. This allows for warm-starting experiments. |
| cl | Cluster created using the parallel package. This cluster is then used to speed up computation through parallelisation. When a cluster is not provided, parallelisation is performed by setting up a cluster on the local machine.<br>This parameter has no effect if the parallel argument is set to FALSE. |

`experimental_design`

(**required**) Defines what the experiment looks like, e.g. `cv(bt(fs,20)+mb,3,2)` for 2 times repeated 3-fold cross-validation with nested variable importance computation on 20 bootstraps and model-building. The basic workflow components are:

- `fs`: (optional) variable importance computation step. If not explicitly declared, feature selection will be done just in time for hyperparameter optimisation.
- `mb`: (required) model building step.
- `ev`: (optional) external validation. If validation batches or cohorts are present in the dataset (`data`), these should be indicated in the `validation_batch_id` argument.

The different components are linked using `+`.

Different subsampling methods can be used in conjunction with the basic workflow components:

- `bs(x,n)`: (stratified) .632 bootstrap, with `n` the number of bootstraps. In contrast to `bt`, feature pre-processing parameters and hyperparameter optimisation are conducted on individual bootstraps.
- `bt(x,n)`: (stratified) .632 bootstrap, with `n` the number of bootstraps. Unlike `bs` and other subsampling methods, no separate pre-processing parameters or optimised hyperparameters will be determined for each bootstrap.
- `cv(x,n,p)`: (stratified) `n`-fold cross-validation, repeated `p` times. Pre-processing parameters are determined for each iteration.
- `lv(x)`: leave-one-out-cross-validation. Pre-processing parameters are determined for each iteration.
- `ip(x)`: imbalance partitioning for addressing class imbalances on the data set. Pre-processing parameters are determined for each partition. The number of partitions generated depends on the imbalance correction method (see the `imbalance_correction_method` parameter).

As shown in the example above, sampling algorithms can be nested.

Though neither variable importance is determined nor models are learned within `precompute_feature_info`, the corresponding elements are still required to prevent issues when using the resulting `experimentData` object to warm-start the experiments.

The simplest valid experimental design is `fs+mb`. This is the default in `precompute_feature_info`, and will determine feature parameters over the entire dataset.

This argument is ignored if the `experiment_data` argument is set.

`verbose` Indicates verbosity of the results. Default is TRUE, and all messages and warnings are returned.

`...` Arguments passed on to `.parse_experiment_settings`, `.parse_setup_settings`, `.parse_preprocessing_settings`

`batch_id_column` (**recommended**) Name of the column containing batch or cohort identifiers. This parameter is required if more than one dataset is provided, or if external validation is performed.
In familiar any row of data is organised by four identifiers:

- The batch identifier batch_id_column: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.
- The sample identifier sample_id_column: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.
- The series identifier series_id_column: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.
- The repetition identifier: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries for the same series of the same sample in the same batch that share the same outcome are encountered.

sample_id_column (**recommended**) Name of the column containing sample or subject identifiers. See batch_id_column above for more details.
If unset, every row will be identified as a single sample.

series_id_column (*optional*) Name of the column containing series identifiers, which distinguish between measurements that are part of a series for a single sample. See batch_id_column above for more details.
If unset, rows which share the same batch and sample identifiers but have a different outcome are assigned unique series identifiers.

development_batch_id (*optional*) One or more batch or cohort identifiers to constitute data sets for development. Defaults to all, or all minus the identifiers in validation_batch_id for external validation. Required if external validation is performed and validation_batch_id is not provided.

validation_batch_id (*optional*) One or more batch or cohort identifiers to constitute data sets for external validation. Defaults to all data sets except those in development_batch_id for external validation, or none if not. Required if development_batch_id is not provided.

outcome_name (*optional*) Name of the modelled outcome. This name will be used in figures created by familiar.
If not set, the column name in outcome_column will be used for binomial, multinomial, and continuous outcomes. For other outcomes (survival and competing_risk) no default is used.

outcome_column (**recommended**) Name of the column containing the outcome of interest. May be identified from a formula, if a formula is provided as an argument. Otherwise an error is raised. Note that survival and competing_risk outcome type outcomes require two columns that indicate the time-to-event or the time of last follow-up and the event status.

outcome_type (**recommended**) Type of outcome found in the outcome column. The outcome type determines many aspects of the overall process, e.g. the available variable importance methods and learners, but also the type of assessments that can be conducted to evaluate the resulting models. Implemented outcome types are:

- binomial: categorical outcome with 2 levels.

- multinomial: categorical outcome with 2 or more levels.
- continuous: general continuous numeric outcomes.
- survival: survival outcome for time-to-event data.

If not provided, the algorithm will attempt to obtain outcome_type from contents of the outcome column. This may lead to unexpected results, and we therefore advise to provide this information manually.
Note that competing_risk survival analysis are not fully supported, and is currently not a valid choice for outcome_type. The count outcome type was deprecated in version 2.0.0, and superseded by continuous.

class_levels (*optional*) Class levels for binomial or multinomial outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column.

event_indicator (**recommended**) Indicator for events in survival and competing_risk analyses. familiar will automatically recognise 1, true, t, y and yes as event indicators, including different capitalisations. If this parameter is set, it replaces the default values.

censoring_indicator (**recommended**) Indicator for right-censoring in survival and competing_risk analyses. familiar will automatically recognise 0, false, f, n, no as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values.

competing_risk_indicator (**recommended**) Indicator for competing risks in competing_risk analyses. There are no default values, and if unset, all values other than those specified by the event_indicator and censoring_indicator parameters are considered to indicate competing risks.

signature (*optional*) One or more names of feature columns that are considered part of a specific signature. Features specified here will always be used for modelling. Ranking of variable importances has no effect for these features.

novelty_features (*optional*) One or more names of feature columns that should be included for the purpose of novelty detection.

exclude_features (*optional*) Feature columns that will be removed from the data set. Cannot overlap with features in signature, novelty_features or include_features.

include_features (*optional*) Feature columns that are specifically included in the data set. By default all features are included. Cannot overlap with exclude_features, but may overlap signature. Features in signature and novelty_features are always included. If both exclude_features and include_features are provided, include_features takes precedence, provided that there is no overlap between the two.

reference_method (*optional*) Method used to set reference levels for categorical features. There are several options:

- auto (default): Categorical features that are not explicitly set by the user, i.e. columns containing boolean values or characters, use the most frequent level as reference. Categorical features that are explicitly set, i.e. as factors, are used as is.

- always: Both automatically detected and user-specified categorical features have the reference level set to the most frequent level. Ordinal features are not altered, but are used as is.
- never: User-specified categorical features are used as is. Automatically detected categorical features are simply sorted, and the first level is then used as the reference level. This was the behaviour prior to familiar version 1.3.0.

imbalance_correction_method (*optional*) Type of method used to address class imbalances. Available options are:

- full_undersampling (default): All data will be used in an ensemble fashion. The full minority class will appear in each partition, but majority classes are undersampled until all data have been used.
- random_undersampling: Randomly undersamples majority classes. This is useful in cases where full undersampling would lead to the formation of many models due major overrepresentation of the largest class.

This parameter is only used in combination with imbalance partitioning in the experimental design, and ip should therefore appear in the string that defines the design.

imbalance_n_partitions (*optional*) Number of times random undersampling should be repeated. 10 undersampled subsets with balanced classes are formed by default.

iteration_seed (*optional*) Integer seed used in sampling algorithms specified by the experimental_design argument. This allows for creating the same sample assignments across different experiments – of course provided that the same dataset is used. By default a random seed is used.

parallel (*optional*) Enable parallel processing. Defaults to TRUE. When set to FALSE, this disables all parallel processing, regardless of specific parameters such as parallel_preprocessing. However, when parallel is TRUE, parallel processing of different parts of the workflow can be disabled by setting respective flags to FALSE.

parallel_nr_cores (*optional*) Number of cores available for parallelisation. Defaults to 2. This setting does nothing if parallelisation is disabled.

restart_cluster (*optional*) Restart nodes used for parallel computing to free up memory prior to starting a parallel process. Note that it does take time to set up the clusters. Therefore setting this argument to TRUE may impact processing speed. This argument is ignored if parallel is FALSE or the cluster was initialised outside of familiar. Default is FALSE, which causes the clusters to be initialised only once.

cluster_type (*optional*) Selection of the cluster type for parallel processing. Available types are the ones supported by the parallel package that is part of the base R distribution: psock (default), fork, mpi, nws, sock. In addition, none is available, which also disables parallel processing.

backend_type (*optional*) Selection of the backend for distributing copies of the data. This backend ensures that only a single master copy is kept in memory. This limits memory usage during parallel processing.

Several backend options are available, notably socket_server, and none (default). socket_server is based on the callr package and R sockets, comes with familiar and is available for any OS. none uses the package environment of familiar to store data, and is available for any OS. However, none requires copying of data to any parallel process, and has a larger memory footprint.

server_port (*optional*) Integer indicating the port on which the socket server or RServe process should communicate. Defaults to port 6311. Note that ports 0 to 1024 and 49152 to 65535 cannot be used.

feature_max_fraction_missing (*optional*) Numeric value between 0.0 and 0.95 that determines the meximum fraction of missing values that still allows a feature to be included in the data set. All features with a missing value fraction over this threshold are not processed further. The default value is 0.30.

sample_max_fraction_missing (*optional*) Numeric value between 0.0 and 0.95 that determines the maximum fraction of missing values that still allows a sample to be included in the data set. All samples with a missing value fraction over this threshold are excluded and not processed further. The default value is 0.30.

filter_method (*optional*) One or methods used to reduce dimensionality of the data set by removing irrelevant or poorly reproducible features.
Several method are available:

- none (default): None of the features will be filtered.
- low_variance: Features with a variance below the low_var_minimum_variance_threshold are filtered. This can be useful to filter, for example, genes that are not differentially expressed.
- univariate_test: Features undergo a univariate regression using an outcome-appropriate regression model. The p-value of the model coefficient is collected. Features with coefficient p-value above the univariate_test_threshold are subsequently filtered.
- robustness: Features that are not sufficiently robust according to the intraclass correlation coefficient are filtered. Use of this method requires that repeated measurements are present in the data set, i.e. there should be entries for which the sample and cohort identifiers are the same.

More than one method can be used simultaneously. Features with singular values are always filtered, as these do not contain information.

univariate_test_threshold (*optional*) Numeric value between 1.0 and 0.0 that determines which features are irrelevant and will be filtered by the univariate_test. p-values are compared to this threshold. All features with values above the threshold are filtered. The default value is 0.20.

univariate_test_threshold_metric (*optional*) Metric used with the to compare the univariate_test_threshold against. The following metric can be chosen:

- p_value (default): The unadjusted p-value of each feature is used for to filter features.

`univariate_test_max_feature_set_size` (*optional*) Maximum size of the feature set after the univariate test. P or q values of features are compared against the threshold, but if the resulting data set would be larger than this setting, only the most relevant features up to the desired feature set size are selected.
The default value is `NULL`, which causes features to be filtered based on their relevance only.

`low_var_minimum_variance_threshold` (required, if used) Numeric value that determines which features will be filtered by the `low_variance` method. The variance of each feature is computed and compared to the threshold. If it is below the threshold, the feature is removed.
This parameter has no default value and should be set if `low_variance` is used.

`low_var_max_feature_set_size` (*optional*) Maximum size of the feature set after filtering features with a low variance. All features are first compared against `low_var_minimum_variance_threshold`. If the resulting feature set would be larger than specified, only the most strongly varying features will be selected, up to the desired size of the feature set.
The default value is `NULL`, which causes features to be filtered based on their variance only.

`robustness_icc_type` (*optional*) String indicating the type of intraclass correlation coefficient (`1`, `2` or `3`) that should be used to compute robustness for features in repeated measurements. These types correspond to the types in Shrout and Fleiss (1979). The default value is `1`.

`robustness_threshold_metric` (*optional*) String indicating which specific intraclass correlation coefficient (ICC) metric should be used to filter features. This should be one of:

- `icc`: The estimated ICC value itself.
- `icc_low` (default): The estimated lower limit of the 95% confidence interval of the ICC, as suggested by Koo and Li (2016).
- `icc_panel`: The estimated ICC value over the panel average, i.e. the ICC that would be obtained if all repeated measurements were averaged.
- `icc_panel_low`: The estimated lower limit of the 95% confidence interval of the panel ICC.

`robustness_threshold_value` (*optional*) The intraclass correlation coefficient value that is as threshold. The default value is `0.70`.

`transformation_method` (*optional*) The transformation method used to change the distribution of the data to be more normal-like. The following methods are available:

- `none`: This disables transformation of features.
- `yeo_johnson`: Transformation using the location and scale invariant version of the Yeo-Johnson transformation (Yeo and Johnson, 2000; Zwanenburg and Löck, 2026).
- `yeo_johnson_robust` (default): A robust version of `yeo_johnson`. This method is less sensitive to outliers.

- `yeo_johnson_conventional`: As `yeo_johnson`, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Yeo and Johnson (2001).
- `box_cox`: Transformation using the location and scale invariant version of the Box-Cox transformation (Box and Cox, 1964; Zwanenburg and Löck, 2026).
- `box_cox_robust`: A robust version of `yeo_johnson`. This method is less sensitive to outliers.
- `box_cox_conventional`: As `box_cox`, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Box and Cox (1964). This method requires strictly positive feature values.

Transformation requires the `power.transform` package. Only features that contain numerical data are transformed. Transformation parameters obtained in development data are stored within `featureInfo` objects for later use with validation data sets.

`transformation_optimisation_criterion` (*optional*) Transformation parameters are optimised using a criterion, conventionally maximum-likelihood-estimation. `power.transform` implements multiple optimisation criteria, of which the following are available:

- `mle` (default): Optimisation using maximum likelihood estimation.
- `cramer_von_mises`: Optimisation using the Cramér-von Mises criterion. Zwanenburg and Löck (2026) found that this criterion was relatively robust against outliers.

`transformation_gof_test_p_value` (*optional*) Not all transformations will lead to features that are roughly normally distributed. Zwanenburg and Löck (2026) established a empirical goodness-of-fit test for central normality. This parameter sets the significance for rejecting the null-hypothesis that a feature distribution is centrally normal. When the null-hypothesis is rejected, no transformation is performed. The default value is `NULL`, which disables the test.

`normalisation_method` (*optional*) The normalisation method used to improve the comparability between numerical features that may have very different scales. The following normalisation methods can be chosen:

- `none`: This disables feature normalisation.
- `standardisation`: Features are normalised by subtraction of their mean values and division by their standard deviations. This causes every feature to be have a center value of 0.0 and standard deviation of 1.0.
- `standardisation_trim`: As `standardisation`, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- `standardisation_winsor`: As `standardisation`, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- `standardisation_robust` (default): A robust version of `standardisation` that relies on computing Huber's M-estimators for location and scale.

- `normalisation`: Features are normalised by subtraction of their minimum values and division by their ranges. This maps all feature values to a $[0, 1]$ interval.
- `normalisation_trim`: As `normalisation`, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- `normalisation_winsor`: As `normalisation`, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- `quantile`: Features are normalised by subtraction of their median values and division by their interquartile range.
- `mean_centering`: Features are centered by substracting the mean, but do not undergo rescaling.

Only features that contain numerical data are normalised. Normalisation parameters obtained in development data are stored within `featureInfo` objects for later use with validation data sets.

`batch_normalisation_method` (*optional*) The method used for batch normalisation. Available methods are:

- `none` (default): This disables batch normalisation of features.
- `standardisation`: Features within each batch are normalised by subtraction of the mean value and division by the standard deviation in each batch.
- `standardisation_trim`: As `standardisation`, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- `standardisation_winsor`: As `standardisation`, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- `standardisation_robust`: A robust version of `standardisation` that relies on computing Huber's M-estimators for location and scale within each batch.
- `normalisation`: Features within each batch are normalised by subtraction of their minimum values and division by their range in each batch. This maps all feature values in each batch to a $[0, 1]$ interval.
- `normalisation_trim`: As `normalisation`, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- `normalisation_winsor`: As `normalisation`, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- `quantile`: Features in each batch are normalised by subtraction of the median value and division by the interquartile range of each batch.
- `mean_centering`: Features in each batch are centered on 0.0 by substracting the mean value in each batch, but are not rescaled.
- `combat_parametric`: Batch adjustments using parametric empirical Bayes (Johnson et al, 2007). `combat_p` leads to the same method.

- `combat_non_parametric`: Batch adjustments using non-parametric empirical Bayes (Johnson et al, 2007). `combat_np` and `combat` lead to the same method. Note that we reduced complexity from O($n^2$) to O($n$) by only computing batch adjustment parameters for each feature on a subset of 50 randomly selected features, instead of all features.

Only features that contain numerical data are normalised using batch normalisation. Batch normalisation parameters obtained in development data are stored within `featureInfo` objects for later use with validation data sets, in case the validation data is from the same batch.
If validation data contains data from unknown batches, normalisation parameters are separately determined for these batches.
Note that for both empirical Bayes methods, the batch effect is assumed to produce results across the features. This is often true for things such as gene expressions, but the assumption may not hold generally.
When performing batch normalisation, it is moreover important to check that differences between batches or cohorts are not related to the studied endpoint.

`imputation_method` (*optional*) Method used for imputing missing feature values. Two methods are implemented:

- `simple`: Simple replacement of a missing value by the median value (for numeric features) or the modal value (for categorical features).
- `lasso`: Imputation of missing value by lasso regression (using `glmnet`) based on information contained in other features.

`simple` imputation precedes `lasso` imputation to ensure that any missing values in predictors required for `lasso` regression are resolved. The `lasso` estimate is then used to replace the missing value.
The default value depends on the number of features in the dataset. If the number is lower than 100, `lasso` is used by default, and `simple` otherwise.
Only single imputation is performed. Imputation models and parameters are stored within `featureInfo` objects for later use with validation data sets.

`cluster_method` (*optional*) Clustering is performed to identify and replace redundant features, for example those that are highly correlated. Such features do not carry much additional information and may be removed or replaced instead (Park et al., 2007; Tolosi and Lengauer, 2011).
The cluster method determines the algorithm used to form the clusters. The following cluster methods are implemented:

- `none`: No clustering is performed.
- `hclust` (default): Hierarchical agglomerative clustering. If the `fastcluster` package is installed, `fastcluster::hclust` is used (Muellner 2013), otherwise `stats::hclust` is used.
- `agnes`: Hierarchical clustering using agglomerative nesting (Kaufman and Rousseeuw, 1990). This algorithm is similar to `hclust`, but uses the `cluster::agnes` implementation.
- `diana`: Divisive analysis hierarchical clustering. This method uses divisive instead of agglomerative clustering (Kaufman and Rousseeuw, 1990). `cluster::diana` is used.

- pam: Partioning around medioids. This partitions the data into $k$ clusters around medioids (Kaufman and Rousseeuw, 1990). $k$ is selected using the silhouette metric. pam is implemented using the cluster::pam function.

Clusters and cluster information is stored within featureInfo objects for later use with validation data sets. This enables reproduction of the same clusters as formed in the development data set.

cluster_linkage_method (*optional*) Linkage method used for agglomerative clustering in hclust and agnes. The following linkage methods can be used:

- average (default): Average linkage.
- single: Single linkage.
- complete: Complete linkage.
- weighted: Weighted linkage, also known as McQuitty linkage.
- ward: Linkage using Ward's minimum variance method.

diana and pam do not require a linkage method.

cluster_cut_method (*optional*) The method used to define the actual clusters. The following methods can be used:

- silhouette: Clusters are formed based on the silhouette score (Rousseeuw, 1987). The average silhouette score is computed from 2 to $n$ clusters, with $n$ the number of features. Clusters are only formed if the average silhouette exceeds 0.50, which indicates reasonable evidence for structure. This procedure may be slow if the number of features is large (>100s).
- fixed_cut: Clusters are formed by cutting the hierarchical tree at the point indicated by the cluster_similarity_threshold, e.g. where features in a cluster have an average Spearman correlation of 0.90. fixed_cut is only available for agnes, diana and hclust.
- dynamic_cut: Dynamic cluster formation using the cutting algorithm in the dynamicTreeCut package. This package should be installed to select this option. dynamic_cut can only be used with agnes and hclust.

The default options are silhouette for partioning around medioids (pam) and fixed_cut otherwise.

cluster_similarity_metric (*optional*) Clusters are formed based on feature similarity. All features are compared in a pair-wise fashion to compute similarity, for example correlation. The resulting similarity grid is converted into a distance matrix that is subsequently used for clustering. The following metrics are supported to compute pairwise similarities:

- mutual_information (default): normalised mutual information.
- mcfadden_r2: McFadden's pseudo R-squared (McFadden, 1974).
- cox_snell_r2: Cox and Snell's pseudo R-squared (Cox and Snell, 1989).
- nagelkerke_r2: Nagelkerke's pseudo R-squared (Nagelkerke, 1991).
- spearman: Spearman's rank order correlation.
- kendall: Kendall rank correlation.

- pearson: Pearson product-moment correlation.

