Package 'mmb'

May 9, 2026

Type Package

Title Arbitrary Dependency Mixed Multivariate Bayesian Models

Version 0.13.3

Author Sebastian Hönel

Maintainer Sebastian Hönel <sebastian.honel@lnu.se>

Description Supports Bayesian models with full and partial (hence arbitrary) dependencies between random variables. Discrete and continuous variables are supported, and conditional joint probabilities and probability densities are estimated using Kernel Density Estimation (KDE). The full general form, which implements an extension to Bayes' theorem, as well as the simple form, which is just a Bayesian network, both support regression through segmentation and KDE and estimation of probability or relative likelihood of discrete or continuous target random variables. This package also provides true statistical distance measures based on Bayesian models. Furthermore, these measures can be facilitated on neighborhood searches, and to estimate the similarity and distance between data points. Related work is by Bayes (1763) <doi:10.1098/rstl.1763.0053> and by Scutari (2010) <doi:10.18637/jss.v035.i03>.

License GPL-3

Encoding UTF-8

URL https://github.com/MrShoenel/R-mmb

BugReports https://github.com/MrShoenel/R-mmb/issues

LazyData true

VignetteBuilder knitr

Suggests devtools, testthat, covr, e1071, caret, knitr, rmarkdown, ggplot2, ggpubr, cowplot, philentropy, Rtsne

RoxygenNote 7.1.1

Imports Rdpack, datasets, stats, foreach, parallel, doParallel

RdMacros Rdpack

NeedsCompilation no

Repository CRAN

Date/Publication 2020-09-23 08:00:02 UTC

Contents

bayesCaret . . . 2
bayesComputeMarginalFactor . . . 3
bayesConvertData . . . 4
bayesFeaturesToSample . . . 5
bayesInferSimple . . . 5
bayesProbability . . . 7
bayesProbabilityAssign . . . 9
bayesProbabilityNaive . . . 11
bayesProbabilitySimple . . . 13
bayesRegress . . . 14
bayesRegressAssign . . . 16
bayesRegressSimple . . . 18
bayesToLatex . . . 19
centralities . . . 20
conditionalDataMin . . . 22
createFeatureForBayes . . . 23
discretizeVariableToRanges . . . 24
distance . . . 25
estimatePdf . . . 26
getDefaultRegressor . . . 27
getMessages . . . 28
getProbForDiscrete . . . 28
getRangeForDiscretizedValue . . . 29
getValueKeyOfBayesFeatures . . . 30
getValueOfBayesFeatures . . . 31
getWarnings . . . 32
neighborhood . . . 32
sampleToBayesFeatures . . . 33
setDefaultRegressor . . . 34
setMessages . . . 34
setWarnings . . . 35
vicinities . . . 35
vicinitiesForSample . . . 37

Index **39**

`bayesCaret` *Provides a caret-compatible wrapper around functionality for classification and regression, as implemented by mmb.*

Description

A wrapper to be used with the package/function `caret::train()`. Supports regression and classification and an extensive default grid.

Usage

```
bayesCaret
```

Format

An object of class `list` of length 7.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
trainIndex <- caret::createDataPartition(
  iris$Species, p = .8, list = FALSE, times = 1)
train <- iris[ trainIndex, ]
test  <- iris[-trainIndex, ]

fitControl <- caret::trainControl(
  method = "repeatedcv", number = 2, repeats = 2)

fit <- caret::train(
  Species ~ ., data = train, method = mmb::bayesCaret,
  trControl = fitControl)
```

---

bayesComputeMarginalFactor

Compute a marginal factor (continuous or discrete random variable).

Description

Computes the probability (discrete feature) or relative likelihood (continuous feature) of one given feature and a concrete value for it.

Usage

```
bayesComputeMarginalFactor(df, feature, doEcdf = FALSE)
```

Arguments

`df`	data.frame that contains all the feature's data
`feature`	data.frame containing the designated feature as created by @seealso `mmb::createFeatureForBayes()`.
`doEcdf`	default FALSE a boolean to indicate whether to use the empirical CDF to return a probability when inferencing a continuous feature. If false, uses the empirical PDF to return the rel. likelihood. This parameter does not have any effect when inferring discrete values. Using the ECDF, a probability to find a value less than or equal to the given value is returned.

Value

numeric the probability or likelihood of the given feature assuming its given value.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
feat <- mmb::createFeatureForBayes(
  name = "Petal.Length", value = mean(iris$Petal.Length))
mmb::bayesComputeMarginalFactor(df = iris, feature = feat)
mmb::bayesComputeMarginalFactor(df = iris, feature = feat, doEcdf = TRUE)
```

---

bayesConvertData *Convert data for usage within Bayesian models.*

Description

Converts all columns in a data.frame that are factors to character, except for the target column.

Usage

```
bayesConvertData(df)
```

Arguments

df	data.frame to be used for bayesian inferencing.

Value

the same data.frame with all factors converted to character.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
df <- mmb::bayesConvertData(df = iris)
```

bayesFeaturesToSample *Transform a collection of Bayesian features back to a sample.*

Description

Counter operation to @seealso `mmb::sampleToBayesFeatures()`. Takes a Bayes-feature data.frame and transforms it back to a row.

Usage

```
bayesFeaturesToSample(dfOrg, features)
```

Arguments

`dfOrg`	data.frame containing at least one row of the original format, so that we can rebuild the sample matching exactly the original column names.
`features`	data.frame of Bayes-features, as for example previously created using `mmb::sampleToBayesFeatures()`.

Value

data.frame the sample as 1-row data.frame.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
samp <- mmb::sampleToBayesFeatures(dfRow = iris[15,], targetCol = "Species")

# Convert the sample (as features) back to a sample that can be, e.g.,
# appended to the data again:
row <- mmb::bayesFeaturesToSample(dfOrg = iris, features = samp)
```

bayesInferSimple *Perform simple (network) Bayesian inferencing and regression.*

Description

Uses simple Bayesian inference to determine the probability or relative likelihood of a given value. This function can also regress to the most likely value instead. Simple means that segmented data is used in a way that is equal to how a Bayesian network works. For a finite set of labels, this function needs to be called for each, to obtain the probability of each label (or, for n-1 labels or until a label with >.5 probability is found). For obtaining the probability of a continuous value, this function is useful for deciding between picking among a finite set of values. The empirical CDF may be used to obtain an actual probability for a given continuous value, otherwise, the empirical PDF is estimated and a relative likelihood is returned. For regression, set `doRegress = TRUE` to obtain the most likely value of the target feature, instead of obtaining its relative likelihood.