The pseudo R-squared metrics can be used to assess similarity between mixed pairs of numeric and categorical features, as these are based on the log-likelihood of regression models. In familiar, the more informative feature is used as the predictor and the other feature as the reponse variable. In numeric-categorical pairs, the numeric feature is considered to be more informative and is thus used as the predictor. In categorical-categorical pairs, the feature with most levels is used as the predictor.

In case any of the classical correlation coefficients (pearson, spearman and kendall) are used with (mixed) categorical features, the categorical features are one-hot encoded and the mean correlation over all resulting pairs is used as similarity.

cluster_similarity_threshold (*optional*) The threshold level for pair-wise similarity that is required to form clusters using fixed_cut. This should be a numerical value between 0.0 and 1.0. Note however, that a reasonable threshold value depends strongly on the similarity metric. The following are the default values used:

- mcfadden_r2 and mutual_information: 0.30
- cox_snell_r2 and nagelkerke_r2: 0.75
- spearman, kendall and pearson: 0.90

Alternatively, if the fixed cut method is not used, this value determines whether any clustering should be performed, because the data may not contain highly similar features. The default values in this situation are:

- mcfadden_r2 and mutual_information: 0.25
- cox_snell_r2 and nagelkerke_r2: 0.40
- spearman, kendall and pearson: 0.70

The threshold value is converted to a distance (1-similarity) prior to cutting hierarchical trees.

cluster_representation_method (*optional*) Method used to determine how the information of co-clustered features is summarised and used to represent the cluster. The following methods can be selected:

- best_predictor (default): The feature with the highest importance according to univariate regression with the outcome is used to represent the cluster.
- medioid: The feature closest to the cluster center, i.e. the feature that is most similar to the remaining features in the cluster, is used to represent the feature.
- mean: A meta-feature is generated by averaging the feature values for all features in a cluster. This method aligns all features so that all features will be positively correlated prior to averaging. Should a cluster contain one or more categorical features, the medioid method will be used instead, as averaging is not possible. Note that if this method is chosen, the normalisation_method parameter should be one of standardisation, standardisation_trim, standardisation_winsor or quantile.‘

If the pam cluster method is selected, only the medioid method can be used. In that case 1 medioid is used by default.

parallel_preprocessing (*optional*) Enable parallel processing for the preprocessing workflow. Defaults to TRUE. When set to FALSE, this will disable the use of parallel processing while preprocessing, regardless of the settings of the parallel parameter. parallel_preprocessing is ignored if parallel=FALSE.

## Details

This is a thin wrapper around summon_familiar, and functions like it, but automatically skips computation of variable importance, learning and subsequent evaluation steps.

The function returns an experimentData object, which can be used to warm-start other experiments by providing it to the experiment_data argument.

## Value

An experimentData object.

---

precompute_vimp *Pre-compute variable importance*

---

## Description

Creates data assignment, extracts feature information and subsequently computes variable importance.

## Usage

```
precompute_vimp(
  formula = NULL,
  data = NULL,
  experiment_data = NULL,
  cl = NULL,
  experimental_design = "fs+mb",
  vimp_method = NULL,
  vimp_method_parameter = NULL,
  verbose = TRUE,
  ...
)
```

## Arguments

formula An R formula. The formula can only contain feature names and dot (.). The * and +1 operators are not supported as these refer to columns that are not present in the data set.

Use of the formula interface is optional.

data
: A data.table object, a data.frame object, list containing multiple data.table or data.frame objects, or paths to data files.

  data should be provided if no file paths are provided to the data_files argument. If both are provided, only data will be used.

  All data is expected to be in wide format, and ideally has a sample identifier (see sample_id_column), batch identifier (see cohort_column) and outcome columns (see outcome_column).

  In case paths are provided, the data should be stored as csv, rds or RData files. See documentation for the data_files argument for more information.

experiment_data
: Experimental data may provided in the form of the output of precompute_data_assignment, precompute_feature_info or precompute_vimp. This allows for warm-starting experiments.

cl
: Cluster created using the parallel package. This cluster is then used to speed up computation through parallelisation. When a cluster is not provided, parallelisation is performed by setting up a cluster on the local machine.

  This parameter has no effect if the parallel argument is set to FALSE.

experimental_design
: (**required**) Defines what the experiment looks like, e.g. cv(bt(fs,20)+mb,3,2) for 2 times repeated 3-fold cross-validation with nested variable importance computation on 20 bootstraps and model-building. The basic workflow components are:

  - fs: (required) variable importance computation step. No variable importances will be prepared if this step is not explicitly used, instead, feature selection will be done just in time for hyperparameter optimisation.
  - mb: (required) model building step. Though models are not learned by precompute_vimp, this element is still required to prevent issues when using the resulting experimentData object to warm-start the experiments.
  - ev: (optional) external validation. If validation batches or cohorts are present in the dataset (data), these should be indicated in the validation_batch_id argument.

  The different components are linked using +.

  Different subsampling methods can be used in conjunction with the basic workflow components:

  - bs(x,n): (stratified) .632 bootstrap, with n the number of bootstraps. In contrast to bt, feature pre-processing parameters and hyperparameter optimisation are conducted on individual bootstraps.
  - bt(x,n): (stratified) .632 bootstrap, with n the number of bootstraps. Unlike bs and other subsampling methods, no separate pre-processing parameters or optimised hyperparameters will be determined for each bootstrap.
  - cv(x,n,p): (stratified) n-fold cross-validation, repeated p times. Pre-processing parameters are determined for each iteration.
  - lv(x): leave-one-out-cross-validation. Pre-processing parameters are determined for each iteration.

- `ip(x)`: imbalance partitioning for addressing class imbalances on the data set. Pre-processing parameters are determined for each partition. The number of partitions generated depends on the imbalance correction method (see the `imbalance_correction_method` parameter).

As shown in the example above, sampling algorithms can be nested.

The simplest valid experimental design is `fs+mb`. This is the default in `precompute_vimp`, and will compute variable importance over the entire dataset.

This argument is ignored if the `experiment_data` argument is set.

`vimp_method` (**required**) Variable importance method. `familiar` implements various variable importance methods. Please refer to the vignette on variable importance methods for more details.

More than one variable importance method can be chosen. The experiment will then repeated for each feature selection method.

Variable importance methods determine the ranking of features. Actual selection of features is done by optimising the signature size model hyperparameter during the hyperparameter optimisation step.

`vimp_method_parameter`

(*optional*) List of lists containing parameters for feature selection methods. Each sublist should have the name of the feature selection method it corresponds to.

Most feature selection methods do not have parameters that can be set. Please refer to the vignette on feature selection methods for more details. Note that if the feature selection method is based on a learner (e.g. lasso regression), hyperparameter optimisation may be performed prior to assessing variable importance.

`verbose` Indicates verbosity of the results. Default is TRUE, and all messages and warnings are returned.

`...` Arguments passed on to `.parse_experiment_settings`, `.parse_setup_settings`, `.parse_preprocessing_settings`, `.parse_variable_importance_settings`

`batch_id_column` (**recommended**) Name of the column containing batch or cohort identifiers. This parameter is required if more than one dataset is provided, or if external validation is performed.

In familiar any row of data is organised by four identifiers:

- The batch identifier `batch_id_column`: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.
- The sample identifier `sample_id_column`: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.
- The series identifier `series_id_column`: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.
- The repetition identifier: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries for the same series of the same sample in the same batch that share the same outcome are encountered.

sample_id_column (**recommended**) Name of the column containing sample or subject identifiers. See batch_id_column above for more details.
If unset, every row will be identified as a single sample.

series_id_column (*optional*) Name of the column containing series identifiers, which distinguish between measurements that are part of a series for a single sample. See batch_id_column above for more details.
If unset, rows which share the same batch and sample identifiers but have a different outcome are assigned unique series identifiers.

development_batch_id (*optional*) One or more batch or cohort identifiers to constitute data sets for development. Defaults to all, or all minus the identifiers in validation_batch_id for external validation. Required if external validation is performed and validation_batch_id is not provided.

validation_batch_id (*optional*) One or more batch or cohort identifiers to constitute data sets for external validation. Defaults to all data sets except those in development_batch_id for external validation, or none if not. Required if development_batch_id is not provided.

outcome_name (*optional*) Name of the modelled outcome. This name will be used in figures created by familiar.
If not set, the column name in outcome_column will be used for binomial, multinomial, and continuous outcomes. For other outcomes (survival and competing_risk) no default is used.

outcome_column (**recommended**) Name of the column containing the outcome of interest. May be identified from a formula, if a formula is provided as an argument. Otherwise an error is raised. Note that survival and competing_risk outcome type outcomes require two columns that indicate the time-to-event or the time of last follow-up and the event status.

outcome_type (**recommended**) Type of outcome found in the outcome column. The outcome type determines many aspects of the overall process, e.g. the available variable importance methods and learners, but also the type of assessments that can be conducted to evaluate the resulting models.
Implemented outcome types are:

- binomial: categorical outcome with 2 levels.
- multinomial: categorical outcome with 2 or more levels.
- continuous: general continuous numeric outcomes.
- survival: survival outcome for time-to-event data.

If not provided, the algorithm will attempt to obtain outcome_type from contents of the outcome column. This may lead to unexpected results, and we therefore advise to provide this information manually.
Note that competing_risk survival analysis are not fully supported, and is currently not a valid choice for outcome_type. The count outcome type was deprecated in version 2.0.0, and superseded by continuous.

class_levels (*optional*) Class levels for binomial or multinomial outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column.

event_indicator (**recommended**) Indicator for events in survival and competing_risk analyses. familiar will automatically recognise 1, true, t, y and yes as

event indicators, including different capitalisations. If this parameter is set, it replaces the default values.

`censoring_indicator` (**recommended**) Indicator for right-censoring in `survival` and `competing_risk` analyses. `familiar` will automatically recognise `0`, `false`, `f`, `n`, `no` as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values.

`competing_risk_indicator` (**recommended**) Indicator for competing risks in `competing_risk` analyses. There are no default values, and if unset, all values other than those specified by the `event_indicator` and `censoring_indicator` parameters are considered to indicate competing risks.

`signature` (*optional*) One or more names of feature columns that are considered part of a specific signature. Features specified here will always be used for modelling. Ranking of variable importances has no effect for these features.

`novelty_features` (*optional*) One or more names of feature columns that should be included for the purpose of novelty detection.

`exclude_features` (*optional*) Feature columns that will be removed from the data set. Cannot overlap with features in `signature`, `novelty_features` or `include_features`.

`include_features` (*optional*) Feature columns that are specifically included in the data set. By default all features are included. Cannot overlap with `exclude_features`, but may overlap `signature`. Features in `signature` and `novelty_features` are always included. If both `exclude_features` and `include_features` are provided, `include_features` takes precedence, provided that there is no overlap between the two.

`reference_method` (*optional*) Method used to set reference levels for categorical features. There are several options:

- `auto` (default): Categorical features that are not explicitly set by the user, i.e. columns containing boolean values or characters, use the most frequent level as reference. Categorical features that are explicitly set, i.e. as factors, are used as is.
- `always`: Both automatically detected and user-specified categorical features have the reference level set to the most frequent level. Ordinal features are not altered, but are used as is.
- `never`: User-specified categorical features are used as is. Automatically detected categorical features are simply sorted, and the first level is then used as the reference level. This was the behaviour prior to familiar version 1.3.0.

`imbalance_correction_method` (*optional*) Type of method used to address class imbalances. Available options are:

- `full_undersampling` (default): All data will be used in an ensemble fashion. The full minority class will appear in each partition, but majority classes are undersampled until all data have been used.
- `random_undersampling`: Randomly undersamples majority classes. This is useful in cases where full undersampling would lead to the formation of many models due major overrepresentation of the largest class.

This parameter is only used in combination with imbalance partitioning in the experimental design, and `ip` should therefore appear in the string that defines the design.

`imbalance_n_partitions` (*optional*) Number of times random undersampling should be repeated. 10 undersampled subsets with balanced classes are formed by default.

`iteration_seed` (*optional*) Integer seed used in sampling algorithms specified by the `experimental_design` argument. This allows for creating the same sample assignments across different experiments – of course provided that the same dataset is used. By default a random seed is used.

`parallel` (*optional*) Enable parallel processing. Defaults to `TRUE`. When set to `FALSE`, this disables all parallel processing, regardless of specific parameters such as `parallel_preprocessing`. However, when `parallel` is `TRUE`, parallel processing of different parts of the workflow can be disabled by setting respective flags to `FALSE`.

`parallel_nr_cores` (*optional*) Number of cores available for parallelisation. Defaults to 2. This setting does nothing if parallelisation is disabled.

`restart_cluster` (*optional*) Restart nodes used for parallel computing to free up memory prior to starting a parallel process. Note that it does take time to set up the clusters. Therefore setting this argument to `TRUE` may impact processing speed. This argument is ignored if `parallel` is `FALSE` or the cluster was initialised outside of familiar. Default is `FALSE`, which causes the clusters to be initialised only once.

`cluster_type` (*optional*) Selection of the cluster type for parallel processing. Available types are the ones supported by the parallel package that is part of the base R distribution: `psock` (default), `fork`, `mpi`, `nws`, `sock`. In addition, `none` is available, which also disables parallel processing.

`backend_type` (*optional*) Selection of the backend for distributing copies of the data. This backend ensures that only a single master copy is kept in memory. This limits memory usage during parallel processing.
Several backend options are available, notably `socket_server`, and `none` (default). `socket_server` is based on the callr package and R sockets, comes with `familiar` and is available for any OS. `none` uses the package environment of familiar to store data, and is available for any OS. However, `none` requires copying of data to any parallel process, and has a larger memory footprint.

`server_port` (*optional*) Integer indicating the port on which the socket server or RServe process should communicate. Defaults to port 6311. Note that ports 0 to 1024 and 49152 to 65535 cannot be used.

`feature_max_fraction_missing` (*optional*) Numeric value between `0.0` and `0.95` that determines the meximum fraction of missing values that still allows a feature to be included in the data set. All features with a missing value fraction over this threshold are not processed further. The default value is `0.30`.

`sample_max_fraction_missing` (*optional*) Numeric value between `0.0` and `0.95` that determines the maximum fraction of missing values that still allows a sample to be included in the data set. All samples with a missing

value fraction over this threshold are excluded and not processed further. The default value is `0.30`.

`filter_method` (*optional*) One or methods used to reduce dimensionality of the data set by removing irrelevant or poorly reproducible features.
Several method are available:

- `none` (default): None of the features will be filtered.
- `low_variance`: Features with a variance below the `low_var_minimum_variance_threshold` are filtered. This can be useful to filter, for example, genes that are not differentially expressed.
- `univariate_test`: Features undergo a univariate regression using an outcome-appropriate regression model. The p-value of the model coefficient is collected. Features with coefficient p-value above the `univariate_test_threshold` are subsequently filtered.
- `robustness`: Features that are not sufficiently robust according to the intraclass correlation coefficient are filtered. Use of this method requires that repeated measurements are present in the data set, i.e. there should be entries for which the sample and cohort identifiers are the same.

More than one method can be used simultaneously. Features with singular values are always filtered, as these do not contain information.

`univariate_test_threshold` (*optional*) Numeric value between `1.0` and `0.0` that determines which features are irrelevant and will be filtered by the `univariate_test`. p-values are compared to this threshold. All features with values above the threshold are filtered. The default value is `0.20`.

`univariate_test_threshold_metric` (*optional*) Metric used with the to compare the `univariate_test_threshold` against. The following metric can be chosen:

- `p_value` (default): The unadjusted p-value of each feature is used for to filter features.

`univariate_test_max_feature_set_size` (*optional*) Maximum size of the feature set after the univariate test. P or q values of features are compared against the threshold, but if the resulting data set would be larger than this setting, only the most relevant features up to the desired feature set size are selected.
The default value is `NULL`, which causes features to be filtered based on their relevance only.

`low_var_minimum_variance_threshold` (required, if used) Numeric value that determines which features will be filtered by the `low_variance` method. The variance of each feature is computed and compared to the threshold. If it is below the threshold, the feature is removed.
This parameter has no default value and should be set if `low_variance` is used.

`low_var_max_feature_set_size` (*optional*) Maximum size of the feature set after filtering features with a low variance. All features are first compared against `low_var_minimum_variance_threshold`. If the resulting feature set would be larger than specified, only the most strongly varying features will be selected, up to the desired size of the feature set.

The default value is NULL, which causes features to be filtered based on their variance only.

robustness_icc_type (*optional*) String indicating the type of intraclass correlation coefficient (1, 2 or 3) that should be used to compute robustness for features in repeated measurements. These types correspond to the types in Shrout and Fleiss (1979). The default value is 1.

robustness_threshold_metric (*optional*) String indicating which specific intraclass correlation coefficient (ICC) metric should be used to filter features. This should be one of:

- icc: The estimated ICC value itself.
- icc_low (default): The estimated lower limit of the 95% confidence interval of the ICC, as suggested by Koo and Li (2016).
- icc_panel: The estimated ICC value over the panel average, i.e. the ICC that would be obtained if all repeated measurements were averaged.
- icc_panel_low: The estimated lower limit of the 95% confidence interval of the panel ICC.

robustness_threshold_value (*optional*) The intraclass correlation coefficient value that is as threshold. The default value is 0.70.

transformation_method (*optional*) The transformation method used to change the distribution of the data to be more normal-like. The following methods are available:

- none: This disables transformation of features.
- yeo_johnson: Transformation using the location and scale invariant version of the Yeo-Johnson transformation (Yeo and Johnson, 2000; Zwanenburg and Löck, 2026).
- yeo_johnson_robust (default): A robust version of yeo_johnson. This method is less sensitive to outliers.
- yeo_johnson_conventional: As yeo_johnson, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Yeo and Johnson (2001).
- box_cox: Transformation using the location and scale invariant version of the Box-Cox transformation (Box and Cox, 1964; Zwanenburg and Löck, 2026).
- box_cox_robust: A robust version of yeo_johnson. This method is less sensitive to outliers.
- box_cox_conventional: As box_cox, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Box and Cox (1964). This method requires strictly positive feature values.

Transformation requires the power.transform package. Only features that contain numerical data are transformed. Transformation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets.

transformation_optimisation_criterion (*optional*) Transformation parameters are optimised using a criterion, conventionally maximum-likelihood-

estimation. power.transform implements multiple optimisation criteria, of which the following are available:

- mle (default): Optimisation using maximum likelihood estimation.
- cramer_von_mises: Optimisation using the Cramér-von Mises criterion. Zwanenburg and Löck (2026) found that this criterion was relatively robust against outliers.

transformation_gof_test_p_value (*optional*) Not all transformations will lead to features that are roughly normally distributed. Zwanenburg and Löck (2026) established a empirical goodness-of-fit test for central normality. This parameter sets the significance for rejecting the null-hypothesis that a feature distribution is centrally normal. When the null-hypothesis is rejected, no transformation is performed. The default value is NULL, which disables the test.

normalisation_method (*optional*) The normalisation method used to improve the comparability between numerical features that may have very different scales. The following normalisation methods can be chosen:

- none: This disables feature normalisation.
- standardisation: Features are normalised by subtraction of their mean values and division by their standard deviations. This causes every feature to be have a center value of 0.0 and standard deviation of 1.0.
- standardisation_trim: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- standardisation_winsor: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- standardisation_robust (default): A robust version of standardisation that relies on computing Huber's M-estimators for location and scale.
- normalisation: Features are normalised by subtraction of their minimum values and division by their ranges. This maps all feature values to a $[0, 1]$ interval.
- normalisation_trim: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- normalisation_winsor: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- quantile: Features are normalised by subtraction of their median values and division by their interquartile range.
- mean_centering: Features are centered by substracting the mean, but do not undergo rescaling.

Only features that contain numerical data are normalised. Normalisation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets.

batch_normalisation_method (*optional*) The method used for batch normalisation. Available methods are:

- none (default): This disables batch normalisation of features.
- standardisation: Features within each batch are normalised by subtraction of the mean value and division by the standard deviation in each batch.
- standardisation_trim: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- standardisation_winsor: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- standardisation_robust: A robust version of standardisation that relies on computing Huber's M-estimators for location and scale within each batch.
- normalisation: Features within each batch are normalised by subtraction of their minimum values and division by their range in each batch. This maps all feature values in each batch to a $[0, 1]$ interval.
- normalisation_trim: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- normalisation_winsor: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- quantile: Features in each batch are normalised by subtraction of the median value and division by the interquartile range of each batch.
- mean_centering: Features in each batch are centered on 0.0 by substracting the mean value in each batch, but are not rescaled.
- combat_parametric: Batch adjustments using parametric empirical Bayes (Johnson et al, 2007). combat_p leads to the same method.
- combat_non_parametric: Batch adjustments using non-parametric empirical Bayes (Johnson et al, 2007). combat_np and combat lead to the same method. Note that we reduced complexity from O($n^2$) to O($n$) by only computing batch adjustment parameters for each feature on a subset of 50 randomly selected features, instead of all features.

Only features that contain numerical data are normalised using batch normalisation. Batch normalisation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets, in case the validation data is from the same batch.

If validation data contains data from unknown batches, normalisation parameters are separately determined for these batches.

Note that for both empirical Bayes methods, the batch effect is assumed to produce results across the features. This is often true for things such as gene expressions, but the assumption may not hold generally.

When performing batch normalisation, it is moreover important to check that differences between batches or cohorts are not related to the studied endpoint.

imputation_method (*optional*) Method used for imputing missing feature values. Two methods are implemented:

- simple: Simple replacement of a missing value by the median value (for numeric features) or the modal value (for categorical features).
- lasso: Imputation of missing value by lasso regression (using glmnet) based on information contained in other features.

simple imputation precedes lasso imputation to ensure that any missing values in predictors required for lasso regression are resolved. The lasso estimate is then used to replace the missing value.

The default value depends on the number of features in the dataset. If the number is lower than 100, lasso is used by default, and simple otherwise.

Only single imputation is performed. Imputation models and parameters are stored within featureInfo objects for later use with validation data sets.

cluster_method (*optional*) Clustering is performed to identify and replace redundant features, for example those that are highly correlated. Such features do not carry much additional information and may be removed or replaced instead (Park et al., 2007; Tolosi and Lengauer, 2011).

The cluster method determines the algorithm used to form the clusters. The following cluster methods are implemented:

- none: No clustering is performed.
- hclust (default): Hierarchical agglomerative clustering. If the fastcluster package is installed, fastcluster::hclust is used (Muellner 2013), otherwise stats::hclust is used.
- agnes: Hierarchical clustering using agglomerative nesting (Kaufman and Rousseeuw, 1990). This algorithm is similar to hclust, but uses the cluster::agnes implementation.
- diana: Divisive analysis hierarchical clustering. This method uses divisive instead of agglomerative clustering (Kaufman and Rousseeuw, 1990). cluster::diana is used.
- pam: Partioning around medioids. This partitions the data into $k$ clusters around medioids (Kaufman and Rousseeuw, 1990). $k$ is selected using the silhouette metric. pam is implemented using the cluster::pam function.

Clusters and cluster information is stored within featureInfo objects for later use with validation data sets. This enables reproduction of the same clusters as formed in the development data set.

cluster_linkage_method (*optional*) Linkage method used for agglomerative clustering in hclust and agnes. The following linkage methods can be used:

- average (default): Average linkage.
- single: Single linkage.
- complete: Complete linkage.
- weighted: Weighted linkage, also known as McQuitty linkage.
- ward: Linkage using Ward's minimum variance method.

diana and pam do not require a linkage method.

cluster_cut_method (*optional*) The method used to define the actual clusters. The following methods can be used:

- silhouette: Clusters are formed based on the silhouette score (Rousseeuw, 1987). The average silhouette score is computed from 2 to $n$ clusters, with $n$ the number of features. Clusters are only formed if the average silhouette exceeds 0.50, which indicates reasonable evidence for structure. This procedure may be slow if the number of features is large (>100s).
- fixed_cut: Clusters are formed by cutting the hierarchical tree at the point indicated by the cluster_similarity_threshold, e.g. where features in a cluster have an average Spearman correlation of 0.90. fixed_cut is only available for agnes, diana and hclust.
- dynamic_cut: Dynamic cluster formation using the cutting algorithm in the dynamicTreeCut package. This package should be installed to select this option. dynamic_cut can only be used with agnes and hclust.

The default options are silhouette for partioning around medioids (pam) and fixed_cut otherwise.

cluster_similarity_metric (*optional*) Clusters are formed based on feature similarity. All features are compared in a pair-wise fashion to compute similarity, for example correlation. The resulting similarity grid is converted into a distance matrix that is subsequently used for clustering. The following metrics are supported to compute pairwise similarities:

- mutual_information (default): normalised mutual information.
- mcfadden_r2: McFadden's pseudo R-squared (McFadden, 1974).
- cox_snell_r2: Cox and Snell's pseudo R-squared (Cox and Snell, 1989).
- nagelkerke_r2: Nagelkerke's pseudo R-squared (Nagelkerke, 1991).
- spearman: Spearman's rank order correlation.
- kendall: Kendall rank correlation.
- pearson: Pearson product-moment correlation.

The pseudo R-squared metrics can be used to assess similarity between mixed pairs of numeric and categorical features, as these are based on the log-likelihood of regression models. In familiar, the more informative feature is used as the predictor and the other feature as the reponse variable. In numeric-categorical pairs, the numeric feature is considered to be more informative and is thus used as the predictor. In categorical-categorical pairs, the feature with most levels is used as the predictor.

In case any of the classical correlation coefficients (pearson, spearman and kendall) are used with (mixed) categorical features, the categorical features are one-hot encoded and the mean correlation over all resulting pairs is used as similarity.

cluster_similarity_threshold (*optional*) The threshold level for pair-wise similarity that is required to form clusters using fixed_cut. This should be a numerical value between 0.0 and 1.0. Note however, that a reasonable threshold value depends strongly on the similarity metric. The following are the default values used:

- mcfadden_r2 and mutual_information: 0.30

- cox_snell_r2 and nagelkerke_r2: 0.75
- spearman, kendall and pearson: 0.90

Alternatively, if the fixed cut method is not used, this value determines whether any clustering should be performed, because the data may not contain highly similar features. The default values in this situation are:

- mcfadden_r2 and mutual_information: 0.25
- cox_snell_r2 and nagelkerke_r2: 0.40
- spearman, kendall and pearson: 0.70

The threshold value is converted to a distance (1-similarity) prior to cutting hierarchical trees.

cluster_representation_method (*optional*) Method used to determine how the information of co-clustered features is summarised and used to represent the cluster. The following methods can be selected:

- best_predictor (default): The feature with the highest importance according to univariate regression with the outcome is used to represent the cluster.
- medioid: The feature closest to the cluster center, i.e. the feature that is most similar to the remaining features in the cluster, is used to represent the feature.
- mean: A meta-feature is generated by averaging the feature values for all features in a cluster. This method aligns all features so that all features will be positively correlated prior to averaging. Should a cluster contain one or more categorical features, the medioid method will be used instead, as averaging is not possible. Note that if this method is chosen, the normalisation_method parameter should be one of standardisation, standardisation_trim, standardisation_winsor or quantile.‘

If the pam cluster method is selected, only the medioid method can be used. In that case 1 medioid is used by default.

parallel_preprocessing (*optional*) Enable parallel processing for the preprocessing workflow. Defaults to TRUE. When set to FALSE, this will disable the use of parallel processing while preprocessing, regardless of the settings of the parallel parameter. parallel_preprocessing is ignored if parallel=FALSE.

config A list of settings, e.g. from an xml file.

vimp_aggregation_method (*optional*) The method used to aggregate variable importances over different data subsets, e.g. bootstraps. The following methods can be selected:

- none: Don’t aggregate ranks, but rather aggregate the variable importance scores themselves.
- mean: Use the mean rank of a feature over the subsets to determine the aggregated feature rank.
- median: Use the median rank of a feature over the subsets to determine the aggregated feature rank.
- best: Use the best rank the feature obtained in any subset to determine the aggregated feature rank.

- worst: Use the worst rank the feature obtained in any subset to determine the aggregated feature rank.
- stability: Use the frequency of the feature being in the subset of highly ranked features as measure for the aggregated feature rank (Meinshausen and Buehlmann, 2010).
- exponential: Use a rank-weighted frequence of occurrence in the subset of highly ranked features as measure for the aggregated feature rank (Haury et al., 2011).
- borda (default): Use the borda count as measure for the aggregated feature rank (Wald et al., 2012).
- enhanced_borda: Use an occurrence frequency-weighted borda count as measure for the aggregated feature rank (Wald et al., 2012).
- truncated_borda: Use borda count computed only on features within the subset of highly ranked features.
- enhanced_truncated_borda: Apply both the enhanced borda method and the truncated borda method and use the resulting borda count as the aggregated feature rank.

The *feature selection methods* vignette provides additional information.

vimp_aggregation_rank_threshold (*optional*) The threshold used to define the subset of highly important features. If set to NULL, this threshold is determined by maximising the variance in the occurrence value over all features over the subset size. The default value is 5.
This parameter is only relevant for stability, exponential, enhanced_borda, truncated_borda and enhanced_truncated_borda methods.

## Details

This is a thin wrapper around summon_familiar, and functions like it, but automatically skips learning and subsequent evaluation steps.

The function returns an experimentData object, which can be used to warm-start other experiments by providing it to the experiment_data argument. Variable importance may be retrieved from this object using the get_vimp_table and aggregate_vimp_table methods.

## Value

An experimentData object.

## See Also

get_vimp_table, aggregate_vimp_table

predict *Model predictions for familiar models and model ensembles*

## Description

Fits the model or ensemble of models to the data and shows the result.

## Usage

```
predict(object, ...)

## S4 method for signature 'familiarModel'
predict(
  object,
  newdata,
  type = "default",
  time = NULL,
  dir_path = NULL,
  ensemble_method = "median",
  stratification_threshold = NULL,
  stratification_method = NULL,
  percentiles = NULL,
  .as_prediction_table = FALSE,
  ...
)

## S4 method for signature 'familiarEnsemble'
predict(
  object,
  newdata,
  type = "default",
  time = NULL,
  dir_path = NULL,
  ensemble_method = "median",
  stratification_threshold = NULL,
  stratification_method = NULL,
  percentiles = NULL,
  .as_prediction_table = FALSE,
  ...
)

## S4 method for signature 'familiarNoveltyDetector'
predict(
  object,
  newdata,
  type = "novelty",
  ensemble_method = "median",
```

```
  .as_prediction_table = FALSE,
  ...
)

## S4 method for signature 'list'
predict(
  object,
  newdata,
  type = "default",
  time = NULL,
  dir_path = NULL,
  ensemble_method = "median",
  stratification_threshold = NULL,
  stratification_method = NULL,
  percentiles = NULL,
  .as_prediction_table = FALSE,
  ...
)

## S4 method for signature 'character'
predict(
  object,
  newdata,
  type = "default",
  time = NULL,
  dir_path = NULL,
  ensemble_method = "median",
  stratification_threshold = NULL,
  stratification_method = NULL,
  percentiles = NULL,
  .as_prediction_table = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A familiar model or ensemble of models that should be used for prediction. This can also be a path to the ensemble model, one or more paths to models, or a list of models. |
| ... | to be documented. |
| newdata | Data to which the models are fitted. familiar performs checks on the data to ensure that all features required for fitting the model are present, and no additional levels are present in categorical features. Unlike other predict methods, newdata cannot be missing in familiar, as training data are not stored with the models. |
| type | Type of prediction made. The following values are directly supported:<br>• default: Default prediction, i.e. value estimates for continuous outcomes, predicted class probabilities and class for binomial and multinomial |

and the model response for `survival` outcomes.

- `survival_probability`: Predicts survival probabilities at the time specified by `time`. Only applicable to `survival` outcomes. Some models may not allow for predicting survival probabilities based on their response.
- `novelty`: Predicts novelty of each sample, which can be used for out-of-distribution detection.
- `risk_stratification`: Predicts the strata to which the data belongs. Only for `survival` outcomes.

Other values for type are passed to the fitting method of the actual underlying model. For example for generalised linear models (`glm`) `type` can be `link`, `response` or `terms` as well. Some of these model-specific prediction types may fail to return results if the model has been trimmed.

`time` Time at which the response (`default`) or survival probability (`survival_probability`) should be predicted for `survival` outcomes. Some models have a response that does not depend on `time`, e.g. `cox`, whereas others do, e.g. `random_forest`.

`dir_path` Path to the folder containing the models. Ensemble objects are stored with the models detached. In case the models were moved since creation, `dir_path` can be used to specify the current folder. Alternatively the `update_model_dir_path` method can be used to update the path.

`ensemble_method`
Method for ensembling predictions from models for the same sample. Available methods are:

- `median` (default): Use the median of the predicted values as the ensemble value for a sample.
- `mean`: Use the mean of the predicted values as the ensemble value for a sample.

`stratification_threshold`
Threshold value(s) used for stratifying instances into risk groups. If this parameter is specified, `stratification_method` and any threshold values that come with the model are ignored, and `stratification_threshold` is used instead.

`stratification_method`
Selects the stratification method from which the threshold values should be selected. If the model or ensemble of models does not contain thresholds for the indicated method, an error is returned. In addition this argument is ignored if a `stratification_threshold` is set.

`percentiles` Currently unused.

`.as_prediction_table`
Selects whether a data.table or prediction table object should be returned.

## Details

This method is used to predict values for instances specified by the `newdata` using the model or ensemble of models specified by the `object` argument.

## Value

A `data.table` with predicted values.

`set_class_names,familiarCollection-method`
*Rename outcome classes for plotting and export*

## Description

Tabular exports and figures created from a familiarCollection object can be customised by providing names for outcome classes.

## Usage

```
## S4 method for signature 'familiarCollection'
set_class_names(x, old = NULL, new = NULL, order = NULL)
```

## Arguments

| | |
|---|---|
| `x` | A familiarCollection object. |
| `old` | (optional) Set of old labels to replace. |
| `new` | Set of replacement labels. The number of replacement labels should be equal to the number of provided old labels or the full number of labels. If a subset of labels is to be replaced, both `old` and `new` should be provided. |
| `order` | (optional) Ordered set of replacement labels. This is used to provide the order in which the labels should be placed, which affects e.g. levels in a plot. If the ordering is not explicitly provided, the old ordering is used. |

## Details

Labels convert the internal naming for class levels to the requested label at export or when plotting. This enables customisation of class names. Currently assigned labels can be found using the `get_class_names` method.

## Value

A familiarCollection object with updated labels.

## See Also

- familiarCollection for information concerning the familiarCollection class. * `get_class_names` for obtaining currently assigned class names.

`set_data_set_names,familiarCollection-method`
*Name datasets for plotting and export*

## Description

Tabular exports and figures created from a familiarCollection object can be customised by setting data labels.

## Usage

```
## S4 method for signature 'familiarCollection'
set_data_set_names(x, old = NULL, new = NULL, order = NULL)
```

## Arguments

| | |
|---|---|
| `x` | A familiarCollection object. |
| `old` | (optional) Set of old labels to replace. |
| `new` | Set of replacement labels. The number of replacement labels should be equal to the number of provided old labels or the full number of labels. If a subset of labels is to be replaced, both `old` and `new` should be provided. |
| `order` | (optional) Ordered set of replacement labels. This is used to provide the order in which the labels should be placed, which affects e.g. levels in a plot. If the ordering is not explicitly provided, the old ordering is used. |

## Details

Labels convert internal naming of data sets to the requested label at export or when plotting. Currently assigned labels can be found using the `get_data_set_names` method.

## Value

A familiarCollection object with custom names for the data sets.

## See Also

- familiarCollection for information concerning the familiarCollection class. * `get_data_set_names` for obtaining currently assigned labels.

`set_feature_names,familiarCollection-method`
*Rename features for plotting and export*

## Description

Tabular exports and figures created from a familiarCollection object can be customised by providing names for features.

## Usage

```
## S4 method for signature 'familiarCollection'
set_feature_names(x, old = NULL, new = NULL, order = NULL)
```

## Arguments

| | |
|---|---|
| `x` | A familiarCollection object. |
| `old` | (optional) Set of old labels to replace. |
| `new` | Set of replacement labels. The number of replacement labels should be equal to the number of provided old labels or the full number of labels. If a subset of labels is to be replaced, both `old` and `new` should be provided. |
| `order` | (optional) Ordered set of replacement labels. This is used to provide the order in which the labels should be placed, which affects e.g. levels in a plot. If the ordering is not explicitly provided, the old ordering is used. |

## Details

Labels convert the internal naming for features to the requested label at export or when plotting. This enables customisation without redoing the analysis with renamed input data. Currently assigned labels can be found using the `get_feature_names` method.

## Value

A familiarCollection object with updated labels.

## See Also

- familiarCollection for information concerning the familiarCollection class. * `get_feature_names` for obtaining currently assigned feature names.

set_learner_names,familiarCollection-method
*Rename learners for plotting and export*

## Description

Tabular exports and figures created from a familiarCollection object can be customised by providing names for the learners.

## Usage

```
## S4 method for signature 'familiarCollection'
set_learner_names(x, old = NULL, new = NULL, order = NULL)
```

## Arguments

| | |
|---|---|
| x | A familiarCollection object. |
| old | (optional) Set of old labels to replace. |
| new | Set of replacement labels. The number of replacement labels should be equal to the number of provided old labels or the full number of labels. If a subset of labels is to be replaced, both old and new should be provided. |
| order | (optional) Ordered set of replacement labels. This is used to provide the order in which the labels should be placed, which affects e.g. levels in a plot. If the ordering is not explicitly provided, the old ordering is used. |

## Details

Labels convert the internal naming for learners to the requested label at export or when plotting. This enables the use of more specific naming, e.g. changing random_forest_rfsrc to Random Forest. Currently assigned labels can be found using the get_learner_names method.

## Value

A familiarCollection object with custom labels for the learners.

## See Also

- familiarCollection for information concerning the familiarCollection class. * get_learner_names for obtaining currently assigned labels.

`set_risk_group_names,familiarCollection-method`
*Rename risk groups for plotting and export*

## Description

Tabular exports and figures created from a familiarCollection object can be customised by providing names for risk groups in survival analysis.

## Usage

```
## S4 method for signature 'familiarCollection'
set_risk_group_names(x, old = NULL, new = NULL, order = NULL)
```

## Arguments

| | |
|---|---|
| `x` | A familiarCollection object. |
| `old` | (optional) Set of old labels to replace. |
| `new` | Set of replacement labels. The number of replacement labels should be equal to the number of provided old labels or the full number of labels. If a subset of labels is to be replaced, both `old` and `new` should be provided. |
| `order` | (optional) Ordered set of replacement labels. This is used to provide the order in which the labels should be placed, which affects e.g. levels in a plot. If the ordering is not explicitly provided, the old ordering is used. |

## Details

Labels convert the internal naming for risk groups to the requested label at export or when plotting. This enables customisation of risk group names. Currently assigned labels can be found using the `get_risk_group_names` method.

## Value

A familiarCollection object with updated labels.

## See Also

- familiarCollection for information concerning the familiarCollection class. * `get_risk_group_names` for obtaining currently assigned risk group labels.

---

`set_vimp_method_names,familiarCollection-method`
*Rename variable importance methods for plotting and export*

---

## Description

Tabular exports and figures created from a familiarCollection object can be customised by providing names for the variable importance methods.

## Usage

```
## S4 method for signature 'familiarCollection'
set_vimp_method_names(x, old = NULL, new = NULL, order = NULL)
```

## Arguments

| | |
|---|---|
| `x` | A familiarCollection object. |
| `old` | (optional) Set of old labels to replace. |
| `new` | Set of replacement labels. The number of replacement labels should be equal to the number of provided old labels or the full number of labels. If a subset of labels is to be replaced, both `old` and `new` should be provided. |
| `order` | (optional) Ordered set of replacement labels. This is used to provide the order in which the labels should be placed, which affects e.g. levels in a plot. If the ordering is not explicitly provided, the old ordering is used. |

## Details

Labels convert the internal naming for variable importance methods to the requested label at export or when plotting. This enables the use of more specific naming, e.g. changing `mim` to `Mutual Information Maximisation`. Currently assigned labels can be found using the `get_vimp_method_names` method.

## Value

A familiarCollection object with updated labels.

## See Also

- familiarCollection for information concerning the familiarCollection class. * `get_vimp_method_names` for obtaining currently assigned labels.

summary *Model summaries*

## Description

summary produces model summaries.

## Usage

```
summary(object, ...)

## S4 method for signature 'familiarModel'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| object | a familiarModel object |
| ... | additional arguments passed to summary methods for the underlying model, when available. |

## Details

This method extends the summary S3 method. For some models summary requires information that is trimmed from the model. In this case a copy of summary data is stored with the model, and returned.