Usage

```
bayesInferSimple(
  df,
  features,
  targetCol,
  selectedFeatureNames = c(),
  retainMinValues = 1,
  doRegress = FALSE,
  doEcdf = FALSE,
  regressor = NULL
)
```

Arguments

df	data.frame
features	data.frame with bayes-features. One of the features needs to be the label-column.
targetCol	string with the name of the feature that represents the label.
selectedFeatureNames	vector default c(). Vector of strings that are the names of the features the to-predict label depends on. If an empty vector is given, then all of the features are used (except for the label). The order then depends on the features' order.
retainMinValues	integer to require a minimum amount of data points when segmenting the data feature by feature.
doRegress	default FALSE a boolean to indicate whether to do a regression instead of returning the relative likelihood of a continuous feature. If the target feature is discrete and regression is requested, will issue a warning.
doEcdf	default FALSE a boolean to indicate whether to use the empirical CDF to return a probability when inferencing a continuous feature. If false, uses the empirical PDF to return the rel. likelihood. This parameter does not have any effect when inferring discrete values or when doing a regression.
regressor	Function that is given the collected values for regression and thus finally used to select a most likely value. Defaults to the built-in estimator for the empirical PDF and returns its argmax. However, any other function can be used, too, such as min, max, median, average etc. You may also use this function to obtain the raw values for further processing. This function is ignored if not doing regression.

Value

numeric probability (inferring discrete labels) or relative likelihood (regression, inferring likelihood of continuous value) or most likely value given the conditional features.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

References

Scutari M (2010). "Learning Bayesian Networks with the **bnlearn** R Package." *Journal of Statistical Software*, **35**(3), 1–22. doi: 10.18637/jss.v035.i03.

Examples

```
feat1 <- mmb::createFeatureForBayes(
  name = "Petal.Length", value = mean(iris$Petal.Length))
feat2 <- mmb::createFeatureForBayes(
  name = "Petal.Width", value = mean(iris$Petal.Width))
featT <- mmb::createFeatureForBayes(
  name = "Species", iris[1,]$Species, isLabel = TRUE)

# Infer likelihood of featT's label:
feats <- rbind(feat1, feat2, featT)
mmb::bayesInferSimple(df = iris, features = feats, targetCol = featT$name)

# Infer likelihood of feat1's value:
featT$isLabel = FALSE
feat1$isLabel = TRUE
# We do not bind featT this time:
feats <- rbind(feat1, feat2)
mmb::bayesInferSimple(df = iris, features = feats, targetCol = feat1$name)
```

`bayesProbability` *Full Bayesian inferencing for determining the probability or relative likelihood of a given value.*

Description

Uses the full extended theorem of Bayes, taking all selected features into account. Expands Bayes' theorem to accomodate all dependent features, then calculates each conditional probability (or relative likelihood) and returns a single result reflecting the probability or relative likelihood of the target feature assuming its given value, given that all the other dependent features assume their given value. The target feature (designated by 'labelCol') may be discrete or continuous. If at least one of the depending features or the the target feature is continuous and the PDF ('doEcdf' = FALSE) is built, the result of this function is a relative likelihood of the target feature's value. If all of the features are discrete or the empirical CDF is used instead of the PDF, the result of this function is a probability.

Usage

```
bayesProbability(
  df,
  features,
  targetCol,
  selectedFeatureNames = c(),
  shiftAmount = 0.1,
```

```
  retainMinValues = 1,
  doEcdf = FALSE,
  useParallel = NULL
)
```

Arguments

df	data.frame that contains all the feature's data
features	data.frame with bayes-features. One of the features needs to be the label-column.
targetCol	string with the name of the feature that represents the label.
selectedFeatureNames	vector default `c()`. Vector of strings that are the names of the features the to-predict label depends on. If an empty vector is given, then all of the features are used (except for the label). The order then depends on the features' order.
shiftAmount	numeric an offset value used to increase any one probability (factor) in the full built equation. In scenarios with many dependencies, it is more likely that a single conditional probability becomes zero, which would result in the entire probability being zero. Since this is often useless, the 'shiftAmount' can be added to each factor, resulting in a non-zero probability that can at least be used to order samples by likelihood. Note that, with a positive 'shiftAmount', the result of this function cannot be said to be a probability any longer, but rather results in a comparable likelihood (a 'probability score').
retainMinValues	integer to require a minimum amount of data points when segmenting the data feature by feature.
doEcdf	default FALSE a boolean to indicate whether to use the empirical CDF to return a probability when inferencing a continuous feature. If false, uses the empirical PDF to return the rel. likelihood. This parameter does not have any effect if all of the variables are discrete or when doing a regression. Otherwise, for each continuous variable, the probability to find a value less then or equal - given the conditions - is returned. Note that the interpretation of probability using the ECDF much deviates and must be used with care, especially since it affects each factor in Bayes equation that is continuous. This is especially true for the case where `shiftAmount > 0`.
useParallel	default NULL a boolean to indicate whether to use a previously registered parallel backend. If no explicit value was given, calls `foreach::getDoParRegistered()` to check for a parallel backend. When using parallelism, this function calculates each factor in the numerator and denominator of the final equation in parallel.

Value

numeric probability (inferring discrete labels) or relative likelihood (regression, inferring likelihood of continuous value) or most likely value given the conditional features. If using a positive `shiftAmount`, the result is a 'probability score'.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

References

Bayes T (1763). "LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S." *Philosophical transactions of the Royal Society of London*, 370–418.

See Also

test-case "a zero denominator can happen"

Examples

```
feat1 <- mmb::createFeatureForBayes(
  name = "Petal.Length", value = mean(iris$Petal.Length))
feat2 <- mmb::createFeatureForBayes(
  name = "Petal.Width", value = mean(iris$Petal.Width))
featT <- mmb::createFeatureForBayes(
  name = "Species", iris[1,]$Species, isLabel = TRUE)

# Check the probability of Species=setosa, given the other 2 features:
mmb::bayesProbability(
  df = iris, features = rbind(feat1, feat2, featT), targetCol = "Species")

# Now check the probability of Species=versicolor:
featT$valueChar <- "versicolor"
mmb::bayesProbability(
  df = iris, features = rbind(feat1, feat2, featT), targetCol = "Species")
```

`bayesProbabilityAssign` *Assign probabilities to one or more samples, given some training data.*

Description

This method uses full-dependency (`simple=F`) Bayesian inferencing to assign a probability to the target feature in all of the samples given in `dfValid`. Tests each sample using @seealso `mmb::bayesProbability()` or @seealso `mmb::bayesProbabilitySimple()`. It mostly forwards the given arguments to these functions, and you will find good documentation there.