## Value

Depends on underlying model. See the documentation for the particular models.

summon_familiar *Perform end-to-end machine learning and data analysis*

## Description

Perform end-to-end machine learning and data analysis

## Usage

```
summon_familiar(
  formula = NULL,
  data = NULL,
  experiment_data = NULL,
  cl = NULL,
  config = NULL,
```

```
  config_id = 1L,
  verbose = TRUE,
  .stop_after = "export",
  .force_output = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| formula | An R formula. The formula can only contain feature names and dot (.). The * and +1 operators are not supported as these refer to columns that are not present in the data set.<br>Use of the formula interface is optional. |
| data | A data.table object, a data.frame object, list containing multiple data.table or data.frame objects, or paths to data files.<br>data should be provided if no file paths are provided to the data_files argument. If both are provided, only data will be used.<br>All data is expected to be in wide format, and ideally has a sample identifier (see sample_id_column), batch identifier (see cohort_column) and outcome columns (see outcome_column).<br>In case paths are provided, the data should be stored as csv, rds or RData files. See documentation for the data_files argument for more information. |
| experiment_data | Experimental data may provided in the form of the output of precompute_data_assignment, precompute_feature_info or precompute_vimp. This allows for warm-starting experiments. |
| cl | Cluster created using the parallel package. This cluster is then used to speed up computation through parallelisation. When a cluster is not provided, parallelisation is performed by setting up a cluster on the local machine.<br>This parameter has no effect if the parallel argument is set to FALSE. |
| config | List containing configuration parameters, or path to an xml file containing these parameters. An empty configuration file can obtained using the get_xml_config function.<br>All parameters can also be set programmatically. These supersede any arguments derived from the configuration list. |
| config_id | Identifier for the configuration in case the list or xml table indicated by config contains more than one set of configurations. |
| verbose | Indicates verbosity of the results. Default is TRUE, and all messages and warnings are returned. |
| .stop_after | Variable for internal use. |
| .force_output | Generates output even if results have been written to the file system. |
| ... | Arguments passed on to .parse_file_paths, .parse_experiment_settings, .parse_setup_settings, .parse_preprocessing_settings, .parse_variable_importance_setti .parse_model_development_settings, .parse_hyperparameter_optimisation_settings, .parse_evaluation_settings |

project_dir (*optional*) Path to the project directory. familiar checks if the directory indicated by experiment_dir and data files in data_file are relative to the project_dir.

experiment_dir (**recommended**) Path to the directory where all intermediate and final results produced by familiar are written to.

The experiment_dir can be a path relative to project_dir or an absolute path.

In case no project directory is provided and the experiment directory is not on an absolute path, a directory will be created in the temporary R directory indicated by tempdir(). This directory is deleted after closing the R session or once data analysis has finished. All information will be lost afterwards. Hence, it is recommended to provide either experiment_dir as an absolute path, or provide both project_dir and experiment_dir.

data_file (*optional*) Path to files containing data that should be analysed. The paths can be relative to project_dir or absolute paths. An error will be raised if the file cannot be found.

The following types of data are supported.

- csv files containing column headers on the first row, and samples per row. csv files are read using data.table::fread.
- rds files that contain a data.table or data.frame object. rds files are imported using base::readRDS.
- RData files that contain a single data.table or data.frame object. RData files are imported using base::load.

All data are expected in wide format, with sample information organised row-wise.

More than one data file can be provided. familiar will try to combine data files based on column names and identifier columns.

Alternatively, data can be provided using the data argument. These data are expected to be data.frame or data.table objects or paths to data files. The latter are handled in the same way as file paths provided to data_file.

batch_id_column (**recommended**) Name of the column containing batch or cohort identifiers. This parameter is required if more than one dataset is provided, or if external validation is performed.

In familiar any row of data is organised by four identifiers:

- The batch identifier batch_id_column: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.
- The sample identifier sample_id_column: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.
- The series identifier series_id_column: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.
- The repetition identifier: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries

for the same series of the same sample in the same batch that share the same outcome are encountered.

`sample_id_column` (**recommended**) Name of the column containing sample or subject identifiers. See `batch_id_column` above for more details.
If unset, every row will be identified as a single sample.

`series_id_column` (*optional*) Name of the column containing series identifiers, which distinguish between measurements that are part of a series for a single sample. See `batch_id_column` above for more details.
If unset, rows which share the same batch and sample identifiers but have a different outcome are assigned unique series identifiers.

`development_batch_id` (*optional*) One or more batch or cohort identifiers to constitute data sets for development. Defaults to all, or all minus the identifiers in `validation_batch_id` for external validation. Required if external validation is performed and `validation_batch_id` is not provided.

`validation_batch_id` (*optional*) One or more batch or cohort identifiers to constitute data sets for external validation. Defaults to all data sets except those in `development_batch_id` for external validation, or none if not. Required if `development_batch_id` is not provided.

`outcome_name` (*optional*) Name of the modelled outcome. This name will be used in figures created by `familiar`.
If not set, the column name in `outcome_column` will be used for `binomial`, `multinomial`, and `continuous` outcomes. For other outcomes (`survival` and `competing_risk`) no default is used.

`outcome_column` (**recommended**) Name of the column containing the outcome of interest. May be identified from a formula, if a formula is provided as an argument. Otherwise an error is raised. Note that `survival` and `competing_risk` outcome type outcomes require two columns that indicate the time-to-event or the time of last follow-up and the event status.

`outcome_type` (**recommended**) Type of outcome found in the outcome column. The outcome type determines many aspects of the overall process, e.g. the available variable importance methods and learners, but also the type of assessments that can be conducted to evaluate the resulting models.
Implemented outcome types are:

- `binomial`: categorical outcome with 2 levels.
- `multinomial`: categorical outcome with 2 or more levels.
- `continuous`: general continuous numeric outcomes.
- `survival`: survival outcome for time-to-event data.

If not provided, the algorithm will attempt to obtain outcome_type from contents of the outcome column. This may lead to unexpected results, and we therefore advise to provide this information manually.
Note that `competing_risk` survival analysis are not fully supported, and is currently not a valid choice for `outcome_type`. The `count` outcome type was deprecated in version 2.0.0, and superseded by `continuous`.

`class_levels` (*optional*) Class levels for `binomial` or `multinomial` outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column.

event_indicator (**recommended**) Indicator for events in survival and competing_risk analyses. familiar will automatically recognise 1, true, t, y and yes as event indicators, including different capitalisations. If this parameter is set, it replaces the default values.

censoring_indicator (**recommended**) Indicator for right-censoring in survival and competing_risk analyses. familiar will automatically recognise 0, false, f, n, no as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values.

competing_risk_indicator (**recommended**) Indicator for competing risks in competing_risk analyses. There are no default values, and if unset, all values other than those specified by the event_indicator and censoring_indicator parameters are considered to indicate competing risks.

signature (*optional*) One or more names of feature columns that are considered part of a specific signature. Features specified here will always be used for modelling. Ranking of variable importances has no effect for these features.

novelty_features (*optional*) One or more names of feature columns that should be included for the purpose of novelty detection.

exclude_features (*optional*) Feature columns that will be removed from the data set. Cannot overlap with features in signature, novelty_features or include_features.

include_features (*optional*) Feature columns that are specifically included in the data set. By default all features are included. Cannot overlap with exclude_features, but may overlap signature. Features in signature and novelty_features are always included. If both exclude_features and include_features are provided, include_features takes precedence, provided that there is no overlap between the two.

reference_method (*optional*) Method used to set reference levels for categorical features. There are several options:

- auto (default): Categorical features that are not explicitly set by the user, i.e. columns containing boolean values or characters, use the most frequent level as reference. Categorical features that are explicitly set, i.e. as factors, are used as is.
- always: Both automatically detected and user-specified categorical features have the reference level set to the most frequent level. Ordinal features are not altered, but are used as is.
- never: User-specified categorical features are used as is. Automatically detected categorical features are simply sorted, and the first level is then used as the reference level. This was the behaviour prior to familiar version 1.3.0.

experimental_design (**required**) Defines what the experiment looks like, e.g. cv(bt(fs,20)+mb,3,2)+ev for 2 times repeated 3-fold cross-validation with nested variable importance computation on 20 bootstraps and model-building, and external validation. The basic workflow components are:

- fs: (optional) variable importance computation step. If not explicitly declared, feature selection will be done just in time for hyperparameter optimisation.

- mb: (required) model building step.
- ev: (optional) external validation. Note that internal validation due to subsampling will always be conducted if the subsampling methods create any validation data sets.

The different components are linked using +.
Different subsampling methods can be used in conjunction with the basic workflow components:

- bs(x,n): (stratified) .632 bootstrap, with n the number of bootstraps. In contrast to bt, feature pre-processing parameters and hyperparameter optimisation are conducted on individual bootstraps.
- bt(x,n): (stratified) .632 bootstrap, with n the number of bootstraps. Unlike bs and other subsampling methods, no separate pre-processing parameters or optimised hyperparameters will be determined for each bootstrap.
- cv(x,n,p): (stratified) n-fold cross-validation, repeated p times. Pre-processing parameters are determined for each iteration.
- lv(x): leave-one-out-cross-validation. Pre-processing parameters are determined for each iteration.
- ip(x): imbalance partitioning for addressing class imbalances on the data set. Pre-processing parameters are determined for each partition. The number of partitions generated depends on the imbalance correction method (see the imbalance_correction_method parameter). Imbalance partitioning does not generate validation sets.

As shown in the example above, sampling algorithms can be nested.
The simplest valid experimental design is fs+mb, which corresponds to a TRIPOD type 1a analysis. Type 1b analyses are only possible using bootstraps, e.g. bt(fs+mb,100). Type 2a analyses can be conducted using cross-validation, e.g. cv(bt(fs,100)+mb,10,1). Depending on the origin of the external validation data, designs such as fs+mb+ev or cv(bt(fs,100)+mb,10,1)+ev constitute type 2b or type 3 analyses. Type 4 analyses can be done by obtaining one or more familiarModel objects from others and applying them to your own data set.
Alternatively, the experimental_design parameter may be used to provide a path to a file containing iterations, which is named ####_iterations.RDS by convention. This path can be relative to the directory of the current experiment (experiment_dir), or an absolute path. The absolute path may thus also point to a file from a different experiment.

imbalance_correction_method (*optional*) Type of method used to address class imbalances. Available options are:

- full_undersampling (default): All data will be used in an ensemble fashion. The full minority class will appear in each partition, but majority classes are undersampled until all data have been used.
- random_undersampling: Randomly undersamples majority classes. This is useful in cases where full undersampling would lead to the formation of many models due major overrepresentation of the largest class.

This parameter is only used in combination with imbalance partitioning in the experimental design, and `ip` should therefore appear in the string that defines the design.

`imbalance_n_partitions` (*optional*) Number of times random undersampling should be repeated. 10 undersampled subsets with balanced classes are formed by default.

`iteration_seed` (*optional*) Integer seed used in sampling algorithms specified by the `experimental_design` argument. This allows for creating the same sample assignments across different experiments – of course provided that the same dataset is used. By default a random seed is used.

`parallel` (*optional*) Enable parallel processing. Defaults to `TRUE`. When set to `FALSE`, this disables all parallel processing, regardless of specific parameters such as `parallel_preprocessing`. However, when `parallel` is `TRUE`, parallel processing of different parts of the workflow can be disabled by setting respective flags to `FALSE`.

`parallel_nr_cores` (*optional*) Number of cores available for parallelisation. Defaults to 2. This setting does nothing if parallelisation is disabled.

`restart_cluster` (*optional*) Restart nodes used for parallel computing to free up memory prior to starting a parallel process. Note that it does take time to set up the clusters. Therefore setting this argument to `TRUE` may impact processing speed. This argument is ignored if `parallel` is `FALSE` or the cluster was initialised outside of familiar. Default is `FALSE`, which causes the clusters to be initialised only once.

`cluster_type` (*optional*) Selection of the cluster type for parallel processing. Available types are the ones supported by the parallel package that is part of the base R distribution: `psock` (default), `fork`, `mpi`, `nws`, `sock`. In addition, `none` is available, which also disables parallel processing.

`backend_type` (*optional*) Selection of the backend for distributing copies of the data. This backend ensures that only a single master copy is kept in memory. This limits memory usage during parallel processing.
Several backend options are available, notably `socket_server`, and `none` (default). `socket_server` is based on the callr package and R sockets, comes with `familiar` and is available for any OS. `none` uses the package environment of familiar to store data, and is available for any OS. However, `none` requires copying of data to any parallel process, and has a larger memory footprint.

`server_port` (*optional*) Integer indicating the port on which the socket server or RServe process should communicate. Defaults to port 6311. Note that ports 0 to 1024 and 49152 to 65535 cannot be used.

`feature_max_fraction_missing` (*optional*) Numeric value between `0.0` and `0.95` that determines the meximum fraction of missing values that still allows a feature to be included in the data set. All features with a missing value fraction over this threshold are not processed further. The default value is `0.30`.

`sample_max_fraction_missing` (*optional*) Numeric value between `0.0` and `0.95` that determines the maximum fraction of missing values that still allows a sample to be included in the data set. All samples with a missing

value fraction over this threshold are excluded and not processed further. The default value is `0.30`.

`filter_method` (*optional*) One or methods used to reduce dimensionality of the data set by removing irrelevant or poorly reproducible features.
Several method are available:

- `none` (default): None of the features will be filtered.
- `low_variance`: Features with a variance below the `low_var_minimum_variance_threshold` are filtered. This can be useful to filter, for example, genes that are not differentially expressed.
- `univariate_test`: Features undergo a univariate regression using an outcome-appropriate regression model. The p-value of the model coefficient is collected. Features with coefficient p-value above the `univariate_test_threshold` are subsequently filtered.
- `robustness`: Features that are not sufficiently robust according to the intraclass correlation coefficient are filtered. Use of this method requires that repeated measurements are present in the data set, i.e. there should be entries for which the sample and cohort identifiers are the same.

More than one method can be used simultaneously. Features with singular values are always filtered, as these do not contain information.

`univariate_test_threshold` (*optional*) Numeric value between `1.0` and `0.0` that determines which features are irrelevant and will be filtered by the `univariate_test`. p-values are compared to this threshold. All features with values above the threshold are filtered. The default value is `0.20`.

`univariate_test_threshold_metric` (*optional*) Metric used with the to compare the `univariate_test_threshold` against. The following metric can be chosen:

- `p_value` (default): The unadjusted p-value of each feature is used for to filter features.

`univariate_test_max_feature_set_size` (*optional*) Maximum size of the feature set after the univariate test. P or q values of features are compared against the threshold, but if the resulting data set would be larger than this setting, only the most relevant features up to the desired feature set size are selected.
The default value is `NULL`, which causes features to be filtered based on their relevance only.

`low_var_minimum_variance_threshold` (required, if used) Numeric value that determines which features will be filtered by the `low_variance` method. The variance of each feature is computed and compared to the threshold. If it is below the threshold, the feature is removed.
This parameter has no default value and should be set if `low_variance` is used.

`low_var_max_feature_set_size` (*optional*) Maximum size of the feature set after filtering features with a low variance. All features are first compared against `low_var_minimum_variance_threshold`. If the resulting feature set would be larger than specified, only the most strongly varying features will be selected, up to the desired size of the feature set.

The default value is NULL, which causes features to be filtered based on their variance only.

robustness_icc_type (*optional*) String indicating the type of intraclass correlation coefficient (1, 2 or 3) that should be used to compute robustness for features in repeated measurements. These types correspond to the types in Shrout and Fleiss (1979). The default value is 1.

robustness_threshold_metric (*optional*) String indicating which specific intraclass correlation coefficient (ICC) metric should be used to filter features. This should be one of:

- icc: The estimated ICC value itself.
- icc_low (default): The estimated lower limit of the 95% confidence interval of the ICC, as suggested by Koo and Li (2016).
- icc_panel: The estimated ICC value over the panel average, i.e. the ICC that would be obtained if all repeated measurements were averaged.
- icc_panel_low: The estimated lower limit of the 95% confidence interval of the panel ICC.

robustness_threshold_value (*optional*) The intraclass correlation coefficient value that is as threshold. The default value is 0.70.

transformation_method (*optional*) The transformation method used to change the distribution of the data to be more normal-like. The following methods are available:

- none: This disables transformation of features.
- yeo_johnson: Transformation using the location and scale invariant version of the Yeo-Johnson transformation (Yeo and Johnson, 2000; Zwanenburg and Löck, 2026).
- yeo_johnson_robust (default): A robust version of yeo_johnson. This method is less sensitive to outliers.
- yeo_johnson_conventional: As yeo_johnson, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Yeo and Johnson (2001).
- box_cox: Transformation using the location and scale invariant version of the Box-Cox transformation (Box and Cox, 1964; Zwanenburg and Löck, 2026).
- box_cox_robust: A robust version of yeo_johnson. This method is less sensitive to outliers.
- box_cox_conventional: As box_cox, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Box and Cox (1964). This method requires strictly positive feature values.

Transformation requires the power.transform package. Only features that contain numerical data are transformed. Transformation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets.

transformation_optimisation_criterion (*optional*) Transformation parameters are optimised using a criterion, conventionally maximum-likelihood-

estimation. power.transform implements multiple optimisation criteria, of which the following are available:

- mle (default): Optimisation using maximum likelihood estimation.
- cramer_von_mises: Optimisation using the Cramér-von Mises criterion. Zwanenburg and Löck (2026) found that this criterion was relatively robust against outliers.

transformation_gof_test_p_value (*optional*) Not all transformations will lead to features that are roughly normally distributed. Zwanenburg and Löck (2026) established a empirical goodness-of-fit test for central normality. This parameter sets the significance for rejecting the null-hypothesis that a feature distribution is centrally normal. When the null-hypothesis is rejected, no transformation is performed. The default value is NULL, which disables the test.

normalisation_method (*optional*) The normalisation method used to improve the comparability between numerical features that may have very different scales. The following normalisation methods can be chosen:

- none: This disables feature normalisation.
- standardisation: Features are normalised by subtraction of their mean values and division by their standard deviations. This causes every feature to be have a center value of 0.0 and standard deviation of 1.0.
- standardisation_trim: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- standardisation_winsor: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- standardisation_robust (default): A robust version of standardisation that relies on computing Huber's M-estimators for location and scale.
- normalisation: Features are normalised by subtraction of their minimum values and division by their ranges. This maps all feature values to a $[0, 1]$ interval.
- normalisation_trim: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- normalisation_winsor: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- quantile: Features are normalised by subtraction of their median values and division by their interquartile range.
- mean_centering: Features are centered by substracting the mean, but do not undergo rescaling.

Only features that contain numerical data are normalised. Normalisation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets.

batch_normalisation_method (*optional*) The method used for batch normalisation. Available methods are:

- none (default): This disables batch normalisation of features.
- standardisation: Features within each batch are normalised by subtraction of the mean value and division by the standard deviation in each batch.
- standardisation_trim: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- standardisation_winsor: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- standardisation_robust: A robust version of standardisation that relies on computing Huber's M-estimators for location and scale within each batch.
- normalisation: Features within each batch are normalised by subtraction of their minimum values and division by their range in each batch. This maps all feature values in each batch to a $[0, 1]$ interval.
- normalisation_trim: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- normalisation_winsor: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- quantile: Features in each batch are normalised by subtraction of the median value and division by the interquartile range of each batch.
- mean_centering: Features in each batch are centered on 0.0 by substracting the mean value in each batch, but are not rescaled.
- combat_parametric: Batch adjustments using parametric empirical Bayes (Johnson et al, 2007). combat_p leads to the same method.
- combat_non_parametric: Batch adjustments using non-parametric empirical Bayes (Johnson et al, 2007). combat_np and combat lead to the same method. Note that we reduced complexity from O($n^2$) to O($n$) by only computing batch adjustment parameters for each feature on a subset of 50 randomly selected features, instead of all features.

Only features that contain numerical data are normalised using batch normalisation. Batch normalisation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets, in case the validation data is from the same batch.

If validation data contains data from unknown batches, normalisation parameters are separately determined for these batches.

Note that for both empirical Bayes methods, the batch effect is assumed to produce results across the features. This is often true for things such as gene expressions, but the assumption may not hold generally.

When performing batch normalisation, it is moreover important to check that differences between batches or cohorts are not related to the studied endpoint.

imputation_method (*optional*) Method used for imputing missing feature values. Two methods are implemented:

- simple: Simple replacement of a missing value by the median value (for numeric features) or the modal value (for categorical features).
- lasso: Imputation of missing value by lasso regression (using glmnet) based on information contained in other features.

simple imputation precedes lasso imputation to ensure that any missing values in predictors required for lasso regression are resolved. The lasso estimate is then used to replace the missing value.

The default value depends on the number of features in the dataset. If the number is lower than 100, lasso is used by default, and simple otherwise.

Only single imputation is performed. Imputation models and parameters are stored within featureInfo objects for later use with validation data sets.

cluster_method (*optional*) Clustering is performed to identify and replace redundant features, for example those that are highly correlated. Such features do not carry much additional information and may be removed or replaced instead (Park et al., 2007; Tolosi and Lengauer, 2011).

The cluster method determines the algorithm used to form the clusters. The following cluster methods are implemented:

- none: No clustering is performed.
- hclust (default): Hierarchical agglomerative clustering. If the fastcluster package is installed, fastcluster::hclust is used (Muellner 2013), otherwise stats::hclust is used.
- agnes: Hierarchical clustering using agglomerative nesting (Kaufman and Rousseeuw, 1990). This algorithm is similar to hclust, but uses the cluster::agnes implementation.
- diana: Divisive analysis hierarchical clustering. This method uses divisive instead of agglomerative clustering (Kaufman and Rousseeuw, 1990). cluster::diana is used.
- pam: Partioning around medioids. This partitions the data into $k$ clusters around medioids (Kaufman and Rousseeuw, 1990). $k$ is selected using the silhouette metric. pam is implemented using the cluster::pam function.

Clusters and cluster information is stored within featureInfo objects for later use with validation data sets. This enables reproduction of the same clusters as formed in the development data set.

cluster_linkage_method (*optional*) Linkage method used for agglomerative clustering in hclust and agnes. The following linkage methods can be used:

- average (default): Average linkage.
- single: Single linkage.
- complete: Complete linkage.
- weighted: Weighted linkage, also known as McQuitty linkage.
- ward: Linkage using Ward's minimum variance method.

diana and pam do not require a linkage method.

cluster_cut_method (*optional*) The method used to define the actual clusters. The following methods can be used:

- silhouette: Clusters are formed based on the silhouette score (Rousseeuw, 1987). The average silhouette score is computed from 2 to $n$ clusters, with $n$ the number of features. Clusters are only formed if the average silhouette exceeds 0.50, which indicates reasonable evidence for structure. This procedure may be slow if the number of features is large (>100s).
- fixed_cut: Clusters are formed by cutting the hierarchical tree at the point indicated by the cluster_similarity_threshold, e.g. where features in a cluster have an average Spearman correlation of 0.90. fixed_cut is only available for agnes, diana and hclust.
- dynamic_cut: Dynamic cluster formation using the cutting algorithm in the dynamicTreeCut package. This package should be installed to select this option. dynamic_cut can only be used with agnes and hclust.

The default options are silhouette for partioning around medioids (pam) and fixed_cut otherwise.

cluster_similarity_metric (*optional*) Clusters are formed based on feature similarity. All features are compared in a pair-wise fashion to compute similarity, for example correlation. The resulting similarity grid is converted into a distance matrix that is subsequently used for clustering. The following metrics are supported to compute pairwise similarities:

- mutual_information (default): normalised mutual information.
- mcfadden_r2: McFadden's pseudo R-squared (McFadden, 1974).
- cox_snell_r2: Cox and Snell's pseudo R-squared (Cox and Snell, 1989).
- nagelkerke_r2: Nagelkerke's pseudo R-squared (Nagelkerke, 1991).
- spearman: Spearman's rank order correlation.
- kendall: Kendall rank correlation.
- pearson: Pearson product-moment correlation.

The pseudo R-squared metrics can be used to assess similarity between mixed pairs of numeric and categorical features, as these are based on the log-likelihood of regression models. In familiar, the more informative feature is used as the predictor and the other feature as the reponse variable. In numeric-categorical pairs, the numeric feature is considered to be more informative and is thus used as the predictor. In categorical-categorical pairs, the feature with most levels is used as the predictor.

In case any of the classical correlation coefficients (pearson, spearman and kendall) are used with (mixed) categorical features, the categorical features are one-hot encoded and the mean correlation over all resulting pairs is used as similarity.

cluster_similarity_threshold (*optional*) The threshold level for pair-wise similarity that is required to form clusters using fixed_cut. This should be a numerical value between 0.0 and 1.0. Note however, that a reasonable threshold value depends strongly on the similarity metric. The following are the default values used:

- mcfadden_r2 and mutual_information: 0.30

- cox_snell_r2 and nagelkerke_r2: 0.75
- spearman, kendall and pearson: 0.90

Alternatively, if the fixed cut method is not used, this value determines whether any clustering should be performed, because the data may not contain highly similar features. The default values in this situation are:

- mcfadden_r2 and mutual_information: 0.25
- cox_snell_r2 and nagelkerke_r2: 0.40
- spearman, kendall and pearson: 0.70

The threshold value is converted to a distance (1-similarity) prior to cutting hierarchical trees.

cluster_representation_method (*optional*) Method used to determine how the information of co-clustered features is summarised and used to represent the cluster. The following methods can be selected:

- best_predictor (default): The feature with the highest importance according to univariate regression with the outcome is used to represent the cluster.
- medioid: The feature closest to the cluster center, i.e. the feature that is most similar to the remaining features in the cluster, is used to represent the feature.
- mean: A meta-feature is generated by averaging the feature values for all features in a cluster. This method aligns all features so that all features will be positively correlated prior to averaging. Should a cluster contain one or more categorical features, the medioid method will be used instead, as averaging is not possible. Note that if this method is chosen, the normalisation_method parameter should be one of standardisation, standardisation_trim, standardisation_winsor or quantile.‘

If the pam cluster method is selected, only the medioid method can be used. In that case 1 medioid is used by default.

parallel_preprocessing (*optional*) Enable parallel processing for the preprocessing workflow. Defaults to TRUE. When set to FALSE, this will disable the use of parallel processing while preprocessing, regardless of the settings of the parallel parameter. parallel_preprocessing is ignored if parallel=FALSE.

vimp_method (**required**) Variable importance method. familiar implements various variable importance methods. Please refer to the vignette on variable importance methods for more details.

More than one variable importance method can be chosen. The experiment will then repeated for each feature selection method.

Variable importance methods determine the ranking of features. Actual selection of features is done by optimising the signature size model hyperparameter during the hyperparameter optimisation step.

vimp_method_parameter (*optional*) List of lists containing parameters for feature selection methods. Each sublist should have the name of the feature selection method it corresponds to.

Most feature selection methods do not have parameters that can be set. Please refer to the vignette on feature selection methods for more details.

Note that if the feature selection method is based on a learner (e.g. lasso regression), hyperparameter optimisation may be performed prior to assessing variable importance.

`vimp_aggregation_method` (*optional*) The method used to aggregate variable importances over different data subsets, e.g. bootstraps. The following methods can be selected:

- `none`: Don't aggregate ranks, but rather aggregate the variable importance scores themselves.
- `mean`: Use the mean rank of a feature over the subsets to determine the aggregated feature rank.
- `median`: Use the median rank of a feature over the subsets to determine the aggregated feature rank.
- `best`: Use the best rank the feature obtained in any subset to determine the aggregated feature rank.
- `worst`: Use the worst rank the feature obtained in any subset to determine the aggregated feature rank.
- `stability`: Use the frequency of the feature being in the subset of highly ranked features as measure for the aggregated feature rank (Meinshausen and Buehlmann, 2010).
- `exponential`: Use a rank-weighted frequence of occurrence in the subset of highly ranked features as measure for the aggregated feature rank (Haury et al., 2011).
- `borda` (default): Use the borda count as measure for the aggregated feature rank (Wald et al., 2012).
- `enhanced_borda`: Use an occurrence frequency-weighted borda count as measure for the aggregated feature rank (Wald et al., 2012).
- `truncated_borda`: Use borda count computed only on features within the subset of highly ranked features.
- `enhanced_truncated_borda`: Apply both the enhanced borda method and the truncated borda method and use the resulting borda count as the aggregated feature rank.

The *feature selection methods* vignette provides additional information.

`vimp_aggregation_rank_threshold` (*optional*) The threshold used to define the subset of highly important features. If set to `NULL`, this threshold is determined by maximising the variance in the occurrence value over all features over the subset size. The default value is `5`.
This parameter is only relevant for `stability`, `exponential`, `enhanced_borda`, `truncated_borda` and `enhanced_truncated_borda` methods.

`parallel_vimp` (*optional*) Enable parallel processing for variable importance tasks. Defaults to `TRUE`. When set to `FALSE`, this will disable the use of parallel processing while performing feature selection, regardless of the settings of the `parallel` parameter. `parallel_vimp` is ignored if `parallel = FALSE`.

`learner` (**required**) One or more algorithms used for model development. A sizeable number learners is supported in `familiar`. Please see the vignette on learners for more information concerning the available learners.