Usage

```
bayesProbabilityAssign(
  dfTrain,
  dfValid,
  targetCol,
  selectedFeatureNames = c(),
  shiftAmount = 0.1,
  retainMinValues = 1,
  doEcdf = FALSE,
```

```
  online = 0,
  simple = FALSE,
  naive = FALSE,
  useParallel = NULL,
  returnProbabilityTable = FALSE
)
```

Arguments

dfTrain	data.frame that holds the training data.
dfValid	data.frame that holds the validation samples, for each of which a probability is sought. The convention is, that if you attempt to assign a probability to a numeric value, it ought to be found in the target column of this data frame (otherwise, the target column is not required in it).
targetCol	character the name of targeted feature, i.e., the feature to assign a probability to.
selectedFeatureNames	character defaults to empty vector which defaults to using all available features. Use this to select subsets of features and to order features.
shiftAmount	numeric an offset value used to increase any one probability (factor) in the full built equation.
retainMinValues	integer to require a minimum amount of data points when segmenting the data feature by feature.
doEcdf	default FALSE a boolean to indicate whether to use the empirical CDF to return a probability when inferencing a continuous feature.
online	default 0 integer to indicate how many rows should be used to do inferencing. If zero, then only the initially given data.frame dfTrain is used. If > 0, then each inferenced sample will be attached to it and the resulting data.frame is truncated to this number. Use an integer large enough (i.e., sum of training and validation rows) to keep all samples during inferencing. A smaller amount as, e.g., in dfTrain, will keep the amount of data restricted, discarding older rows. A larger amount than, e.g., in dfTrain is also fine; dfTrain will grow to it and then discard rows.
simple	default FALSE boolean to indicate whether or not to use simple Bayesian inferencing instead of full. This is faster but the results are less good. If true, uses `mmb::bayesProbabilitySimple()`. Otherwise, uses `mmb::bayesProbability()`.
naive	default FALSE boolean to indicate whether or not to use naive Bayesian inferencing instead of full or simple.
useParallel	boolean DEFAULT NULL this is forwarded to the underlying function `mmb::bayesProbability()` (only in simple=FALSE mode).
returnProbabilityTable	default FALSE boolean to indicate whether to return only the probabilities for each validation sample or whether a table with a probability for each tested label should be returned. This has no effect when inferencing probabilities for numeric values, as the table then only has one column "probability". The first column of this table is always called "rowname" and corresponds to the rownames of dfValid.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

References

Bayes T (1763). "LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S." *Philosophical transactions of the Royal Society of London*, 370–418.

Examples

```
w <- mmb::getWarnings()
mmb::setWarnings(FALSE)

set.seed(84735)
rn <- base::sample(rownames(iris), 150)
dfTrain <- iris[rn[1:120], ]
dfValid <- iris[rn[121:150], !(colnames(iris) %in% "Species") ]
mmb::bayesProbabilityAssign(dfTrain, dfValid, "Species")

mmb::setWarnings(w)
```

`bayesProbabilityNaive` *Naive Bayesian inferencing for determining the probability or relative likelihood of a given value.*

Description

A complementary implementation using methods common in mmb, such as computing factors or segmenting data. Supports Laplacian smoothing and early-stopping segmenting, as well as PDF and CDF and selecting any subset of features for dependency.

Usage

```
bayesProbabilityNaive(
  df,
  features,
  targetCol,
  selectedFeatureNames = c(),
  shiftAmount = 0.1,
  retainMinValues = 1,
  doEcdf = FALSE,
  useParallel = NULL
)
```

Arguments

df	data.frame that contains all the feature's data
features	data.frame with bayes-features. One of the features needs to be the label-column.
targetCol	string with the name of the feature that represents the label.
selectedFeatureNames	vector default c(). Vector of strings that are the names of the features the to-predict label depends on. If an empty vector is given, then all of the features are used (except for the label). The order then depends on the features' order.
shiftAmount	numeric an offset value used to increase any one probability (factor) in the full built equation. In scenarios with many dependencies, it is more likely that a single conditional probability becomes zero, which would result in the entire probability being zero. Since this is often useless, the 'shiftAmount' can be added to each factor, resulting in a non-zero probability that can at least be used to order samples by likelihood. Note that, with a positive 'shiftAmount', the result of this function cannot be said to be a probability any longer, but rather results in a comparable likelihood (a 'probability score').
retainMinValues	integer to require a minimum amount of data points when segmenting the data feature by feature.
doEcdf	default FALSE a boolean to indicate whether to use the empirical CDF to return a probability when inferencing a continuous feature. If false, uses the empirical PDF to return the rel. likelihood. This parameter does not have any effect if all of the variables are discrete or when doing a regression. Otherwise, for each continuous variable, the probability to find a value less then or equal - given the conditions - is returned. Note that the interpretation of probability using the ECDF much deviates and must be used with care, especially since it affects each factor in Bayes equation that is continuous. This is especially true for the case where shiftAmount > 0.
useParallel	default NULL a boolean to indicate whether to use a previously registered parallel backend. If no explicit value was given, calls foreach::getDoParRegistered() to check for a parallel backend. When using parallelism, this function calculates each factor in the numerator and denominator of the final equation in parallel.

Value

numeric probability (inferring discrete labels) or relative likelihood (regression, inferring likelihood of continuous value) or most likely value given the conditional features. If using a positive shiftAmount, the result is a 'probability score'.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
feat1 <- mmb::createFeatureForBayes(
  name = "Petal.Length", value = mean(iris$Petal.Length))
```

```
feat2 <- mmb::createFeatureForBayes(
  name = "Petal.Width", value = mean(iris$Petal.Width))
featT <- mmb::createFeatureForBayes(
  name = "Species", iris[1,]$Species, isLabel = TRUE)

# Check the probability of Species=setosa, given the other 2 features:
mmb::bayesProbabilityNaive(
  df = iris, features = rbind(feat1, feat2, featT), targetCol = "Species")

# Now check the probability of Species=versicolor:
featT$valueChar <- "versicolor"
mmb::bayesProbabilityNaive(
  df = iris, features = rbind(feat1, feat2, featT), targetCol = "Species")
```

`bayesProbabilitySimple` *Assign a probability using a simple (network) Bayesian classifier.*

Description

Uses simple Bayesian inference to return the probability or relative likelihood or a discrete label or continuous value.

Usage

```
bayesProbabilitySimple(
  df,
  features,
  targetCol,
  selectedFeatureNames = c(),
  retainMinValues = 1,
  doEcdf = FALSE
)
```

Arguments

`df`	data.frame
`features`	data.frame with bayes-features. One of the features needs to be the label-column.
`targetCol`	string with the name of the feature that represents the label.
`selectedFeatureNames`	vector default `c()`. Vector of strings that are the names of the features the to-predict label depends on. If an empty vector is given, then all of the features are used (except for the label). The order then depends on the features' order.
`retainMinValues`	integer to require a minimum amount of data points when segmenting the data feature by feature.
`doEcdf`	default FALSE a boolean to indicate whether to use the empirical CDF to return a probability when inferencing a continuous feature. If false, uses the empirical PDF to return the rel. likelihood.

Value

double the probability of the target-label, using the maximum a posteriori estimate.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

References

Scutari M (2010). "Learning Bayesian Networks with the **bnlearn** R Package." *Journal of Statistical Software*, **35**(3), 1–22. doi: 10.18637/jss.v035.i03.

See Also

`mmb::bayesInferSimple()`

Examples

```
feat1 <- mmb::createFeatureForBayes(
  name = "Sepal.Length", value = mean(iris$Sepal.Length))
feat2 <- mmb::createFeatureForBayes(
  name = "Sepal.Width", value = mean(iris$Sepal.Width), isLabel = TRUE)

# Assign a probability to a continuous variable (also works with nominal):
mmb::bayesProbabilitySimple(df = iris, features = rbind(feat1, feat2),
  targetCol = feat2$name, retainMinValues = 5, doEcdf = TRUE)
```

---

bayesRegress *Perform full-dependency Bayesian regression for a sample.*

Description

This method performs full-dependency regression by discretizing the continuous target variable into ranges (buckets), then finding the most probable ranges. It can either regress on the values in the most likely range or sample from all ranges, according to their likelihood.

Usage