hyperparameter (*optional*) List of lists containing hyperparameters for learners. Each sublist should have the name of the learner method it corresponds to, with list elements being named after the intended hyperparameter, e.g. "glm_logistic"=list("sign_size"=3)

All learners have hyperparameters. Please refer to the vignette on learners for more details. If no parameters are provided, sequential model-based optimisation is used to determine optimal hyperparameters.

Hyperparameters provided by the user are never optimised. However, if more than one value is provided for a single hyperparameter, optimisation will be conducted using these values.

novelty_detector (*optional*) Specify the algorithm used for training a novelty detector. This detector can be used to identify out-of-distribution data prospectively.

detector_parameters (*optional*) List lists containing hyperparameters for novelty detectors. Currently not used.

parallel_model_development (*optional*) Enable parallel processing for the model development workflow. Defaults to TRUE. When set to FALSE, this will disable the use of parallel processing while developing models, regardless of the settings of the parallel parameter. parallel_model_development is ignored if parallel=FALSE.

optimisation_bootstraps (*optional*) Number of bootstraps that should be generated from the development data set. During the optimisation procedure one or more of these bootstraps (indicated by smbo_step_bootstraps) are used for model development using different combinations of hyperparameters. The effect of the hyperparameters is then assessed by comparing in-bag and out-of-bag model performance.

The default number of bootstraps is 50. Hyperparameter optimisation may finish before exhausting the set of bootstraps.

optimisation_determine_vimp (*optional*) Logical value that indicates whether variable importance is determined separately for each of the bootstraps created during the optimisation process (TRUE) or the applicable results from the variable importance computation step are used (FALSE).

Determining variable importance increases the initial computational overhead. However, it prevents positive biases for the out-of-bag data due to overlap of these data with the development data set used for the feature selection step. In this case, any hyperparameters of the variable importance method are not determined separately for each bootstrap, but those obtained during the variable importance computation step are used instead. In case multiple of such hyperparameter sets could be applicable, the set that will be used is randomly selected for each bootstrap.

This parameter only affects hyperparameter optimisation of learners. The default is TRUE.

smbo_random_initialisation (*optional*) String indicating the initialisation method for the hyperparameter space. Can be one of fixed_subsample (default), fixed, or random. fixed and fixed_subsample first create hyperparameter sets from a range of default values set by familiar. fixed_subsample then randomly draws up to smbo_n_random_sets from the grid. random does not rely upon a fixed grid, and randomly draws up to smbo_n_random_sets

hyperparameter sets from the hyperparameter space.

`smbo_n_random_sets` (*optional*) Number of random or subsampled hyperparameters drawn during the initialisation process. Default: `100`. Cannot be smaller than `10`. The parameter is not used when `smbo_random_initialisation` is `fixed`, as the entire pre-defined grid will be explored.

`max_smbo_iterations` (*optional*) Maximum number of intensify iterations of the SMBO algorithm. During an intensify iteration a run-off occurs between the current *best* hyperparameter combination and either 10 challenger combination with the highest expected improvement or a set of 20 random combinations.

Run-off with random combinations is used to force exploration of the hyperparameter space, and is performed every second intensify iteration, or if there is no expected improvement for any challenger combination.

If a combination of hyperparameters leads to better performance on the same data than the incumbent *best* set of hyperparameters, it replaces the incumbent set at the end of the intensify iteration.

The default number of intensify iteration is `20`. Iterations may be stopped early if the incumbent set of hyperparameters remains the same for `smbo_stop_convergent_iterat` iterations, or performance improvement is minimal. This behaviour is suppressed during the first 4 iterations to enable the algorithm to explore the hyperparameter space.

`smbo_stop_convergent_iterations` (*optional*) The number of subsequent convergent SMBO iterations required to stop hyperparameter optimisation early.

An iteration is convergent if the *best* parameter set has not changed or the optimisation score over the 4 most recent iterations has not changed beyond the tolerance level in `smbo_stop_tolerance`.

The default value is `3`.

`smbo_stop_tolerance` (*optional*) Tolerance for early stopping due to convergent optimisation score.

The default value depends on the square root of the number of samples (at the series level), and is `0.01` for 100 samples. This value is computed as `0.1 * 1 / sqrt(n_samples)`. The upper limit is `0.0001` for 1M or more samples.

`smbo_time_limit` (*optional*) Time limit (in minutes) for the optimisation process. Optimisation is stopped after this limit is exceeded. Time taken to determine variable importance for the optimisation process (see the `optimisation_determine_vimp` parameter) does not count.

The default is `NULL`, indicating that there is no time limit for the optimisation process. The time limit cannot be less than 1 minute.

`smbo_initial_bootstraps` (*optional*) The number of bootstraps taken from the set of `optimisation_bootstraps` as the bootstraps assessed initially.

The default value is `1`. The value cannot be larger than `optimisation_bootstraps`.

`smbo_step_bootstraps` (*optional*) The number of bootstraps taken from the set of `optimisation_bootstraps` bootstraps as the bootstraps assessed during the steps of each intensify iteration.

The default value is `3`. The value cannot be larger than `optimisation_bootstraps`.

`smbo_intensify_steps` (*optional*) The number of steps in each SMBO intensify iteration. Each step a new set of `smbo_step_bootstraps` bootstraps is

drawn and used in the run-off between the incumbent *best* hyperparameter combination and its challengers.

The default value is 5. Higher numbers allow for a more detailed comparison, but this comes with added computational cost.

`optimisation_metric` (*optional*) One or more metrics used to compute performance scores. See the vignette on performance metrics for the available metrics.

If unset, the following metrics are used by default:

- `auc_roc`: For `binomial` and `multinomial` models.
- `mse`: Mean squared error for `continuous` models.
- `concordance_index`: For `survival` models.

Multiple optimisation metrics can be specified. Actual metric values are converted to an objective value by comparison with a baseline metric value that derives from a trivial model, i.e. majority class for binomial and multinomial outcomes, the median outcome for count and continuous outcomes and a fixed risk or time for survival outcomes.

`optimisation_function` (*optional*) Type of optimisation function used to quantify the performance of a hyperparameter set. Model performance is assessed using the metric(s) specified by `optimisation_metric` on the in-bag (IB) and out-of-bag (OOB) samples of a bootstrap. These values are converted to objective scores with a standardised interval of $[-1.0, 1.0]$. Each pair of objective is subsequently used to compute an optimisation score. The optimisation score across different bootstraps is than aggregated to a summary score. This summary score is used to rank hyperparameter sets, and select the optimal set.

The combination of optimisation score and summary score is determined by the optimisation function indicated by this parameter:

- `validation` or `max_validation` (default): seeks to maximise OOB score.
- `balanced`: seeks to balance IB and OOB score.
- `stronger_balance`: similar to `balanced`, but with stronger penalty for differences between IB and OOB scores.
- `validation_minus_sd`: seeks to optimise the average OOB score minus its standard deviation.
- `validation_25th_percentile`: seeks to optimise the 25th percentile of OOB scores, and is conceptually similar to `validation_minus_sd`.
- `model_estimate`: seeks to maximise the OOB score estimate predicted by the hyperparameter learner (not available for random search).
- `model_estimate_minus_sd`: seeks to maximise the OOB score estimate minus its estimated standard deviation, as predicted by the hyperparameter learner (not available for random search).
- `model_balanced_estimate`: seeks to maximise the estimate of the balanced IB and OOB score. This is similar to the `balanced` score, and in fact uses a hyperparameter learner to predict said score (not available for random search).
- `model_balanced_estimate_minus_sd`: seeks to maximise the estimate of the balanced IB and OOB score, minus its estimated standard

deviation. This is similar to the balanced score, but takes into account its estimated spread.

Additional detail are provided in the *Learning algorithms and hyperparameter optimisation* vignette.

hyperparameter_learner (*optional*) Any point in the hyperparameter space has a single, scalar, optimisation score value that is *a priori* unknown. During the optimisation process, the algorithm samples from the hyperparameter space by selecting hyperparameter sets and computing the optimisation score value for one or more bootstraps. For each hyperparameter set the resulting values are distributed around the actual value. The learner indicated by hyperparameter_learner is then used to infer optimisation score estimates for unsampled parts of the hyperparameter space.
The following models are available:

- bayesian_additive_regression_trees or bart: Uses Bayesian Additive Regression Trees (Sparapani et al., 2021) for inference. Unlike standard random forests, BART allows for estimating posterior distributions directly and can extrapolate.
- gaussian_process (default): Creates a localised approximate Gaussian process for inference (Gramacy, 2016). This allows for better scaling than deterministic Gaussian Processes.
- random_forest: Creates a random forest for inference. Originally suggested by Hutter et al. (2011). A weakness of random forests is their lack of extrapolation beyond observed values, which limits their usefulness in exploiting promising areas of hyperparameter space.
- random or random_search: Forgoes the use of models to steer optimisation. Instead, a random search is performed.

acquisition_function (*optional*) The acquisition function influences how new hyperparameter sets are selected. The algorithm uses the model learned by the learner indicated by hyperparameter_learner to search the hyperparameter space for hyperparameter sets that are either likely better than the best known set (*exploitation*) or where there is considerable uncertainty (*exploration*). The acquisition function quantifies this (Shahriari et al., 2016).
The following acquisition functions are available, and are described in more detail in the *learner algorithms* vignette:

- improvement_probability: The probability of improvement quantifies the probability that the expected optimisation score for a set is better than the best observed optimisation score
- improvement_empirical_probability: Similar to improvement_probability, but based directly on optimisation scores predicted by the individual decision trees.
- expected_improvement (default): Computes expected improvement.
- upper_confidence_bound: This acquisition function is based on the upper confidence bound of the distribution (Srinivas et al., 2012).
- bayes_upper_confidence_bound: This acquisition function is based on the upper confidence bound of the distribution (Kaufmann et al., 2012).

exploration_method (*optional*) Method used to steer exploration in post-initialisation intensive searching steps. As stated earlier, each SMBO iteration step compares suggested alternative parameter sets with an incumbent **best** set in a series of steps. The exploration method controls how the set of alternative parameter sets is pruned after each step in an iteration. Can be one of the following:

- single_shot (default): The set of alternative parameter sets is not pruned, and each intensification iteration contains only a single intensification step that only uses a single bootstrap. This is the fastest exploration method, but only superficially tests each parameter set.
- successive_halving: The set of alternative parameter sets is pruned by removing the worst performing half of the sets after each step (Jamieson and Talwalkar, 2016).
- stochastic_reject: The set of alternative parameter sets is pruned by comparing the performance of each parameter set with that of the incumbent **best** parameter set using a paired Wilcoxon test based on shared bootstraps. Parameter sets that perform significantly worse, at an alpha level indicated by smbo_stochastic_reject_p_value, are pruned.
- none: The set of alternative parameter sets is not pruned.

smbo_stochastic_reject_p_value (*optional*) The p-value threshold used for the stochastic_reject exploration method.
The default value is 0.05.

parallel_hyperparameter_optimisation (*optional*) Enable parallel processing for hyperparameter optimisation. Defaults to TRUE. When set to FALSE, this will disable the use of parallel processing while performing optimisation, regardless of the settings of the parallel parameter. The parameter moreover specifies whether parallelisation takes place within the optimisation algorithm (inner, default), or in an outer loop ( outer) over learners, data subsamples, etc.
parallel_hyperparameter_optimisation is ignored if parallel=FALSE.

evaluate_top_level_only (*optional*) Flag that signals that only evaluation at the most global experiment level is required. Consider a cross-validation experiment with additional external validation. The global experiment level consists of data that are used for development, internal validation and external validation. The next lower experiment level are the individual cross-validation iterations.
When the flag is true, evaluations take place on the global level only, and no results are generated for the next lower experiment levels. In our example, this means that results from individual cross-validation iterations are not computed and shown. When the flag is false, results are computed from both the global layer and the next lower level.
Setting the flag to true saves computation time.

evaluation_elements (*optional*) Specifies which evaluation steps should be performed during the evaluation process. Defaults to all, indicating that all evaluation steps should be performed. It can have one or more of the following values:

- all, true: all evaluation steps are performed.
- none, false: no evaluation is performed.
- auc_data: data for assessing and plotting the area under the receiver operating characteristic curve are computed.
- calibration_data: data for assessing and plotting model calibration are computed.
- calibration_info: data required to assess calibration, such as baseline survival curves, are collected. These data will still be present in the models.
- confusion_matrix: data for assessing and plotting a confusion matrix are collected.
- decision_curve_analyis: data for performing a decision curve analysis are computed.
- feature_expressions: data for assessing and plotting sample clustering are computed.
- feature_similarity: data for assessing and plotting feature clusters not computed.
- fs_vimp: data for assessing and plotting feature selection-based variable importance are collected.
- hyperparameters: data for assessing model hyperparameters are collected. These data will still be present in the models.
- ice_data: data for individual conditional expectation and partial dependence plots are created.
- model_performance: data for assessing and visualising model performance are created.
- model_vimp: data for assessing and plotting model-based variable importance are collected.
- permutation_vimp: data for assessing and plotting model-agnostic permutation variable importance are computed.
- prediction_data: predictions for each sample are made and exported.
- risk_stratification_data: data for assessing and plotting Kaplan-Meier survival curves are collected.
- risk_stratification_info: data for assessing stratification into risk groups are computed.
- sample_similarity: data for assessing sample similarity are computed.
- shap: data for assessing marginal contributions of feature values in a SHAP analysis.
- univariate_analysis: data for assessing and plotting univariate feature importance are computed.

The logical intersect of the evaluation elements specified by evaluation_elements and skip_evaluation_elements is used to define which evaluation steps are performed.

skip_evaluation_elements (*optional*) Specifies which evaluation steps, if any, should be skipped as part of the evaluation process. Defaults to none, which

means that all relevant evaluation steps are performed. It can have one or more of the same values as `evaluation_elements`.

The logical intersect of the evaluation elements specified by `evaluation_elements` and `skip_evaluation_elements` is used to define which evaluation steps are performed.

`ensemble_method` (*optional*) Method for ensembling predictions from models for the same sample. Available methods are:

- `median` (default): Use the median of the predicted values as the ensemble value for a sample.
- `mean`: Use the mean of the predicted values as the ensemble value for a sample.

This parameter is only used if `detail_level` is `ensemble`.

`evaluation_metric` (*optional*) One or more metrics for assessing model performance. See the vignette on performance metrics for the available metrics.

Confidence intervals (or rather credibility intervals) are computed for each metric during evaluation. This is done using bootstraps, the number of which depends on the value of `confidence_level` (Davison and Hinkley, 1997).

If unset, the metric in the `optimisation_metric` variable is used.

`sample_limit` (*optional*) Set the upper limit of the number of samples that are used during evaluation steps. Cannot be fewer than 20.

This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. `list("sample_similarity"=100, "permutation_vimp"=1000)`.

This parameter can be set for the following data elements: `sample_similarity`, `shap`, `permutation_vimp`, and `ice_data`.

`n_important_features` (*optional*) Set the number of features that are evaluated in evaluation steps. Cannot be 0 or fewer.

This setting can be specified per data element by providing a parameter value in a named list with data elements, e.g. `list("ice_data"=10, "permutation_vimp"=5)`.

This parameter can be set for the following data elements: `ice_data`, `permutation_vimp`, and `shap`.

`detail_level` (*optional*) Sets the level at which results are computed and aggregated.

- `ensemble`: Results are computed at the ensemble level, i.e. over all models in the ensemble. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the model performance of the ensemble model for each bootstrap.
- `hybrid` (default): Results are computed at the level of models in an ensemble. This means that, for example, bias-corrected estimates of model performance are directly computed using the models in the ensemble. If there are at least 20 trained models in the ensemble, performance is computed for each model, in contrast to `ensemble` where performance is computed for the ensemble of models. If there are less than 20 trained models in the ensemble, bootstraps are created so that at least 20 point estimates can be made.

- model: Results are computed at the model level. This means that, for example, bias-corrected estimates of model performance are assessed by creating (at least) 20 bootstraps and computing the performance of the model for each bootstrap.

Note that each level of detail has a different interpretation for bootstrap confidence intervals. For ensemble and model these are the confidence intervals for the ensemble and an individual model, respectively. That is, the confidence interval describes the range where an estimate produced by a respective ensemble or model trained on a repeat of the experiment may be found with the probability of the confidence level. For hybrid, it represents the range where any single model trained on a repeat of the experiment may be found with the probability of the confidence level. By definition, confidence intervals obtained using hybrid are at least as wide as those for ensemble. hybrid offers the correct interpretation if the goal of the analysis is to assess the result of a single, unspecified, model.
hybrid is generally computationally less expensive then ensemble, which in turn is somewhat less expensive than model.
A non-default detail_level parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="ensemble", "model_performance"="hybrid").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, prediction_data and confusion_matrix.
If results are computed from 10 samples or fewer, ensemble is automatically used. This prevents issues where evaluation steps do not have a required minimum number of samples for hybrid or model.

estimation_type (*optional*) Sets the type of estimation that should be possible. This has the following options:

- point: Point estimates.
- bias_correction or bc: Bias-corrected estimates. A bias-corrected estimate is computed from (at least) 20 point estimates, and familiar may bootstrap the data to create them.
- bootstrap_confidence_interval or bci (default): Bias-corrected estimates with bootstrap confidence intervals (Efron and Hastie, 2016). The number of point estimates required depends on the confidence_level parameter, and familiar may bootstrap the data to create them.

As with detail_level, a non-default estimation_type parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"="bci", "model_performance"="point").
This parameter can be set for the following data elements: auc_data, decision_curve_analyis, model_performance, permutation_vimp, ice_data, and prediction_data.

aggregate_results (*optional*) Flag that signifies whether results should be aggregated during evaluation. If estimation_type is bias_correction or bc, aggregation leads to a single bias-corrected estimate. If estimation_type is bootstrap_confidence_interval or bci, aggregation leads to a single bias-corrected estimate with lower and upper boundaries of the confidence interval. This has no effect if estimation_type is point.

The default value is equal to TRUE except when assessing metrics to assess model performance, as the default violin plot requires underlying data.
As with detail_level and estimation_type, a non-default aggregate_results parameter can be specified for separate evaluation steps by providing a parameter value in a named list with data elements, e.g. list("auc_data"=TRUE, , "model_performance"=FALSE). This parameter exists for the same elements as estimation_type.

confidence_level (*optional*) Numeric value for the level at which confidence intervals are determined. In the case bootstraps are used to determine the confidence intervals bootstrap estimation, familiar uses the rule of thumb $n = 20/ci.level$ to determine the number of required bootstraps.
The default value is 0.95.

bootstrap_ci_method (*optional*) Method used to determine bootstrap confidence intervals (Efron and Hastie, 2016). The following methods are implemented:

- percentile (default): Confidence intervals obtained using the percentile method.
- bc: Bias-corrected confidence intervals.

Note that the standard method is not implemented because this method is often not suitable due to non-normal distributions. The bias-corrected and accelerated (BCa) method is not implemented yet.

shap_tolerance (*optional*) Relative tolerance for convergence of SHAP values. The tolerance is scaled with the range in SHAP values.
The default value is 0.05.

shap_max_iterations (*optional*) Maximum iterations for convergence of SHAP values.
The default value is 1000 iterations.

shap_phi_0 (*optional*) Value(s) to use for computing marginal contributions. By default, the average predicted value in the population is used. For multinomial and survival outcomes, this parameter requires a value for each class and evaluation time (evaluation_times), respectively.

feature_cluster_method (*optional*) Method used to perform clustering of features. The same methods as for the cluster_method configuration parameter are available: none, hclust, agnes, diana and pam.
The value for the cluster_method configuration parameter is used by default. When generating clusters for the purpose of determining mutual correlation and ordering feature expressions, none is ignored and hclust is used instead.

feature_linkage_method (*optional*) Method used for agglomerative clustering with hclust and agnes. Linkage determines how features are sequentially combined into clusters based on distance. The methods are shared with the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.
The value for the cluster_linkage_method configuration parameters is used by default.

feature_cluster_cut_method (*optional*) Method used to divide features into separate clusters. The available methods are the same as for the cluster_cut_method

configuration parameter: silhouette, fixed_cut and dynamic_cut.
silhouette is available for all cluster methods, but fixed_cut only applies to methods that create hierarchical trees (hclust, agnes and diana).
dynamic_cut requires the dynamicTreeCut package and can only be used with agnes and hclust.
The value for the cluster_cut_method configuration parameter is used by default.

feature_similarity_metric (*optional*) Metric to determine pairwise similarity between features. Similarity is computed in the same manner as for clustering, and feature_similarity_metric therefore has the same options as cluster_similarity_metric: mcfadden_r2, cox_snell_r2, nagelkerke_r2, mutual_information, spearman, kendall and pearson.
The value used for the cluster_similarity_metric configuration parameter is used by default.

feature_similarity_threshold (*optional*) The threshold level for pair-wise similarity that is required to form feature clusters with the fixed_cut method.
This threshold functions in the same manner as the one defined using the cluster_similarity_threshold parameter.
By default, the value for the cluster_similarity_threshold configuration parameter is used.
Unlike for cluster_similarity_threshold, more than one value can be supplied here.

sample_cluster_method (*optional*) The method used to perform clustering based on distance between samples. These are the same methods as for the cluster_method configuration parameter: hclust, agnes, diana and pam.
The value for the cluster_method configuration parameter is used by default. When generating clusters for the purpose of ordering samples in feature expressions, none is ignored and hclust is used instead.

sample_linkage_method (*optional*) The method used for agglomerative clustering in hclust and agnes. These are the same methods as for the cluster_linkage_method configuration parameter: average, single, complete, weighted, and ward.
The value for the cluster_linkage_method configuration parameters is used by default.

sample_similarity_metric (*optional*) Metric to determine pairwise similarity between samples. Similarity is computed in the same manner as for clustering, but sample_similarity_metric has different options that are better suited to computing distance between samples instead of between features. The following metrics are available.

- gower (default): compute Gower's distance between samples. By default, Gower's distance is computed based on winsorised data to reduce the effect of outliers (see below).
- euclidean: compute the Euclidean distance between samples.

The underlying feature data for numerical features is scaled to the $[0, 1]$ range using the feature values across the samples. The normalisation parameters required can optionally be computed from feature data with the outer 5% (on both sides) of feature values trimmed or winsorised. To do

so append _trim (trimming) or _winsor (winsorising) to the metric name. This reduces the effect of outliers somewhat.
Regardless of metric, all categorical features are handled as for the Gower's distance: distance is 0 if the values in a pair of samples match, and 1 if they do not.

eval_aggregation_method (*optional*) Method for aggregating variable importances for the purpose of evaluation. Variable importances are determined both during initial variable importance computation steps and after training the model. Both types are evaluated after training the model, but only the initial variable importances are evaluated at run-time.
See the documentation for the vimp_aggregation_method argument for information concerning the different methods available.

eval_aggregation_rank_threshold (*optional*) The threshold used to define the subset of highly important features during evaluation.
See the documentation for the vimp_aggregation_rank_threshold argument for more information.

eval_icc_type (*optional*) String indicating the type of intraclass correlation coefficient (1, 2 or 3) that should be used to compute robustness for features in repeated measurements during the evaluation of univariate importance. These types correspond to the types in Shrout and Fleiss (1979). The default value is 1.

stratification_method (*optional*) Method for determining the stratification threshold for creating survival groups. The actual, model-dependent, threshold value is obtained from the development data, and can afterwards be used to perform stratification on validation data.
The following stratification methods are available:

- median (default): The median predicted value in the development cohort is used to stratify the samples into two risk groups. For predicted outcome values that build a continuous spectrum, the two risk groups in the development cohort will be roughly equal in size.
- mean: The mean predicted value in the development cohort is used to stratify the samples into two risk groups.
- mean_trim: As mean, but based on the set of predicted values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- mean_winsor: As mean, but based on the set of predicted values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- fixed: Samples are stratified based on the sample quantiles of the predicted values. These quantiles are defined using the stratification_threshold parameter.
- optimised: Use maximally selected rank statistics to determine the optimal threshold (Lausen and Schumacher, 1992; Hothorn et al., 2003) to stratify samples into two optimally separated risk groups.

One or more stratification methods can be selected simultaneously.
This parameter is only relevant for survival outcomes.

`stratification_threshold` (*optional*) Numeric value(s) signifying the sample quantiles for stratification using the `fixed` method. The number of risk groups will be the number of values +1.
The default value is `c(1/3, 2/3)`, which will yield two thresholds that divide samples into three equally sized groups. If `fixed` is not among the selected stratification methods, this parameter is ignored.
This parameter is only relevant for `survival` outcomes.

`time_max` (*optional*) Time point which is used as the benchmark for e.g. cumulative risks generated by random forest, or the cutoff for Uno's concordance index.
If `time_max` is not provided, but `evaluation_times` is, the largest value of `evaluation_times` is used. If both are not provided, `time_max` is set to the 98th percentile of the distribution of survival times for samples with an event in the development data set.
This parameter is only relevant for `survival` outcomes.

`evaluation_times` (*optional*) One or more time points that are used for assessing calibration in survival problems. This is done as expected and observed survival probabilities depend on time.
If unset, `evaluation_times` will be equal to `time_max`.
This parameter is only relevant for `survival` outcomes.

`dynamic_model_loading` (*optional*) Enables dynamic loading of models during the evaluation process, if `TRUE`. Defaults to `FALSE`. Dynamic loading of models may reduce the overall memory footprint, at the cost of increased disk or network IO. Models can only be dynamically loaded if they are found at an accessible disk or network location. Setting this parameter to `TRUE` may help if parallel processing causes out-of-memory issues during evaluation.

`parallel_evaluation` (*optional*) Enable parallel processing for hyperparameter optimisation. Defaults to `TRUE`. When set to `FALSE`, this will disable the use of parallel processing while performing optimisation, regardless of the settings of the `parallel` parameter. The parameter moreover specifies whether parallelisation takes place within the evaluation process steps (`inner`, default), or in an outer loop ( `outer`) over learners, data subsamples, etc.
`parallel_evaluation` is ignored if `parallel=FALSE`.

## Value

Nothing. All output is written to the experiment directory. If the experiment directory is in a temporary location, a list with all familiarModel, familiarEnsemble, familiarData and familiarCollection objects will be returned.

## References

1. Shrout, P. E. & Fleiss, J. L. Intraclass correlations: uses in assessing rater reliability. Psychol. Bull. 86, 420–428 (1979).
2. Koo, T. K. & Li, M. Y. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. J. Chiropr. Med. 15, 155–163 (2016).

3. Yeo, I. & Johnson, R. A. A new family of power transformations to improve normality or symmetry. Biometrika 87, 954–959 (2000).

4. Box, G. E. P. & Cox, D. R. An analysis of transformations. J. R. Stat. Soc. Series B Stat. Methodol. 26, 211–252 (1964).

5. Raymaekers, J., Rousseeuw, P. J. Transforming variables to central normality. Mach Learn. (2021).

6. Zwanenburg, A., & Löck, S. (2026). Location and scale-invariant power transformations for transforming data to normality. Machine Learning, 115(3), 34.

7. Park, M. Y., Hastie, T. & Tibshirani, R. Averaged gene expressions for regression. Biostatistics 8, 212–227 (2007).

8. Tolosi, L. & Lengauer, T. Classification with correlated features: unreliability of feature ranking and solutions. Bioinformatics 27, 1986–1994 (2011).

9. Johnson, W. E., Li, C. & Rabinovic, A. Adjusting batch effects in microarray expression data using empirical Bayes methods. Biostatistics 8, 118–127 (2007)

10. Kaufman, L. & Rousseeuw, P. J. Finding groups in data: an introduction to cluster analysis. (John Wiley & Sons, 2009).

11. Muellner, D. fastcluster: fast hierarchical, agglomerative clustering routines for R and Python. J. Stat. Softw. 53, 1–18 (2013).

12. Rousseeuw, P. J. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. 20, 53–65 (1987).

13. Langfelder, P., Zhang, B. & Horvath, S. Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R. Bioinformatics 24, 719–720 (2008).

14. McFadden, D. Conditional logit analysis of qualitative choice behavior. in Frontiers in Econometrics (ed. Zarembka, P.) 105–142 (Academic Press, 1974).

15. Cox, D. R. & Snell, E. J. Analysis of binary data. (Chapman and Hall, 1989).

16. Nagelkerke, N. J. D. A note on a general definition of the coefficient of determination. Biometrika 78, 691–692 (1991).

17. Meinshausen, N. & Buehlmann, P. Stability selection. J. R. Stat. Soc. Series B Stat. Methodol. 72, 417–473 (2010).

18. Haury, A.-C., Gestraud, P. & Vert, J.-P. The influence of feature selection methods on accuracy, stability and interpretability of molecular signatures. PLoS One 6, e28210 (2011).

19. Wald, R., Khoshgoftaar, T. M., Dittman, D., Awada, W. & Napolitano,A. An extensive comparison of feature ranking aggregation techniques in bioinformatics. in 2012 IEEE 13th International Conference on Information Reuse Integration (IRI) 377–384 (2012).

20. Hutter, F., Hoos, H. H. & Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. in Learning and Intelligent Optimization (ed. Coello, C. A. C.) 6683, 507–523 (Springer Berlin Heidelberg, 2011).

21. Shahriari, B., Swersky, K., Wang, Z., Adams, R. P. & de Freitas, N. Taking the Human Out of the Loop: A Review of Bayesian Optimization. Proc. IEEE 104, 148–175 (2016)

22. Srinivas, N., Krause, A., Kakade, S. M. & Seeger, M. W. Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting. IEEE Trans. Inf. Theory 58, 3250–3265 (2012)

23. Kaufmann, E., Cappé, O. & Garivier, A. On Bayesian upper confidence bounds for bandit problems. in Artificial intelligence and statistics 592–600 (2012).
24. Jamieson, K. & Talwalkar, A. Non-stochastic Best Arm Identification and Hyperparameter Optimization. in Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (eds. Gretton, A. & Robert, C. C.) vol. 51 240–248 (PMLR, 2016).
25. Gramacy, R. B. laGP: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in R. Journal of Statistical Software 72, 1–46 (2016)
26. Sparapani, R., Spanbauer, C. & McCulloch, R. Nonparametric Machine Learning and Efficient Computation with Bayesian Additive Regression Trees: The BART R Package. Journal of Statistical Software 97, 1–66 (2021)
27. Davison, A. C. & Hinkley, D. V. Bootstrap methods and their application. (Cambridge University Press, 1997).
28. Efron, B. & Hastie, T. Computer Age Statistical Inference. (Cambridge University Press, 2016).
29. Lausen, B. & Schumacher, M. Maximally Selected Rank Statistics. Biometrics 48, 73 (1992).
30. Hothorn, T. & Lausen, B. On the exact distribution of maximally selected rank statistics. Comput. Stat. Data Anal. 43, 121–137 (2003).

---

`theme_familiar` *Familiar ggplot2 theme*

---

## Description

This is the default theme used for plots created by familiar. The theme uses `ggplot2::theme_light` as the base template.

## Usage