```
bayesRegress(
  df,
  features,
  targetCol,
  selectedFeatureNames = c(),
  shiftAmount = 0.1,
  retainMinValues = 2,
  doEcdf = FALSE,
  useParallel = NULL,
  numBuckets = ceiling(log2(nrow(df))),
```

```
  sampleFromAllBuckets = TRUE,
  regressor = NULL
)
```

Arguments

df	data.frame that contains all the feature's data
features	data.frame with bayes-features. One of the features needs to be the label-column.
targetCol	string with the name of the feature that represents the label.
selectedFeatureNames	vector default `c()`. Vector of strings that are the names of the features the to-predict label depends on. If an empty vector is given, then all of the features are used (except for the label). The order then depends on the features' order.
shiftAmount	numeric an offset value used to increase any one probability (factor) in the full built equation. In scenarios with many dependencies, it is more likely that a single conditional probability becomes zero, which would result in the entire probability being zero. Since this is often useless, the 'shiftAmount' can be added to each factor, resulting in a non-zero probability that can at least be used to order samples by likelihood. Note that, with a positive 'shiftAmount', the result of this function cannot be said to be a probability any longer, but rather results in a comparable likelihood (a 'probability score').
retainMinValues	integer to require a minimum amount of data points when segmenting the data feature by feature.
doEcdf	default FALSE a boolean to indicate whether to use the empirical CDF to return a probability when inferencing a continuous feature. If false, uses the empirical PDF to return the rel. likelihood. This parameter does not have any effect if all of the variables are discrete or when doing a regression. Otherwise, for each continuous variable, the probability to find a value less then or equal - given the conditions - is returned. Note that the interpretation of probability using the ECDF much deviates and must be used with care, especially since it affects each factor in Bayes equation that is continuous. This is especially true for the case where `shiftAmount > 0`.
useParallel	default NULL a boolean to indicate whether to use a previously registered parallel backend. If no explicit value was given, calls `foreach::getDoParRegistered()` to check for a parallel backend. When using parallelism, this function calculates each factor in the numerator and denominator of the final equation in parallel.
numBuckets	integer the amount of buckets to for discretization. Buckets are built in an equidistant manner, not as quantiles (i.e., one bucket has likely a different amount of values than another).
sampleFromAllBuckets	default TRUE boolean to indicate how to obtain values for regression from the buckets. If true, than takes values from those buckets with a non-zero probability, and according to their probability. If false, selects all values from the bucket with the highest probability.

`regressor` Function that is given the collected values for regression and thus finally used to select a most likely value. Defaults to the built-in estimator for the empirical PDF and returns its argmax. However, any other function can be used, too, such as min, max, median, average etc. You may also use this function to obtain the raw values for further processing.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
w <- mmb::getWarnings()
mmb::setWarnings(FALSE)

df <- iris[, ]
set.seed(84735)
rn <- base::sample(rownames(df), 150)
dfTrain <- df[1:120, ]
dfValid <- df[121:150, ]
tf <- mmb::sampleToBayesFeatures(dfValid[1,], "Sepal.Length")
mmb::bayesRegress(dfTrain, tf, "Sepal.Length")

mmb::setWarnings(w)
```

---

bayesRegressAssign *Regression for one or more samples, given some training data.*

Description

This method uses full-dependency (`simple=F`) Bayesian inferencing to to a regression for the target features for all of the samples given in `dfValid`. Assigns a regression value using either

Usage

```
bayesRegressAssign(
  dfTrain,
  dfValid,
  targetCol,
  selectedFeatureNames = c(),
  shiftAmount = 0.1,
  retainMinValues = 2,
  doEcdf = FALSE,
  online = 0,
  simple = FALSE,
  useParallel = NULL,
  numBuckets = ceiling(log2(nrow(df))),
  sampleFromAllBuckets = TRUE,
  regressor = NULL
)
```

Arguments

dfTrain	data.frame that holds the training data.
dfValid	data.frame that holds the validation samples, for each of which a probability is sought. The convention is, that if you attempt to assign a probability to a numeric value, it ought to be found in the target column of this data frame (otherwise, the target column is not required in it).
targetCol	character the name of targeted feature, i.e., the feature to assign a probability to.
selectedFeatureNames	character defaults to empty vector which defaults to using all available features. Use this to select subsets of features and to order features.
shiftAmount	numeric an offset value used to increase any one probability (factor) in the full built equation.
retainMinValues	integer to require a minimum amount of data points when segmenting the data feature by feature.
doEcdf	default FALSE a boolean to indicate whether to use the empirical CDF to return a probability when inferencing a continuous feature.
online	default 0 integer to indicate how many rows should be used to do inferencing. If zero, then only the initially given data.frame dfTrain is used. If > 0, then each inferenced sample will be attached to it and the resulting data.frame is truncated to this number. Use an integer large enough (i.e., sum of training and validation rows) to keep all samples during inferencing. A smaller amount as, e.g., in dfTrain, will keep the amount of data restricted, discarding older rows. A larger amount than, e.g., in dfTrain is also fine; dfTrain will grow to it and then discard rows.
simple	default FALSE boolean to indicate whether or not to use simple Bayesian inferencing instead of full. This is faster but the results are less good. If true, uses `mmb::bayesRegressSimple()`. Otherwise, uses `mmb::bayesRegress()`.
useParallel	boolean DEFAULT NULL this is forwarded to the underlying function `mmb::bayesRegress()` (only in simple=FALSE mode).
numBuckets	integer the amount of buckets to for discretization. Buckets are built in an equidistant manner, not as quantiles (i.e., one bucket has likely a different amount of values than another).
sampleFromAllBuckets	default TRUE boolean to indicate how to obtain values for regression from the buckets. If true, than takes values from those buckets with a non-zero probability, and according to their probability. If false, selects all values from the bucket with the highest probability.
regressor	Function that is given the collected values for regression and thus finally used to select a most likely value. Defaults to the built-in estimator for the empirical PDF and returns its argmax. However, any other function can be used, too, such as min, max, median, average etc. You may also use this function to obtain the raw values for further processing.#'

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

See Also

`mmb::bayesRegress()` (full) or @seealso `mmb::bayesRegressSimple()` if `simple=T`. It mostly forwards the given arguments to these functions, and you will find good documentation there.

Examples

```
df <- iris[, ]
set.seed(84735)
rn <- base::sample(rownames(df), 150)
dfTrain <- df[1:120, ]
dfValid <- df[121:150, ]
res <- mmb::bayesRegressAssign(
  dfTrain, dfValid[, !(colnames(dfValid) %in% "Sepal.Length")],
  "Sepal.Length", sampleFromAllBuckets = TRUE, doEcdf = TRUE)
cov(res, iris[121:150,]$Sepal.Length)^2
```

---

bayesRegressSimple *Perform simple (network) Bayesian regression.*

Description

Uses simple Bayesian inferencing to segment the data given the conditional features. Then estimates a density over the remaining values of the target feature and returns the most likely value using a maximum a posteriori estimate of the kernel (returning its mode).