```
theme_familiar(
  base_size = 10,
  base_family = "",
  base_line_size = 0.5,
  base_rect_size = 0.5
)
```

## Arguments

| | |
|---|---|
| `base_size` | Base font size in points. Size of other plot text elements is based off this. |
| `base_family` | Font family used for text elements. |
| `base_line_size` | Base size for line elements, in points. |
| `base_rect_size` | Base size for rectangular elements, in points. |

## Value

A complete plotting theme.

`train_familiar` *Create models using end-to-end machine learning*

## Description

Train models using familiar. Evaluation is not performed.

## Usage

```
train_familiar(
  formula = NULL,
  data = NULL,
  experiment_data = NULL,
  cl = NULL,
  experimental_design = "fs+mb",
  learner = NULL,
  hyperparameter = NULL,
  verbose = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `formula` | An R formula. The formula can only contain feature names and dot (`.`). The `*` and `+1` operators are not supported as these refer to columns that are not present in the data set.<br>Use of the formula interface is optional. |
| `data` | A `data.table` object, a `data.frame` object, list containing multiple `data.table` or `data.frame` objects, or paths to data files.<br>`data` should be provided if no file paths are provided to the `data_files` argument. If both are provided, only `data` will be used.<br>All data is expected to be in wide format, and ideally has a sample identifier (see `sample_id_column`), batch identifier (see `cohort_column`) and outcome columns (see `outcome_column`).<br>In case paths are provided, the data should be stored as `csv`, `rds` or `RData` files. See documentation for the `data_files` argument for more information. |
| `experiment_data` | Experimental data may provided in the form of the output of `precompute_data_assignment`, `precompute_feature_info` or `precompute_vimp`. This allows for warm-starting experiments. |
| `cl` | Cluster created using the `parallel` package. This cluster is then used to speed up computation through parallelisation. When a cluster is not provided, parallelisation is performed by setting up a cluster on the local machine.<br>This parameter has no effect if the `parallel` argument is set to `FALSE`. |

`experimental_design`

(**required**) Defines what the experiment looks like, e.g. `cv(bt(fs,20)+mb,3,2)` for 2 times repeated 3-fold cross-validation with nested variable importance computation on 20 bootstraps and model-building. The basic workflow components are:

- `fs`: (optional) variable importance computation step. If not explicitly declared, feature selection will be done just in time for hyperparameter optimisation.
- `mb`: (required) model building step.
- `ev`: (optional) external validation. Setting this is not required for `train_familiar`, but if validation batches or cohorts are present in the dataset (`data`), these should be indicated in the `validation_batch_id` argument.

The different components are linked using `+`.

Different subsampling methods can be used in conjunction with the basic workflow components:

- `bs(x,n)`: (stratified) .632 bootstrap, with `n` the number of bootstraps. In contrast to `bt`, feature pre-processing parameters and hyperparameter optimisation are conducted on individual bootstraps.
- `bt(x,n)`: (stratified) .632 bootstrap, with `n` the number of bootstraps. Unlike `bs` and other subsampling methods, no separate pre-processing parameters or optimised hyperparameters will be determined for each bootstrap.
- `cv(x,n,p)`: (stratified) `n`-fold cross-validation, repeated `p` times. Pre-processing parameters are determined for each iteration.
- `lv(x)`: leave-one-out-cross-validation. Pre-processing parameters are determined for each iteration.
- `ip(x)`: imbalance partitioning for addressing class imbalances on the data set. Pre-processing parameters are determined for each partition. The number of partitions generated depends on the imbalance correction method (see the `imbalance_correction_method` parameter).

As shown in the example above, sampling algorithms can be nested.

The simplest valid experimental design is `fs+mb`. This is the default in `train_familiar`, and will create one model for each variable importance method in `vimp_method`. To create more models, a subsampling method should be introduced, e.g. `bs(fs+mb,20)` to create 20 models based on bootstraps of the data.