Usage

```
bayesRegressSimple(
  df,
  features,
  targetCol,
  selectedFeatureNames = c(),
  retainMinValues = 2,
  regressor = NULL
)
```

Arguments

`df`	data.frame
`features`	data.frame with bayes-features. One of the features needs to be the label-column (not required or no value required).
`targetCol`	string with the name of the feature that represents the label (here the target variable for regression).

`selectedFeatureNames`
: vector default `c()`. Vector of strings that are the names of the features the to-predict label depends on. If an empty vector is given, then all of the features are used (except for the label). The order then depends on the features' order.

`retainMinValues`
: integer to require a minimum amount of data points when segmenting the data feature by feature.

`regressor`
: Function that is given the collected values for regression and thus finally used to select a most likely value. Defaults to the built-in estimator for the empirical PDF and returns its argmax. However, any other function can be used, too, such as min, max, median, average etc. You may also use this function to obtain the raw values for further processing.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

References

Scutari M (2010). "Learning Bayesian Networks with the **bnlearn** R Package." *Journal of Statistical Software*, **35**(3), 1–22. doi: 10.18637/jss.v035.i03.

See Also

`mmb::bayesInferSimple()`

Examples

```
feat1 <- mmb::createFeatureForBayes(
  name = "Sepal.Length", value = mean(iris$Sepal.Length))
feat2 <- mmb::createFeatureForBayes(
  name = "Sepal.Width", value = mean(iris$Sepal.Width))

# Note how we do not require "Petal.Length" among the features when regressing:
mmb::bayesRegressSimple(df = iris, features = rbind(feat1, feat2),
  targetCol = "Petal.Length")
```

---

bayesToLatex *Create a string that can be used in Latex in an e.g. equation-environment.*

Description

This function can be used to generate Latex-markup that models the full dependency between covariates and a target variable.

Usage

```
bayesToLatex(conditionalFeatures, targetFeature, includeValues = FALSE)
```

Arguments

conditionalFeatures	data.frame of Bayesian features, the target feature depends on.
targetFeature	data.frame that holds exactly one Bayesian feature, that is supposed to be the target-feture for Bayesian inferencing.
includeValues	default FALSE boolean to indicate whether to include the features' values or not, i.e. "A" vs. "A = setosa".

Value

a string that can be used in Latex documents.

Note

Use `cat()` to print a string that can be copy-pasted.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
feat1 <- mmb::createFeatureForBayes(
  name = "Petal.Length", value = mean(iris$Petal.Length))
feat2 <- mmb::createFeatureForBayes(
  name = "Petal.Width", value = mean(iris$Petal.Width))
featT <- mmb::createFeatureForBayes(
  name = "Species", iris[1,]$Species, isLabel = TRUE)

cat(mmb::bayesToLatex(conditionalFeatures = rbind(feat1, feat2),
  targetFeature = featT, includeValues = TRUE))
```

---

centralities — *Given a neighborhood of data, computes the similarity of each sample in the neighborhood to the neighborhood.*

Description

Takes a data.frame of samples, then builds a PDF/PMF or ECDF for each of the selected features. Then, for each sample, computes the product of probabilities. The result is a vector that holds a probability for each sample. That probability (or relative likelihood) then represents the vicinity (or similarity) of the sample to the given neighborhood.

Usage

```
centralities(
  dfNeighborhood,
  selectedFeatureNames = c(),
  shiftAmount = 0.1,
  doEcdf = FALSE,
  ecdfMinusOne = FALSE
)
```

Arguments

dfNeighborhood	data.frame that holds all rows that make up the neighborhood.
selectedFeatureNames	vector of names of features to use. The centrality of each row in the neighborhood is calculated based on the selected features.
shiftAmount	numeric DEFAULT 0.1 optional amount to shift each features probability by. This is useful for when the centrality not necessarily must be an actual probability and too many features are selected. To obtain actual probabilities, this needs to be 0, and you must use the ECDF.
doEcdf	boolean DEFAULT FALSE whether to use the ECDF instead of the EPDF to find the likelihood of continuous values.
ecdfMinusOne	boolean DEFAULT FALSE only has an effect if the ECDF is used. If true, uses 1 minus the ECDF to find the probability of a continuous value. Depending on the interpretation of what you try to do, this may be of use.

Value

a named vector, where the names correspond to the rownames of the rows in the given neighborhood, and the value is the centrality of that row.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
# Create a neighborhood:
nbh <- mmb::neighborhood(df = iris, features = mmb::createFeatureForBayes(
  name = "Sepal.Width", value = mean(iris$Sepal.Width)))

cent <- mmb::centralities(dfNeighborhood = nbh, shiftAmount = 0.1,
  doEcdf = TRUE, ecdfMinusOne = TRUE)

# Plot the ordered samples to get an idea of the centralities in the neighborhood:
plot(x = names(cent), y=cent)
```

`conditionalDataMin` *Segment data according to one or more random variables.*

Description

Takes a data.frame and segments it, according to the selected variables. Only rows satisfying all conditions are kept. Supports discrete and continuous variables. Supports NA, NaN and NULL by using is.na, is.nan and is.null as comparator.

Usage

```
conditionalDataMin(
  df,
  features,
  selectedFeatureNames = c(),
  retainMinValues = 1
)
```

Arguments

`df`	data.frame with data to segment. If it contains less than or equally many rows as specified by `retainMinValues`, then the same data.frame is returned.
`features`	data.frame of bayes-features that are used to segment. Each feature's value is used to segment the data, and the features are used in the order as given by `selectedFeatureNames`. If those are not given, then the order of this data.frame is used.
`selectedFeatureNames`	default `c()`. Character vector with the names of the variables that shall be used for segmenting. Segmenting is done variable by variable, and the order depends on this vector. If this vector is empty, then the originally given data.frame is returned.
`retainMinValues`	default 1. The minimum amount of rows to retain. Filtering the data by the selected features may reduce the amount of remaining rows quickly, and this can be used as an early stopping criteria. Note that filtering is done variable by variable, and the amount of remaining rows is evaluated after each segmenting-step. If the threshold is undercut, then the result from the previous round is returned.

Value

data.frame that is segmented according to the selected variables and the minimum amount of rows to retain.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

See Also

`getValueKeyOfBayesFeatures()`

Examples

```
feat1 <- mmb::createFeatureForBayes(
  name = "Petal.Length", value = mean(iris$Petal.Length))
feat2 <- mmb::createFeatureForBayes(
  name = "Petal.Width", value = mean(iris$Petal.Width))
feats <- rbind(feat1, feat2)

data <- mmb::conditionalDataMin(df = iris, features = feats,
  selectedFeatureNames = feats$name, retainMinValues = 1)
```

`createFeatureForBayes` *Create a Bayesian feature by name and value.*

Description

Transforms a sample's feature's value into a dataframe, that holds its name, type and value. Currently supports numeric, factor, character and boolean values. Note that factor is internally converted to character.

Usage

```
createFeatureForBayes(name, value, isLabel = FALSE, isDiscrete = FALSE)
```

Arguments

`name`	the name of the feature or variable.
`value`	the value of the feature or variable.
`isLabel`	default FALSE. Indicates whether this feature or variable is the target variable (the label or value to predict).
`isDiscrete`	default FALSE. Used to indicate whether the feature or variable given is discrete. This will also be set to true if the value given is a charater, factor or a logical.

Value

A data.frame with one row holding all the feature's value's properties.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

See Also

`sampleToBayesFeatures` that uses this function

Examples

```
feat <- mmb::createFeatureForBayes(
  name = "Petal.Width", value = mean(iris$Petal.Width))
featTarget <- mmb::createFeatureForBayes(
  name = "Species", iris[1,]$Species, isLabel = TRUE)
```

`discretizeVariableToRanges` *Discretize a continuous random variable to ranges/buckets.*

Description

Discretizes a continuous random variable into buckets (ranges). Each range is delimited by an exclusive minimum value and an inclusive maximum value.

Usage

```
discretizeVariableToRanges(
  data,
  openEndRanges = TRUE,
  numRanges = NA,
  exclMinVal = NULL,
  inclMaxVal = NULL
)
```

Arguments

`data`	a vector with numeric data
`openEndRanges`	boolean default True. If true, then the minimum value of the first range will be set to @seealso `.Machine$double.xmin` and the maximum value of the last range will be set to @seealso `.Machine$double.xmax`, so that all values get covered.
`numRanges`	integer default NA. If NULL, then the amount of ranges (buckets) depends on the amount of data given. A minimum of two buckets is used then, and a maximum of ceiling(log2(length(data))).
`exclMinVal`	numeric default NULL. Used to delimit the lower bound of the given data. If not given, then no value is excluded, as the exclusive lower bound becomes the minimum of the given data minus an epsilon of 1e-15.
`inclMaxVal`	numeric default NULL. Used to delimit the upper bound of the given data. If not given, then the upper inclusive bound is the max of the given data.

Value

List a List of vectors, where each vector has two values, the first being the exclusive minimum value of the range, and the second being the inclusive maximum value of the range. The list will be as long as the number of buckets requested.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
buckets <- mmb::discretizeVariableToRanges(
  data = iris$Sepal.Length, openEndRanges = TRUE)

length(buckets)
buckets[[5]]
```

`distance` *Given a neighborhood of data and two samples from that neighborhood, calculates the distance between the samples.*

Description

The distance of two samples x,y from each other within a given neighborhood is defined as the absolute value of the subtraction of each sample's centrality to the neighborhood.

Usage

```
distance(
  dfNeighborhood,
  rowNrOfSample1,
  rowNrOfSample2,
  selectedFeatureNames = c(),
  shiftAmount = 0.1,
  doEcdf = FALSE,
  ecdfMinusOne = FALSE
)
```

Arguments

`dfNeighborhood`	data.frame that holds all rows that make up the neighborhood.
`rowNrOfSample1`	character the name of the row that constitutes the first sample from the given neighborhood.
`rowNrOfSample2`	character the name of the row that constitutes the second sample from the given neighborhood.
`selectedFeatureNames`	vector of names of features to use. The centrality of each row in the neighborhood is calculated based on the selected features.
`shiftAmount`	numeric DEFAULT 0.1 optional amount to shift each features probability by. This is useful for when the centrality not necessarily must be an actual probability and too many features are selected. To obtain actual probabilities, this needs to be 0, and you must use the ECDF.

doEcdf	boolean DEFAULT FALSE whether to use the ECDF instead of the EPDF to find the likelihood of continuous values.
ecdfMinusOne	boolean DEFAULT FALSE only has an effect if the ECDF is used. If true, uses 1 minus the ECDF to find the probability of a continuous value. Depending on the interpretation of what you try to do, this may be of use.

Value

numeric the distance as a positive number.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
# Show the distance between two samples using all their features:
mmb::distance(dfNeighborhood = iris, rowNrOfSample1 = 10, rowNrOfSample2 = 99)

# Let's use an actual neighborhood:
nbh <- mmb::neighborhood(df = iris, features = mmb::createFeatureForBayes(
  name = "Sepal.Length", value = mean(iris$Sepal.Length)))
mmb::distance(dfNeighborhood = nbh, rowNrOfSample1 = 1, rowNrOfSample2 = 30,
  selectedFeatureNames = colnames(iris)[1:3])

# Let's compare this to the distances as they are in iris (should be smaller):
mmb::distance(dfNeighborhood = iris, rowNrOfSample1 = 1, rowNrOfSample2 = 30,
  selectedFeatureNames = colnames(iris)[1:3])
```

---

estimatePdf *Safe PDF estimation that works also for sparse random variables.*

Description

Given a few observations of a random variable, this function returns an approximation of the PDF as a function. Returns also the PDF's support and argmax and works when only zero or one value was given. Depending on the used density function, two values are often enough to estimate a PDF.

Usage

```
estimatePdf(
  data = c(),
  densFun = function(vec) {     stats::density(vec, bw = "SJ") }
)
```

Arguments

`data`	vector of numeric data. Used to compute the empirical density of the data.
`densFun`	function default `stats::density` with bandwith 'SJ'. Function to compute the empirical density of a non-empty vector of numerical data. Note that this function needs to return the properties 'x' and 'y' as `stats::density` does, so that we can return the argmax.

Value

list with a function that is the empirical PDF using KDE. The list also has two properties 'min' and 'max' which represent the integratable range of that function. 'min' and 'max' are both zero if not data (an empty vector) was given. If one data point was given, then they correspond to its value -/+ `.Machine$double.eps`. The list further contains two numeric vectors 'x' and 'y', and a property 'argmax'. If no data was given, 'x' and 'y' are zero, and 'argmax' is NA. If one data points was given, then 'x' and 'argmax' equal it, and 'y' is set to 1. If two or more data points given, then the empirical density is estimated and 'x' and y' are filled from its estimate. 'argmax' is then set to that 'x', where 'y' becomes max.

Note

If the given vector is empty, warns and returns a constant function that always returns zero for all values.

If the given vector contains only one observation, then a function is returned that returns 1 iff the value supplied is the same as the observation. Otherwise, that function will return zero.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
epdf <- mmb::estimatePdf(data = iris$Petal.Width)
print(epdf$argmax)
plot(epdf)

# Get relative likelihood of some values:
epdf$fun(0.5)
epdf$fun(1.7)
```

`getDefaultRegressor` *Get the system-wide default regressor.*

Description

Getting and setting the default regressor affects all functions that have an overridable regressor. If this is not given, the default has defined here will be obtained.

Usage

```
getDefaultRegressor()
```

Value

Function the function used as the regressor. Defaults to `function(data) mmb::estimatePdf(data)$argmax`.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

---

getMessages *Get a boolean indicating whether messages are enabled system-wide.*

Description

Getter for the state of messages. Returns true if enabled.

Usage

```
getMessages()
```

Value

Boolean to indicate whether messages are enabled or not.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

---

getProbForDiscrete *Get a probability of a discrete value.*

Description

Similar to @seealso `estimatePdf`, this function returns the probability for a discrete value, given some observations.

Usage

```
getProbForDiscrete(data, value)
```

Arguments

`data`	vector of observations that have the same type as the given value.
`value`	a single observation of the same type as the data vector.

Value

the probability of value given data.

Note

If no observations are given, then this function will warn and return a probability of zero for the value given. While we could technically return positive infinity, 0 is more suitable in the context of Bayesian inferencing.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
mmb::getProbForDiscrete(data = c(), value = iris[1,]$Species)
mmb::getProbForDiscrete(data = iris$Species, value = iris[1,]$Species)
```

---

getRangeForDiscretizedValue

Get the range-/bucket-ID of a given value.