This argument is ignored if the `experiment_data` argument is set.

`learner`

(**required**) Name of the learner used to develop a model. A sizeable number learners is supported in `familiar`. Please see the vignette on learners for more information concerning the available learners. Unlike the `summon_familiar` function, `train_familiar` only allows for a single learner.

`hyperparameter`

(*optional*) List, or nested list containing hyperparameters for learners. If a nested list is provided, each sublist should have the name of the learner method it corresponds to, with list elements being named after the intended hyperparameter, e.g. `"glm_logistic"=list("sign_size"=3)`

All learners have hyperparameters. Please refer to the vignette on learners for more details. If no parameters are provided, sequential model-based optimisation is used to determine optimal hyperparameters.

Hyperparameters provided by the user are never optimised. However, if more than one value is provided for a single hyperparameter, optimisation will be conducted using these values.

`verbose` Indicates verbosity of the results. Default is TRUE, and all messages and warnings are returned.

`...` Arguments passed on to `.parse_experiment_settings`, `.parse_setup_settings`, `.parse_preprocessing_settings`, `.parse_variable_importance_settings`, `.parse_model_development_settings`, `.parse_hyperparameter_optimisation_settings`

`batch_id_column` (**recommended**) Name of the column containing batch or cohort identifiers. This parameter is required if more than one dataset is provided, or if external validation is performed.
In familiar any row of data is organised by four identifiers:

- The batch identifier `batch_id_column`: This denotes the group to which a set of samples belongs, e.g. patients from a single study, samples measured in a batch, etc. The batch identifier is used for batch normalisation, as well as selection of development and validation datasets.
- The sample identifier `sample_id_column`: This denotes the sample level, e.g. data from a single individual. Subsets of data, e.g. bootstraps or cross-validation folds, are created at this level.
- The series identifier `series_id_column`: Indicates measurements on a single sample that may not share the same outcome value, e.g. a time series, or the number of cells in a view.
- The repetition identifier: Indicates repeated measurements in a single series where any feature values may differ, but the outcome does not. Repetition identifiers are always implicitly set when multiple entries for the same series of the same sample in the same batch that share the same outcome are encountered.

`sample_id_column` (**recommended**) Name of the column containing sample or subject identifiers. See `batch_id_column` above for more details.
If unset, every row will be identified as a single sample.

`series_id_column` (*optional*) Name of the column containing series identifiers, which distinguish between measurements that are part of a series for a single sample. See `batch_id_column` above for more details.
If unset, rows which share the same batch and sample identifiers but have a different outcome are assigned unique series identifiers.

`development_batch_id` (*optional*) One or more batch or cohort identifiers to constitute data sets for development. Defaults to all, or all minus the identifiers in `validation_batch_id` for external validation. Required if external validation is performed and `validation_batch_id` is not provided.

`validation_batch_id` (*optional*) One or more batch or cohort identifiers to constitute data sets for external validation. Defaults to all data sets except those in `development_batch_id` for external validation, or none if not. Required if `development_batch_id` is not provided.

`outcome_name` (*optional*) Name of the modelled outcome. This name will be used in figures created by `familiar`.

If not set, the column name in `outcome_column` will be used for `binomial`, `multinomial`, and `continuous` outcomes. For other outcomes (`survival` and `competing_risk`) no default is used.

`outcome_column` (**recommended**) Name of the column containing the outcome of interest. May be identified from a formula, if a formula is provided as an argument. Otherwise an error is raised. Note that `survival` and `competing_risk` outcome type outcomes require two columns that indicate the time-to-event or the time of last follow-up and the event status.

`outcome_type` (**recommended**) Type of outcome found in the outcome column. The outcome type determines many aspects of the overall process, e.g. the available variable importance methods and learners, but also the type of assessments that can be conducted to evaluate the resulting models. Implemented outcome types are:

- `binomial`: categorical outcome with 2 levels.
- `multinomial`: categorical outcome with 2 or more levels.
- `continuous`: general continuous numeric outcomes.
- `survival`: survival outcome for time-to-event data.

If not provided, the algorithm will attempt to obtain outcome_type from contents of the outcome column. This may lead to unexpected results, and we therefore advise to provide this information manually.
Note that `competing_risk` survival analysis are not fully supported, and is currently not a valid choice for `outcome_type`. The `count` outcome type was deprecated in version 2.0.0, and superseded by `continuous`.

`class_levels` (*optional*) Class levels for `binomial` or `multinomial` outcomes. This argument can be used to specify the ordering of levels for categorical outcomes. These class levels must exactly match the levels present in the outcome column.

`event_indicator` (**recommended**) Indicator for events in `survival` and `competing_risk` analyses. `familiar` will automatically recognise `1`, `true`, `t`, `y` and `yes` as event indicators, including different capitalisations. If this parameter is set, it replaces the default values.

`censoring_indicator` (**recommended**) Indicator for right-censoring in `survival` and `competing_risk` analyses. `familiar` will automatically recognise `0`, `false`, `f`, `n`, `no` as censoring indicators, including different capitalisations. If this parameter is set, it replaces the default values.

`competing_risk_indicator` (**recommended**) Indicator for competing risks in `competing_risk` analyses. There are no default values, and if unset, all values other than those specified by the `event_indicator` and `censoring_indicator` parameters are considered to indicate competing risks.

`signature` (*optional*) One or more names of feature columns that are considered part of a specific signature. Features specified here will always be used for modelling. Ranking of variable importances has no effect for these features.

`novelty_features` (*optional*) One or more names of feature columns that should be included for the purpose of novelty detection.

`exclude_features` (*optional*) Feature columns that will be removed from the data set. Cannot overlap with features in `signature`, `novelty_features`

or include_features.

include_features (*optional*) Feature columns that are specifically included in the data set. By default all features are included. Cannot overlap with exclude_features, but may overlap signature. Features in signature and novelty_features are always included. If both exclude_features and include_features are provided, include_features takes precedence, provided that there is no overlap between the two.

reference_method (*optional*) Method used to set reference levels for categorical features. There are several options:

- auto (default): Categorical features that are not explicitly set by the user, i.e. columns containing boolean values or characters, use the most frequent level as reference. Categorical features that are explicitly set, i.e. as factors, are used as is.
- always: Both automatically detected and user-specified categorical features have the reference level set to the most frequent level. Ordinal features are not altered, but are used as is.
- never: User-specified categorical features are used as is. Automatically detected categorical features are simply sorted, and the first level is then used as the reference level. This was the behaviour prior to familiar version 1.3.0.

imbalance_correction_method (*optional*) Type of method used to address class imbalances. Available options are:

- full_undersampling (default): All data will be used in an ensemble fashion. The full minority class will appear in each partition, but majority classes are undersampled until all data have been used.
- random_undersampling: Randomly undersamples majority classes. This is useful in cases where full undersampling would lead to the formation of many models due major overrepresentation of the largest class.

This parameter is only used in combination with imbalance partitioning in the experimental design, and ip should therefore appear in the string that defines the design.

imbalance_n_partitions (*optional*) Number of times random undersampling should be repeated. 10 undersampled subsets with balanced classes are formed by default.

iteration_seed (*optional*) Integer seed used in sampling algorithms specified by the experimental_design argument. This allows for creating the same sample assignments across different experiments – of course provided that the same dataset is used. By default a random seed is used.

parallel (*optional*) Enable parallel processing. Defaults to TRUE. When set to FALSE, this disables all parallel processing, regardless of specific parameters such as parallel_preprocessing. However, when parallel is TRUE, parallel processing of different parts of the workflow can be disabled by setting respective flags to FALSE.

parallel_nr_cores (*optional*) Number of cores available for parallelisation. Defaults to 2. This setting does nothing if parallelisation is disabled.

restart_cluster (*optional*) Restart nodes used for parallel computing to free up memory prior to starting a parallel process. Note that it does take time to set up the clusters. Therefore setting this argument to TRUE may impact processing speed. This argument is ignored if parallel is FALSE or the cluster was initialised outside of familiar. Default is FALSE, which causes the clusters to be initialised only once.

cluster_type (*optional*) Selection of the cluster type for parallel processing. Available types are the ones supported by the parallel package that is part of the base R distribution: psock (default), fork, mpi, nws, sock. In addition, none is available, which also disables parallel processing.

backend_type (*optional*) Selection of the backend for distributing copies of the data. This backend ensures that only a single master copy is kept in memory. This limits memory usage during parallel processing.
Several backend options are available, notably socket_server, and none (default). socket_server is based on the callr package and R sockets, comes with familiar and is available for any OS. none uses the package environment of familiar to store data, and is available for any OS. However, none requires copying of data to any parallel process, and has a larger memory footprint.

server_port (*optional*) Integer indicating the port on which the socket server or RServe process should communicate. Defaults to port 6311. Note that ports 0 to 1024 and 49152 to 65535 cannot be used.

feature_max_fraction_missing (*optional*) Numeric value between 0.0 and 0.95 that determines the meximum fraction of missing values that still allows a feature to be included in the data set. All features with a missing value fraction over this threshold are not processed further. The default value is 0.30.

sample_max_fraction_missing (*optional*) Numeric value between 0.0 and 0.95 that determines the maximum fraction of missing values that still allows a sample to be included in the data set. All samples with a missing value fraction over this threshold are excluded and not processed further. The default value is 0.30.

filter_method (*optional*) One or methods used to reduce dimensionality of the data set by removing irrelevant or poorly reproducible features.
Several method are available:

- none (default): None of the features will be filtered.
- low_variance: Features with a variance below the low_var_minimum_variance_threshold are filtered. This can be useful to filter, for example, genes that are not differentially expressed.
- univariate_test: Features undergo a univariate regression using an outcome-appropriate regression model. The p-value of the model coefficient is collected. Features with coefficient p-value above the univariate_test_threshold are subsequently filtered.
- robustness: Features that are not sufficiently robust according to the intraclass correlation coefficient are filtered. Use of this method requires that repeated measurements are present in the data set, i.e. there should be entries for which the sample and cohort identifiers are the same.

More than one method can be used simultaneously. Features with singular values are always filtered, as these do not contain information.

`univariate_test_threshold` (*optional*) Numeric value between `1.0` and `0.0` that determines which features are irrelevant and will be filtered by the `univariate_test`. p-values are compared to this threshold. All features with values above the threshold are filtered. The default value is `0.20`.

`univariate_test_threshold_metric` (*optional*) Metric used with the to compare the `univariate_test_threshold` against. The following metric can be chosen:

- `p_value` (default): The unadjusted p-value of each feature is used for to filter features.

`univariate_test_max_feature_set_size` (*optional*) Maximum size of the feature set after the univariate test. P or q values of features are compared against the threshold, but if the resulting data set would be larger than this setting, only the most relevant features up to the desired feature set size are selected.
The default value is `NULL`, which causes features to be filtered based on their relevance only.

`low_var_minimum_variance_threshold` (required, if used) Numeric value that determines which features will be filtered by the `low_variance` method. The variance of each feature is computed and compared to the threshold. If it is below the threshold, the feature is removed.
This parameter has no default value and should be set if `low_variance` is used.

`low_var_max_feature_set_size` (*optional*) Maximum size of the feature set after filtering features with a low variance. All features are first compared against `low_var_minimum_variance_threshold`. If the resulting feature set would be larger than specified, only the most strongly varying features will be selected, up to the desired size of the feature set.
The default value is `NULL`, which causes features to be filtered based on their variance only.

`robustness_icc_type` (*optional*) String indicating the type of intraclass correlation coefficient (`1`, `2` or `3`) that should be used to compute robustness for features in repeated measurements. These types correspond to the types in Shrout and Fleiss (1979). The default value is `1`.

`robustness_threshold_metric` (*optional*) String indicating which specific intraclass correlation coefficient (ICC) metric should be used to filter features. This should be one of:

- `icc`: The estimated ICC value itself.
- `icc_low` (default): The estimated lower limit of the 95% confidence interval of the ICC, as suggested by Koo and Li (2016).
- `icc_panel`: The estimated ICC value over the panel average, i.e. the ICC that would be obtained if all repeated measurements were averaged.
- `icc_panel_low`: The estimated lower limit of the 95% confidence interval of the panel ICC.

robustness_threshold_value (*optional*) The intraclass correlation coefficient value that is as threshold. The default value is 0.70.

transformation_method (*optional*) The transformation method used to change the distribution of the data to be more normal-like. The following methods are available:

- none: This disables transformation of features.
- yeo_johnson: Transformation using the location and scale invariant version of the Yeo-Johnson transformation (Yeo and Johnson, 2000; Zwanenburg and Löck, 2026).
- yeo_johnson_robust (default): A robust version of yeo_johnson. This method is less sensitive to outliers.
- yeo_johnson_conventional: As yeo_johnson, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Yeo and Johnson (2001).
- box_cox: Transformation using the location and scale invariant version of the Box-Cox transformation (Box and Cox, 1964; Zwanenburg and Löck, 2026).
- box_cox_robust: A robust version of yeo_johnson. This method is less sensitive to outliers.
- box_cox_conventional: As box_cox, but without optimisation of location and scale parameters. This method is equivalent to the original transformation proposed by Box and Cox (1964). This method requires strictly positive feature values.

Transformation requires the power.transform package. Only features that contain numerical data are transformed. Transformation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets.

transformation_optimisation_criterion (*optional*) Transformation parameters are optimised using a criterion, conventionally maximum-likelihood-estimation. power.transform implements multiple optimisation criteria, of which the following are available:

- mle (default): Optimisation using maximum likelihood estimation.
- cramer_von_mises: Optimisation using the Cramér-von Mises criterion. Zwanenburg and Löck (2026) found that this criterion was relatively robust against outliers.

transformation_gof_test_p_value (*optional*) Not all transformations will lead to features that are roughly normally distributed. Zwanenburg and Löck (2026) established a empirical goodness-of-fit test for central normality. This parameter sets the significance for rejecting the null-hypothesis that a feature distribution is centrally normal. When the null-hypothesis is rejected, no transformation is performed. The default value is NULL, which disables the test.

normalisation_method (*optional*) The normalisation method used to improve the comparability between numerical features that may have very different scales. The following normalisation methods can be chosen:

- none: This disables feature normalisation.

- standardisation: Features are normalised by subtraction of their mean values and division by their standard deviations. This causes every feature to be have a center value of 0.0 and standard deviation of 1.0.
- standardisation_trim: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- standardisation_winsor: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- standardisation_robust (default): A robust version of standardisation that relies on computing Huber's M-estimators for location and scale.
- normalisation: Features are normalised by subtraction of their minimum values and division by their ranges. This maps all feature values to a $[0, 1]$ interval.
- normalisation_trim: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- normalisation_winsor: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- quantile: Features are normalised by subtraction of their median values and division by their interquartile range.
- mean_centering: Features are centered by substracting the mean, but do not undergo rescaling.

Only features that contain numerical data are normalised. Normalisation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets.

batch_normalisation_method (*optional*) The method used for batch normalisation. Available methods are:

- none (default): This disables batch normalisation of features.
- standardisation: Features within each batch are normalised by subtraction of the mean value and division by the standard deviation in each batch.
- standardisation_trim: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- standardisation_winsor: As standardisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- standardisation_robust: A robust version of standardisation that relies on computing Huber's M-estimators for location and scale within each batch.
- normalisation: Features within each batch are normalised by subtraction of their minimum values and division by their range in each batch. This maps all feature values in each batch to a $[0, 1]$ interval.

- normalisation_trim: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are discarded. This reduces the effect of outliers.
- normalisation_winsor: As normalisation, but based on the set of feature values where the 5% lowest and 5% highest values are winsorised. This reduces the effect of outliers.
- quantile: Features in each batch are normalised by subtraction of the median value and division by the interquartile range of each batch.
- mean_centering: Features in each batch are centered on 0.0 by substracting the mean value in each batch, but are not rescaled.
- combat_parametric: Batch adjustments using parametric empirical Bayes (Johnson et al, 2007). combat_p leads to the same method.
- combat_non_parametric: Batch adjustments using non-parametric empirical Bayes (Johnson et al, 2007). combat_np and combat lead to the same method. Note that we reduced complexity from O($n^2$) to O($n$) by only computing batch adjustment parameters for each feature on a subset of 50 randomly selected features, instead of all features.

Only features that contain numerical data are normalised using batch normalisation. Batch normalisation parameters obtained in development data are stored within featureInfo objects for later use with validation data sets, in case the validation data is from the same batch.
If validation data contains data from unknown batches, normalisation parameters are separately determined for these batches.
Note that for both empirical Bayes methods, the batch effect is assumed to produce results across the features. This is often true for things such as gene expressions, but the assumption may not hold generally.
When performing batch normalisation, it is moreover important to check that differences between batches or cohorts are not related to the studied endpoint.

imputation_method (*optional*) Method used for imputing missing feature values. Two methods are implemented:

- simple: Simple replacement of a missing value by the median value (for numeric features) or the modal value (for categorical features).
- lasso: Imputation of missing value by lasso regression (using glmnet) based on information contained in other features.

simple imputation precedes lasso imputation to ensure that any missing values in predictors required for lasso regression are resolved. The lasso estimate is then used to replace the missing value.
The default value depends on the number of features in the dataset. If the number is lower than 100, lasso is used by default, and simple otherwise.
Only single imputation is performed. Imputation models and parameters are stored within featureInfo objects for later use with validation data sets.

cluster_method (*optional*) Clustering is performed to identify and replace redundant features, for example those that are highly correlated. Such features do not carry much additional information and may be removed or replaced instead (Park et al., 2007; Tolosi and Lengauer, 2011).

The cluster method determines the algorithm used to form the clusters. The following cluster methods are implemented:

- none: No clustering is performed.
- hclust (default): Hierarchical agglomerative clustering. If the fastcluster package is installed, fastcluster::hclust is used (Muellner 2013), otherwise stats::hclust is used.
- agnes: Hierarchical clustering using agglomerative nesting (Kaufman and Rousseeuw, 1990). This algorithm is similar to hclust, but uses the cluster::agnes implementation.
- diana: Divisive analysis hierarchical clustering. This method uses divisive instead of agglomerative clustering (Kaufman and Rousseeuw, 1990). cluster::diana is used.
- pam: Partioning around medioids. This partitions the data into $k$ clusters around medioids (Kaufman and Rousseeuw, 1990). $k$ is selected using the silhouette metric. pam is implemented using the cluster::pam function.

Clusters and cluster information is stored within featureInfo objects for later use with validation data sets. This enables reproduction of the same clusters as formed in the development data set.

cluster_linkage_method (*optional*) Linkage method used for agglomerative clustering in hclust and agnes. The following linkage methods can be used:

- average (default): Average linkage.
- single: Single linkage.
- complete: Complete linkage.
- weighted: Weighted linkage, also known as McQuitty linkage.
- ward: Linkage using Ward's minimum variance method.

diana and pam do not require a linkage method.

cluster_cut_method (*optional*) The method used to define the actual clusters. The following methods can be used:

- silhouette: Clusters are formed based on the silhouette score (Rousseeuw, 1987). The average silhouette score is computed from 2 to $n$ clusters, with $n$ the number of features. Clusters are only formed if the average silhouette exceeds 0.50, which indicates reasonable evidence for structure. This procedure may be slow if the number of features is large (>100s).
- fixed_cut: Clusters are formed by cutting the hierarchical tree at the point indicated by the cluster_similarity_threshold, e.g. where features in a cluster have an average Spearman correlation of 0.90. fixed_cut is only available for agnes, diana and hclust.
- dynamic_cut: Dynamic cluster formation using the cutting algorithm in the dynamicTreeCut package. This package should be installed to select this option. dynamic_cut can only be used with agnes and hclust.

The default options are silhouette for partioning around medioids (pam) and fixed_cut otherwise.

cluster_similarity_metric (*optional*) Clusters are formed based on feature similarity. All features are compared in a pair-wise fashion to compute similarity, for example correlation. The resulting similarity grid is converted into a distance matrix that is subsequently used for clustering. The following metrics are supported to compute pairwise similarities:

- mutual_information (default): normalised mutual information.
- mcfadden_r2: McFadden's pseudo R-squared (McFadden, 1974).
- cox_snell_r2: Cox and Snell's pseudo R-squared (Cox and Snell, 1989).
- nagelkerke_r2: Nagelkerke's pseudo R-squared (Nagelkerke, 1991).
- spearman: Spearman's rank order correlation.
- kendall: Kendall rank correlation.
- pearson: Pearson product-moment correlation.

The pseudo R-squared metrics can be used to assess similarity between mixed pairs of numeric and categorical features, as these are based on the log-likelihood of regression models. In familiar, the more informative feature is used as the predictor and the other feature as the reponse variable. In numeric-categorical pairs, the numeric feature is considered to be more informative and is thus used as the predictor. In categorical-categorical pairs, the feature with most levels is used as the predictor.