Description

Given a list of previously computed ranges for a random variable, this function returns the index of the range the given value belongs to (i.e., in which bucket it belongs). The indexes start R-typically at 1. Per definition, a value is within a range, if it is larger than the range's minimum and less than or equal to its maximum.

Usage

```
getRangeForDiscretizedValue(ranges, value)
```

Arguments

`ranges`	list of ranges, as obtained by @seealso `discretizeVariableToRanges`
`value`	numeric a value drawn from the previously discretized random variable.

Value

integer the index of the range the given value falls into.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
buckets <- mmb::discretizeVariableToRanges(
  data = iris$Sepal.Length, openEndRanges = TRUE)

mmb::getRangeForDiscretizedValue(
  ranges = buckets, value = mean(iris$Sepal.Length))
```

`getValueKeyOfBayesFeatures` *Obtain the type of the value of a Bayesian feature.*

Description

Given a data.frame with one or multiple features as constructed by @seealso `createFeatureForBayes` and a name, extracts the type of the feature specified by name. Note that this is only used internally.

Usage

```
getValueKeyOfBayesFeatures(dfFeature, featName)
```

Arguments

`dfFeature`	a data.frame for a single feature or variable as constructed by @seealso `createFeatureForBayes`.
`featName`	the name of the feature or variable of which to obtain the type.

Value

the (internal) type of the feature.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
feats <- rbind(
  mmb::createFeatureForBayes(
    "Petal.Width", value = mean(iris$Petal.Width)),
  mmb::createFeatureForBayes(
    name = "Species", iris[1,]$Species, isLabel = TRUE)
)

print(mmb::getValueKeyOfBayesFeatures(feats, "Species"))
print(mmb::getValueKeyOfBayesFeatures(feats, "Petal.Width"))
```

getValueOfBayesFeatures *Obtain the value of a Bayesian feature.*

Description

Given a data.frame with one or multiple features as constructed by @seealso `createFeatureForBayes` and a name, extracts the value of the feature specified by name.

Usage

```
getValueOfBayesFeatures(dfFeature, featName)
```

Arguments

`dfFeature`	a data.frame for a single feature or variable as constructed by @seealso `createFeatureForBayes`.
`featName`	the name of the feature or variable of which to obtain the value.

Value

the value of the feature.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
feats <- rbind(
  mmb::createFeatureForBayes(
    "Petal.Width", value = mean(iris$Petal.Width)),
  mmb::createFeatureForBayes(
    name = "Species", iris[1,]$Species, isLabel = TRUE)
)

print(mmb::getValueOfBayesFeatures(feats, "Species"))
print(mmb::getValueOfBayesFeatures(feats, "Petal.Width"))
```

`getWarnings` *Get a boolean indicating whether warnings are enabled system-wide.*

Description

Getter for the state of warnings. Returns true if enabled.

Usage

```
getWarnings()
```

Value

Boolean to indicate whether warnings are enabled or not.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

`neighborhood` *Given Bayesian features, returns those samples from a dataset that exhibit a similarity (i.e., the neighborhood).*

Description

The neighborhood N_i is defined as the set of samples that have a similarity greater than zero to the given sample s_i. Segmentation is done using equality (==) for discrete features and less than or equal (<=) for continuous features. Note that feature values NA and NaN are also supported using `is.na()` and `is.nan()`.

Usage

```
neighborhood(df, features, selectedFeatureNames = c(), retainMinValues = 0)
```

Arguments

`df`	data.frame to select the neighborhood from
`features`	data.frame of Bayes-features, used to segment/select the rows that should make up the neighborhood.
`selectedFeatureNames`	vector of names of features to use to demarcate the neighborhood. If empty, uses all features' names.
`retainMinValues`	DEFAULT 0 the amount of samples to retain during segmentation. For separating a neighborhood, this value typically should be 0, so that no samples are included that are not within it. However, for very sparse data or a great amount of variables, it might still make sense to retain samples.

Value

data.frame with rows that were selected as neighborhood. It is guaranteed that the rownames are maintained.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
nbh <- mmb::neighborhood(df = iris, features = mmb::createFeatureForBayes(
  name = "Sepal.Width", value = mean(iris$Sepal.Width)))

print(nrow(nbh))
```

---

sampleToBayesFeatures *Transform an entire sample into a collection of Bayesian features.*

Description

Helper function that takes one sample (e.g., a row of a dataframe with validation data) and transforms it into a data.frame where each row corresponds to one feature (and its value) of the sample. This is done using @seealso `createFeatureForBayes`. This operation can be thought of transposing a matrix.

Usage

```
sampleToBayesFeatures(dfRow, targetCol)
```

Arguments

`dfRow`	a row of a data.frame with a value for each feature.
`targetCol`	the name of the feature (column in the data.frame) that is the target variable for classification or regression.

Value

a data.frame where the first row is the feature that represents the label.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
# Converts all features of iris; the result is a data.frame of length
# equal to the amount of features in iris (5). The first feature is
# targetCol (has isLabel=TRUE).
samp <- mmb::sampleToBayesFeatures(dfRow = iris[15,], targetCol = "Species")
```

`setDefaultRegressor` *Set a system-wide default regressor.*

Description

Getting and setting the default regressor affects all functions that have an overridable regressor. If this is not given, the default has defined here will be obtained.

Usage

```
setDefaultRegressor(func)
```

Arguments

`func`	a Function to use a regressor, should accept one argument, which is a vector of numeric, and return one value, the regression.

Value

void

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

`setMessages` *Enable or disable messages system-wide.*

Description

Setter for enabling or disabling messages. Messages are disabled by default. Use these to enable high verbosity.

Usage

```
setMessages(enable = TRUE)
```

Arguments

`enable`	a boolean to indicate whether to enable messages or not.

Value

Boolean the state of enabled

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

`setWarnings` *Enable or disable warnings system-wide.*

Description

Setter for enabling or disabling warnings. Warnings are enabled by default.

Usage

```
setWarnings(enable = TRUE)
```

Arguments

`enable`	a boolean to indicate whether to enable warnings or not.

Value

Boolean the state of enabled

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

`vicinities` *Segment a dataset by each row once, then compute vicinities of samples in the neighborhood.*

Description

Given an entire dataset, uses each instance in it to demarcate a neighborhood using the selected features. Then, for each neighborhood, the vicinity of all samples to it is computed. The result of this is an N x N matrix, where the entry $m_{i,j}$ corresponds to the vicinity of sample s_j in neighborhood N_i.

Usage

```
vicinities(
  df,
  selectedFeatureNames = c(),
  shiftAmount = 0.1,
  doEcdf = FALSE,
  ecdfMinusOne = FALSE,
  retainMinValues = 0,
  useParallel = NULL
)
```

Arguments

df	data.frame to compute the matrix of vicinites for.
selectedFeatureNames	vector of names of features to use for computing the vicinity/centrality of each sample to each neighborhood.
shiftAmount	numeric DEFAULT 0.1 optional amount to shift each features probability by. This is useful for when the centrality not necessarily must be an actual probability and too many features are selected. To obtain actual probabilities, this needs to be 0, and you must use the ECDF.
doEcdf	boolean DEFAULT FALSE whether to use the ECDF instead of the EPDF to find the likelihood of continuous values.
ecdfMinusOne	boolean DEFAULT FALSE only has an effect if the ECDF is used. If true, uses 1 minus the ECDF to find the probability of a continuous value. Depending on the interpretation of what you try to do, this may be of use.
retainMinValues	DEFAULT 0 the amount of samples to retain during segmentation. For separating a neighborhood, this value typically should be 0, so that no samples are included that are not within it. However, for very sparse data or a great amount of variables, it might still make sense to retain samples.
useParallel	boolean DEFAULT NULL whether to use parallelism or not. Setting this to true requires also having previously registered a parallel backend. If parallel computing is enabled, then each neighborhood is computed separately.

Value

matrix of length N^2 (N being the length of the data.frame). Each row i demarcates the neighborhood as selected by sample i, and each column j then is the vicinity of sample s_j to that neighborhood. No value of the diagonal is zero, because each neighborhood always contains the sample it was demarcated by, and that sample has a similarity greater than zero to it.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

See Also

`vicinitiesForSample()`

Examples

```
w <- mmb::getWarnings()
mmb::setWarnings(FALSE)
mmb::vicinities(df = iris[1:10,])

# Run the same, but use the ECDF and retain more values:
mmb::vicinities(df = iris[1:10,], doEcdf = TRUE, retainMinValues = 10)
mmb::setWarnings(w)
```

`vicinitiesForSample` *Segment a dataset by a single sample and compute vicinities for it and the remaining samples in the neighborhood.*

Description

Given some data and one sample s_i from it, constructs the neighborhood N_i of that sample and assigns centralities to all other samples in that neighborhood to it. Samples that lie outside the neighborhood are assigned a vicinity of zero. Uses `mmb::neighborhood()` and `mmb::centralities()`.

Usage

```
vicinitiesForSample(
  df,
  sampleFromDf,
  selectedFeatureNames = c(),
  shiftAmount = 0.1,
  doEcdf = FALSE,
  ecdfMinusOne = FALSE,
  retainMinValues = 0
)
```

Arguments

`df`	data.frame that holds the data (and also the sample to use to define the neighborhood). Each sample in this data.frame is assigned a vicinity.
`sampleFromDf`	data.frame a single row from the given data.frame. This is used to select a neighborhood from the given data.
`selectedFeatureNames`	vector of names of features to use to compute the vicinity/centrality. This is passed to `mmb::neighborhood()`.
`shiftAmount`	numeric DEFAULT 0.1 optional amount to shift each features probability by. This is useful for when the centrality not necessarily must be an actual probability and too many features are selected. To obtain actual probabilities, this needs to be 0, and you must use the ECDF.
`doEcdf`	boolean DEFAULT FALSE whether to use the ECDF instead of the EPDF to find the likelihood of continuous values.
`ecdfMinusOne`	boolean DEFAULT FALSE only has an effect if the ECDF is used. If true, uses 1 minus the ECDF to find the probability of a continuous value. Depending on the interpretation of what you try to do, this may be of use.
`retainMinValues`	DEFAULT 0 the amount of samples to retain during segmentation. For separating a neighborhood, this value typically should be 0, so that no samples are included that are not within it. However, for very sparse data or a great amount of variables, it might still make sense to retain samples.

Value

data.frame with a single column 'vicinity' and the same rownames as the given data.frame. Each row then holds the vicinity for the corresponding row.

Author(s)

Sebastian Hönel sebastian.honel@lnu.se

Examples

```
vic <- mmb::vicinitiesForSample(
  df = iris, sampleFromDf = iris[1,], shiftAmount = 0.1)
vic$vicinity

# Plot the ordered samples to get an idea which ones have a vicinity > 0
plot(x=rownames(vic), y=vic$vicinity)
```

Index

∗ **classification**
bayesProbability, 7
bayesProbabilityAssign, 9
bayesProbabilityNaive, 11
∗ **datasets**
bayesCaret, 2
∗ **density-estimation**
estimatePdf, 26
∗ **discretization**
discretizeVariableToRanges, 24
getRangeForDiscretizedValue, 29
∗ **feature**
bayesFeaturesToSample, 5
createFeatureForBayes, 23
getValueKeyOfBayesFeatures, 30
getValueOfBayesFeatures, 31
sampleToBayesFeatures, 33
∗ **full-dependency**
bayesProbability, 7
bayesProbabilityAssign, 9
bayesRegress, 14
bayesRegressAssign, 16
∗ **inferencing**
bayesInferSimple, 5
bayesProbability, 7
bayesProbabilityAssign, 9
bayesProbabilityNaive, 11
bayesProbabilitySimple, 13
∗ **likelihood**
estimatePdf, 26
getProbForDiscrete, 28
∗ **naive**
bayesProbabilityNaive, 11
∗ **network**
centralities, 20
distance, 25
neighborhood, 32
vicinities, 35
vicinitiesForSample, 37
∗ **probability**
getProbForDiscrete, 28
∗ **regression**
bayesInferSimple, 5
bayesRegress, 14
bayesRegressAssign, 16
bayesRegressSimple, 18
∗ **segmentation**
conditionalDataMin, 22
∗ **simple**
bayesInferSimple, 5
bayesProbabilitySimple, 13
bayesRegressSimple, 18

bayesCaret, 2
bayesComputeMarginalFactor, 3
bayesConvertData, 4
bayesFeaturesToSample, 5
bayesInferSimple, 5
bayesProbability, 7
bayesProbabilityAssign, 9
bayesProbabilityNaive, 11
bayesProbabilitySimple, 13
bayesRegress, 14
bayesRegressAssign, 16
bayesRegressSimple, 18
bayesToLatex, 19

centralities, 20
conditionalDataMin, 22
createFeatureForBayes, 23

discretizeVariableToRanges, 24
distance, 25

estimatePdf, 26

getDefaultRegressor, 27
getMessages, 28
getProbForDiscrete, 28
getRangeForDiscretizedValue, 29

getValueKeyOfBayesFeatures, 30
getValueOfBayesFeatures, 31
getWarnings, 32

neighborhood, 32

sampleToBayesFeatures, 33
setDefaultRegressor, 34
setMessages, 34
setWarnings, 35

vicinities, 35
vicinitiesForSample, 37