In case any of the classical correlation coefficients (pearson, spearman and kendall) are used with (mixed) categorical features, the categorical features are one-hot encoded and the mean correlation over all resulting pairs is used as similarity.

cluster_similarity_threshold (*optional*) The threshold level for pair-wise similarity that is required to form clusters using fixed_cut. This should be a numerical value between 0.0 and 1.0. Note however, that a reasonable threshold value depends strongly on the similarity metric. The following are the default values used:

- mcfadden_r2 and mutual_information: 0.30
- cox_snell_r2 and nagelkerke_r2: 0.75
- spearman, kendall and pearson: 0.90

Alternatively, if the fixed cut method is not used, this value determines whether any clustering should be performed, because the data may not contain highly similar features. The default values in this situation are:

- mcfadden_r2 and mutual_information: 0.25
- cox_snell_r2 and nagelkerke_r2: 0.40
- spearman, kendall and pearson: 0.70

The threshold value is converted to a distance (1-similarity) prior to cutting hierarchical trees.

cluster_representation_method (*optional*) Method used to determine how the information of co-clustered features is summarised and used to represent the cluster. The following methods can be selected:

- best_predictor (default): The feature with the highest importance according to univariate regression with the outcome is used to represent the cluster.

- `medioid`: The feature closest to the cluster center, i.e. the feature that is most similar to the remaining features in the cluster, is used to represent the feature.
- `mean`: A meta-feature is generated by averaging the feature values for all features in a cluster. This method aligns all features so that all features will be positively correlated prior to averaging. Should a cluster contain one or more categorical features, the `medioid` method will be used instead, as averaging is not possible. Note that if this method is chosen, the `normalisation_method` parameter should be one of `standardisation`, `standardisation_trim`, `standardisation_winsor` or `quantile`.‘

If the `pam` cluster method is selected, only the `medioid` method can be used. In that case 1 medioid is used by default.

`parallel_preprocessing` (*optional*) Enable parallel processing for the preprocessing workflow. Defaults to `TRUE`. When set to `FALSE`, this will disable the use of parallel processing while preprocessing, regardless of the settings of the `parallel` parameter. `parallel_preprocessing` is ignored if `parallel=FALSE`.

`vimp_method` (**required**) Variable importance method. `familiar` implements various variable importance methods. Please refer to the vignette on variable importance methods for more details.

More than one variable importance method can be chosen. The experiment will then repeated for each feature selection method.

Variable importance methods determine the ranking of features. Actual selection of features is done by optimising the signature size model hyperparameter during the hyperparameter optimisation step.

`vimp_method_parameter` (*optional*) List of lists containing parameters for feature selection methods. Each sublist should have the name of the feature selection method it corresponds to.

Most feature selection methods do not have parameters that can be set. Please refer to the vignette on feature selection methods for more details. Note that if the feature selection method is based on a learner (e.g. lasso regression), hyperparameter optimisation may be performed prior to assessing variable importance.

`vimp_aggregation_method` (*optional*) The method used to aggregate variable importances over different data subsets, e.g. bootstraps. The following methods can be selected:

- `none`: Don’t aggregate ranks, but rather aggregate the variable importance scores themselves.
- `mean`: Use the mean rank of a feature over the subsets to determine the aggregated feature rank.
- `median`: Use the median rank of a feature over the subsets to determine the aggregated feature rank.
- `best`: Use the best rank the feature obtained in any subset to determine the aggregated feature rank.
- `worst`: Use the worst rank the feature obtained in any subset to determine the aggregated feature rank.

- stability: Use the frequency of the feature being in the subset of highly ranked features as measure for the aggregated feature rank (Meinshausen and Buehlmann, 2010).
- exponential: Use a rank-weighted frequence of occurrence in the subset of highly ranked features as measure for the aggregated feature rank (Haury et al., 2011).
- borda (default): Use the borda count as measure for the aggregated feature rank (Wald et al., 2012).
- enhanced_borda: Use an occurrence frequency-weighted borda count as measure for the aggregated feature rank (Wald et al., 2012).
- truncated_borda: Use borda count computed only on features within the subset of highly ranked features.
- enhanced_truncated_borda: Apply both the enhanced borda method and the truncated borda method and use the resulting borda count as the aggregated feature rank.

The *feature selection methods* vignette provides additional information.

vimp_aggregation_rank_threshold (*optional*) The threshold used to define the subset of highly important features. If set to NULL, this threshold is determined by maximising the variance in the occurrence value over all features over the subset size. The default value is 5.
This parameter is only relevant for stability, exponential, enhanced_borda, truncated_borda and enhanced_truncated_borda methods.

parallel_vimp (*optional*) Enable parallel processing for variable importance tasks. Defaults to TRUE. When set to FALSE, this will disable the use of parallel processing while performing feature selection, regardless of the settings of the parallel parameter. parallel_vimp is ignored if parallel = FALSE.

novelty_detector (*optional*) Specify the algorithm used for training a novelty detector. This detector can be used to identify out-of-distribution data prospectively.

detector_parameters (*optional*) List lists containing hyperparameters for novelty detectors. Currently not used.

parallel_model_development (*optional*) Enable parallel processing for the model development workflow. Defaults to TRUE. When set to FALSE, this will disable the use of parallel processing while developing models, regardless of the settings of the parallel parameter. parallel_model_development is ignored if parallel=FALSE.

optimisation_bootstraps (*optional*) Number of bootstraps that should be generated from the development data set. During the optimisation procedure one or more of these bootstraps (indicated by smbo_step_bootstraps) are used for model development using different combinations of hyperparameters. The effect of the hyperparameters is then assessed by comparing in-bag and out-of-bag model performance.
The default number of bootstraps is 50. Hyperparameter optimisation may finish before exhausting the set of bootstraps.

optimisation_determine_vimp (*optional*) Logical value that indicates whether variable importance is determined separately for each of the bootstraps cre-

ated during the optimisation process (TRUE) or the applicable results from the variable importance computation step are used (FALSE).
Determining variable importance increases the initial computational overhead. However, it prevents positive biases for the out-of-bag data due to overlap of these data with the development data set used for the feature selection step. In this case, any hyperparameters of the variable importance method are not determined separately for each bootstrap, but those obtained during the variable importance computation step are used instead. In case multiple of such hyperparameter sets could be applicable, the set that will be used is randomly selected for each bootstrap.
This parameter only affects hyperparameter optimisation of learners. The default is TRUE.

smbo_random_initialisation (*optional*) String indicating the initialisation method for the hyperparameter space. Can be one of fixed_subsample (default), fixed, or random. fixed and fixed_subsample first create hyperparameter sets from a range of default values set by familiar. fixed_subsample then randomly draws up to smbo_n_random_sets from the grid. random does not rely upon a fixed grid, and randomly draws up to smbo_n_random_sets hyperparameter sets from the hyperparameter space.

smbo_n_random_sets (*optional*) Number of random or subsampled hyperparameters drawn during the initialisation process. Default: 100. Cannot be smaller than 10. The parameter is not used when smbo_random_initialisation is fixed, as the entire pre-defined grid will be explored.

max_smbo_iterations (*optional*) Maximum number of intensify iterations of the SMBO algorithm. During an intensify iteration a run-off occurs between the current *best* hyperparameter combination and either 10 challenger combination with the highest expected improvement or a set of 20 random combinations.
Run-off with random combinations is used to force exploration of the hyperparameter space, and is performed every second intensify iteration, or if there is no expected improvement for any challenger combination.
If a combination of hyperparameters leads to better performance on the same data than the incumbent *best* set of hyperparameters, it replaces the incumbent set at the end of the intensify iteration.
The default number of intensify iteration is 20. Iterations may be stopped early if the incumbent set of hyperparameters remains the same for smbo_stop_convergent_iterat iterations, or performance improvement is minimal. This behaviour is suppressed during the first 4 iterations to enable the algorithm to explore the hyperparameter space.

smbo_stop_convergent_iterations (*optional*) The number of subsequent convergent SMBO iterations required to stop hyperparameter optimisation early.
An iteration is convergent if the *best* parameter set has not changed or the optimisation score over the 4 most recent iterations has not changed beyond the tolerance level in smbo_stop_tolerance.
The default value is 3.

smbo_stop_tolerance (*optional*) Tolerance for early stopping due to convergent optimisation score.
The default value depends on the square root of the number of samples (at

the series level), and is `0.01` for 100 samples. This value is computed as `0.1 * 1 / sqrt(n_samples)`. The upper limit is `0.0001` for 1M or more samples.

`smbo_time_limit` (*optional*) Time limit (in minutes) for the optimisation process. Optimisation is stopped after this limit is exceeded. Time taken to determine variable importance for the optimisation process (see the `optimisation_determine_vimp` parameter) does not count.

The default is `NULL`, indicating that there is no time limit for the optimisation process. The time limit cannot be less than 1 minute.

`smbo_initial_bootstraps` (*optional*) The number of bootstraps taken from the set of `optimisation_bootstraps` as the bootstraps assessed initially.

The default value is `1`. The value cannot be larger than `optimisation_bootstraps`.

`smbo_step_bootstraps` (*optional*) The number of bootstraps taken from the set of `optimisation_bootstraps` bootstraps as the bootstraps assessed during the steps of each intensify iteration.

The default value is `3`. The value cannot be larger than `optimisation_bootstraps`.

`smbo_intensify_steps` (*optional*) The number of steps in each SMBO intensify iteration. Each step a new set of `smbo_step_bootstraps` bootstraps is drawn and used in the run-off between the incumbent *best* hyperparameter combination and its challengers.

The default value is `5`. Higher numbers allow for a more detailed comparison, but this comes with added computational cost.

`optimisation_metric` (*optional*) One or more metrics used to compute performance scores. See the vignette on performance metrics for the available metrics.

If unset, the following metrics are used by default:

- `auc_roc`: For `binomial` and `multinomial` models.
- `mse`: Mean squared error for `continuous` models.
- `concordance_index`: For `survival` models.

Multiple optimisation metrics can be specified. Actual metric values are converted to an objective value by comparison with a baseline metric value that derives from a trivial model, i.e. majority class for binomial and multinomial outcomes, the median outcome for count and continuous outcomes and a fixed risk or time for survival outcomes.

`optimisation_function` (*optional*) Type of optimisation function used to quantify the performance of a hyperparameter set. Model performance is assessed using the metric(s) specified by `optimisation_metric` on the in-bag (IB) and out-of-bag (OOB) samples of a bootstrap. These values are converted to objective scores with a standardised interval of $[-1.0, 1.0]$. Each pair of objective is subsequently used to compute an optimisation score. The optimisation score across different bootstraps is than aggregated to a summary score. This summary score is used to rank hyperparameter sets, and select the optimal set.

The combination of optimisation score and summary score is determined by the optimisation function indicated by this parameter:

- `validation` or `max_validation` (default): seeks to maximise OOB score.

- balanced: seeks to balance IB and OOB score.
- stronger_balance: similar to balanced, but with stronger penalty for differences between IB and OOB scores.
- validation_minus_sd: seeks to optimise the average OOB score minus its standard deviation.
- validation_25th_percentile: seeks to optimise the 25th percentile of OOB scores, and is conceptually similar to validation_minus_sd.
- model_estimate: seeks to maximise the OOB score estimate predicted by the hyperparameter learner (not available for random search).
- model_estimate_minus_sd: seeks to maximise the OOB score estimate minus its estimated standard deviation, as predicted by the hyperparameter learner (not available for random search).
- model_balanced_estimate: seeks to maximise the estimate of the balanced IB and OOB score. This is similar to the balanced score, and in fact uses a hyperparameter learner to predict said score (not available for random search).
- model_balanced_estimate_minus_sd: seeks to maximise the estimate of the balanced IB and OOB score, minus its estimated standard deviation. This is similar to the balanced score, but takes into account its estimated spread.

Additional detail are provided in the *Learning algorithms and hyperparameter optimisation* vignette.

hyperparameter_learner (*optional*) Any point in the hyperparameter space has a single, scalar, optimisation score value that is *a priori* unknown. During the optimisation process, the algorithm samples from the hyperparameter space by selecting hyperparameter sets and computing the optimisation score value for one or more bootstraps. For each hyperparameter set the resulting values are distributed around the actual value. The learner indicated by hyperparameter_learner is then used to infer optimisation score estimates for unsampled parts of the hyperparameter space.
The following models are available:

- bayesian_additive_regression_trees or bart: Uses Bayesian Additive Regression Trees (Sparapani et al., 2021) for inference. Unlike standard random forests, BART allows for estimating posterior distributions directly and can extrapolate.
- gaussian_process (default): Creates a localised approximate Gaussian process for inference (Gramacy, 2016). This allows for better scaling than deterministic Gaussian Processes.
- random_forest: Creates a random forest for inference. Originally suggested by Hutter et al. (2011). A weakness of random forests is their lack of extrapolation beyond observed values, which limits their usefulness in exploiting promising areas of hyperparameter space.
- random or random_search: Forgoes the use of models to steer optimisation. Instead, a random search is performed.

acquisition_function (*optional*) The acquisition function influences how new hyperparameter sets are selected. The algorithm uses the model learned by

the learner indicated by hyperparameter_learner to search the hyperparameter space for hyperparameter sets that are either likely better than the best known set (*exploitation*) or where there is considerable uncertainty (*exploration*). The acquisition function quantifies this (Shahriari et al., 2016). The following acquisition functions are available, and are described in more detail in the *learner algorithms* vignette:

- improvement_probability: The probability of improvement quantifies the probability that the expected optimisation score for a set is better than the best observed optimisation score
- improvement_empirical_probability: Similar to improvement_probability, but based directly on optimisation scores predicted by the individual decision trees.
- expected_improvement (default): Computes expected improvement.
- upper_confidence_bound: This acquisition function is based on the upper confidence bound of the distribution (Srinivas et al., 2012).
- bayes_upper_confidence_bound: This acquisition function is based on the upper confidence bound of the distribution (Kaufmann et al., 2012).

exploration_method (*optional*) Method used to steer exploration in post-initialisation intensive searching steps. As stated earlier, each SMBO iteration step compares suggested alternative parameter sets with an incumbent **best** set in a series of steps. The exploration method controls how the set of alternative parameter sets is pruned after each step in an iteration. Can be one of the following:

- single_shot (default): The set of alternative parameter sets is not pruned, and each intensification iteration contains only a single intensification step that only uses a single bootstrap. This is the fastest exploration method, but only superficially tests each parameter set.
- successive_halving: The set of alternative parameter sets is pruned by removing the worst performing half of the sets after each step (Jamieson and Talwalkar, 2016).
- stochastic_reject: The set of alternative parameter sets is pruned by comparing the performance of each parameter set with that of the incumbent **best** parameter set using a paired Wilcoxon test based on shared bootstraps. Parameter sets that perform significantly worse, at an alpha level indicated by smbo_stochastic_reject_p_value, are pruned.
- none: The set of alternative parameter sets is not pruned.

smbo_stochastic_reject_p_value (*optional*) The p-value threshold used for the stochastic_reject exploration method.
The default value is 0.05.

parallel_hyperparameter_optimisation (*optional*) Enable parallel processing for hyperparameter optimisation. Defaults to TRUE. When set to FALSE, this will disable the use of parallel processing while performing optimisation, regardless of the settings of the parallel parameter. The parameter moreover specifies whether parallelisation takes place within the optimisation algorithm (inner, default), or in an outer loop ( outer) over learners,

data subsamples, etc.
`parallel_hyperparameter_optimisation` is ignored if `parallel=FALSE`.

## Details

This is a thin wrapper around `summon_familiar`, and functions like it, but automatically skips all evaluation steps. Only a single learner is allowed.

## Value

One or more familiarModel objects.

---

`update_model_dir_path` *Updates model directory path for ensemble objects.*

---

## Description

Updates the model directory path of a `familiarEnsemble` object.

## Usage

```
update_model_dir_path(object, dir_path, ...)

## S4 method for signature 'familiarEnsemble'
update_model_dir_path(object, dir_path)

## S4 method for signature 'ANY'
update_model_dir_path(object, dir_path)
```

## Arguments

| | |
|---|---|
| `object` | A `familiarEnsemble` object, or one or more `familiarModel` objects that will be internally converted to a `familiarEnsemble` object. Paths to such objects can also be provided. |
| `dir_path` | Path to the directory where models are stored. |
| `...` | Unused arguments. |

## Details

Ensemble models created by familiar are often written to a directory on a local drive or network. In such cases, the actual models are detached, and paths to the models are stored instead. When the models are moved from their original location, they can no longer be found and attached to the ensemble. This method allows for pointing to the new directory containing the models.

## Value

A `familiarEnsemble` object.

update_object *Update familiar S4 objects to the most recent version.*

## Description

Provides backward compatibility for familiar objects exported to a file. This mitigates compatibility issues when working with files that become outdated as new versions of familiar are released, e.g. because slots have been removed.

Major version releases (e.g., versions 1.0.0, 2.0.0) introduce breaking changes which can result in objects that cannot be updated.

## Usage

```
update_object(object, ...)

## S4 method for signature 'familiarModel'
update_object(object, ...)

## S4 method for signature 'familiarEnsemble'
update_object(object, ...)

## S4 method for signature 'familiarData'
update_object(object, ...)

## S4 method for signature 'familiarCollection'
update_object(object, ...)

## S4 method for signature 'vimpTable'
update_object(object, ...)

## S4 method for signature 'familiarNoveltyDetector'
update_object(object, ...)

## S4 method for signature 'featureInfo'
update_object(object, ...)

## S4 method for signature 'featureInfoParametersTransformationPowerTransform'
update_object(object, ...)

## S4 method for signature 'experimentData'
update_object(object, ...)

## S4 method for signature 'list'
update_object(object, ...)

## S4 method for signature 'ANY'
update_object(object, ...)
```

### Arguments

| | |
|---|---|
| object | A familiarModel, a familiarEnsemble, a familiarData or familiarCollection object. |
| ... | Unused arguments. |

### Value

An up-to-date version of the respective S4 object.

---

## vcov *Calculate variance-covariance matrix for a model*

---

### Description

Calculate variance-covariance matrix for a model

### Usage

```
vcov(object, ...)

## S4 method for signature 'familiarModel'
vcov(object, ...)
```

### Arguments

| | |
|---|---|
| object | a familiarModel object |
| ... | additional arguments passed to vcov methods for the underlying model, when available. |

### Details

This method extends the vcov S3 method. For some models vcov requires information that is trimmed from the model. In this case a copy of the variance-covariance matrix is stored with the model, and returned.

### Value

Variance-covariance matrix of the model in the familiarModel object, if any.

`vimpTable-class` *Variable importance table*

## Description

A vimpTable object contains information concerning variable importance of one or more features. These objects are created during variable importance computation..

## Details

vimpTable objects exists in various states. These states are generally incremental, i.e. one cannot turn a declustered table into the initial version. Some methods such as aggregation internally do some state reshuffling.

This object replaces the ad-hoc lists with information that were used in versions prior to familiar 1.2.0.

## Slots

`vimp_table` Table containing features with corresponding scores.

`vimp_method` Method used to compute variable importance scores for each feature.

`data_id` Internal identifier for the dataset used to derive the variable importance table.

`run_id` Internal identifier for the specific subset of the dataset used to derive the variable importance table.

`run_table` Run table for the data used to compute variable importances from. Used internally.

`score_aggregation` Method used to aggregate the score of contrasts for each categorical feature, if any,

`encoding_table` Table used to relate categorical features to their contrasts, if any. Not used for all variable importance methods.

`cluster_table` Table used to relate original features with features after clustering. Variable importance is determined after feature processing, which includes clustering.

`invert` Determines whether increasing score corresponds to increasing (`FALSE`) or decreasing rank (`TRUE`). Used internally to determine how ranks should be formed.

`project_id` Identifier of the project that generated the vimpTable object.

`familiar_version` Version of the familiar package used to create this table.

`state` State of the variable importance table. The object can have the following states:

- `initial`: initial state, directly after the variable importance table is filled.
- `decoded`: depending on the variable importance method, the initial variable importance table may contain the scores of individual contrasts for categorical variables. When decoded, data in the `encoding_table` attribute has been used to aggregate scores from all contrasts into a single score for each feature.
- `declustered`: variable importance is determined from fully processed features, which includes clustering. This means that a single feature in the variable importance table may represent multiple original features. When a variable importance table has been declustered, all clusters have been turned into their constituent features.

- `reclustered`: When the table is reclustered, features are replaced by their respective clusters. This is actually used when updating the cluster table to ensure it fits to a local context. This prevents issues when attempting to aggregate or apply variable importance tables in data with different feature preprocessing, and as a result, different clusters.
- `ranked`: The scores have been used to create ranks, with lower ranks indicating better features.
- `aggregated`: Score and ranks from multiple variable importance tables were aggregated.

## See Also

`get_vimp_table`, `aggregate_vimp_table`

---

`waiver` *Create a waiver object*

---

## Description

This function is functionally identical to `ggplot2::waiver()` function and creates a waiver object. A waiver object is an otherwise empty object that serves the same purpose as `NULL`, i.e. as placeholder for a default value. Because `NULL` can sometimes be a valid input argument, it can therefore not be used to switch to an internal default value.

## Usage

```
waiver()
```

## Value

waiver object

# Index