Package 'pangoling'

May 9, 2026

Type Package

Title Access to Large Language Model Predictions

Version 1.0.3

Description Provides access to word predictability estimates using large language models (LLMs) based on 'transformer' architectures via integration with the 'Hugging Face' ecosystem <https://huggingface.co/>. The package interfaces with pre-trained neural networks and supports both causal/auto-regressive LLMs (e.g., 'GPT-2') and masked/bidirectional LLMs (e.g., 'BERT') to compute the probability of words, phrases, or tokens given their linguistic context. For details on GPT-2 and causal models, see Radford et al. (2019) <https://storage.prod.researchhub.com/uploads/papers/2020/06/01/language-models.pdf>, for details on BERT and masked models, see Devlin et al. (2019) <doi:10.48550/arXiv.1810.04805>. By enabling a straightforward estimation of word predictability, the package facilitates research in psycholinguistics, computational linguistics, and natural language processing (NLP).

License MIT + file LICENSE

URL https://docs.ropensci.org/pangoling/, https://github.com/ropensci/pangoling

BugReports https://github.com/ropensci/pangoling/issues

Depends R (>= 4.1.0)

Imports cachem, data.table, memoise, reticulate, rstudioapi, stats, tidyselect, tidytable (>= 0.7.2), utils

Suggests brms, knitr, parallel, rmarkdown, spelling, testthat (>= 3.0.0), tictoc, covr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.1

StagedInstall yes

VignetteBuilder knitr

NeedsCompilation no

Author Bruno Nicenboim [aut, cre] (ORCID:
<https://orcid.org/0000-0002-5176-3943>),
Chris Emmerly [ctb],
Giovanni Cassani [ctb],
Lisa Levinson [rev],
Utku Turk [rev]

Maintainer Bruno Nicenboim <b.nicenboim@tilburguniversity.edu>

Repository CRAN

Date/Publication 2025-04-07 17:00:02 UTC

Contents

causal_config . . . 3
causal_next_tokens_pred_tbl . . . 4
causal_pred_mats . . . 6
causal_preload . . . 8
causal_words_pred . . . 9
df_jaeger14 . . . 12
df_sent . . . 15
installed_py_pangoling . . . 15
install_py_pangoling . . . 16
masked_config . . . 17
masked_preload . . . 18
masked_targets_pred . . . 19
masked_tokens_pred_tbl . . . 21
ntokens . . . 23
perplexity_calc . . . 24
set_cache_folder . . . 25
tokenize_lst . . . 26
transformer_vocab . . . 27

Index **28**

causal_config *Returns the configuration of a causal model*

Description

Returns the configuration of a causal model

Usage

```
causal_config(
  model = getOption("pangoling.causal.default"),
  checkpoint = NULL,
  config_model = NULL
)
```

Arguments

model	Name of a pre-trained model or folder. One should be able to use models based on "gpt2". See hugging face website.
checkpoint	Folder of a checkpoint.
config_model	List with other arguments that control how the model from Hugging Face is accessed.

Value

A list with the configuration of the model.

More details about causal models

A causal language model (also called GPT-like, auto-regressive, or decoder model) is a type of large language model usually used for text-generation that can predict the next word (or more accurately in fact token) based on a preceding context.

If not specified, the causal model used will be the one set in the global option `pangoling.causal.default`, this can be accessed via `getOption("pangoling.causal.default")` (by default "gpt2"). To change the default option use `options(pangoling.causal.default = "newcausalmodel")`.

A list of possible causal models can be found in Hugging Face website.

Using the `config_model` and `config_tokenizer` arguments, it's possible to control how the model and tokenizer from Hugging Face is accessed, see the Python method `from_pretrained` for details.

In case of errors when a new model is run, check the status of `https://status.huggingface.co/`

See Also

Other causal model helper functions: `causal_preload()`

Examples

```
causal_config(model = "gpt2")
```

`causal_next_tokens_pred_tbl` *Generate next tokens after a context and their predictability using a causal transformer model*

Description

This function predicts the possible next tokens and their predictability (log-probabilities by default). The function sorts tokens in descending order of their predictability.

Usage

```
causal_next_tokens_pred_tbl(
  context,
  log.p = getOption("pangoling.log.p"),
  decode = FALSE,
  model = getOption("pangoling.causal.default"),
  checkpoint = NULL,
  add_special_tokens = NULL,
  config_model = NULL,
  config_tokenizer = NULL
)
```

Arguments

`context`	A single string representing the context for which the next tokens and their predictabilities are predicted.
`log.p`	Base of the logarithm used for the output predictability values. If `TRUE` (default), the natural logarithm (base *e*) is used. If `FALSE`, the raw probabilities are returned. Alternatively, `log.p` can be set to a numeric value specifying the base of the logarithm (e.g., `2` for base-2 logarithms). To get surprisal in bits (rather than predictability), set `log.p = 1/2`.
`decode`	Logical. If `TRUE`, decodes the tokens into human-readable strings, handling special characters and diacritics. Default is `FALSE`.
`model`	Name of a pre-trained model or folder. One should be able to use models based on "gpt2". See hugging face website.
`checkpoint`	Folder of a checkpoint.
`add_special_tokens`	Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.

`config_model` List with other arguments that control how the model from Hugging Face is accessed.

`config_tokenizer` List with other arguments that control how the tokenizer from Hugging Face is accessed.

Details

The function uses a causal transformer model to compute the predictability of all tokens in the model's vocabulary, given a single input context. It returns a table where each row represents a token, along with its predictability score. By default, the function returns log-probabilities in natural logarithm (base *e*), but you can specify a different logarithm base (e.g., `log.p = 1/2` for surprisal in bits).

If `decode = TRUE`, the tokens are converted into human-readable strings, handling special characters like accents and diacritics. This ensures that tokens are more interpretable, especially for languages with complex tokenization.

Value

A table with possible next tokens and their log-probabilities.

More details about causal models

A causal language model (also called GPT-like, auto-regressive, or decoder model) is a type of large language model usually used for text-generation that can predict the next word (or more accurately in fact token) based on a preceding context.

If not specified, the causal model used will be the one set in the global option `pangoling.causal.default`, this can be accessed via `getOption("pangoling.causal.default")` (by default "gpt2"). To change the default option use `options(pangoling.causal.default = "newcausalmodel")`.

A list of possible causal models can be found in Hugging Face website.

Using the `config_model` and `config_tokenizer` arguments, it's possible to control how the model and tokenizer from Hugging Face is accessed, see the Python method `from_pretrained` for details.

In case of errors when a new model is run, check the status of `https://status.huggingface.co/`

See Also

Other causal model functions: `causal_pred_mats()`, `causal_words_pred()`

Examples

```
causal_next_tokens_pred_tbl(
  context = "The apple doesn't fall far from the",
  model = "gpt2"
)
```

causal_pred_mats *Generate a list of predictability matrices using a causal transformer model*

Description

This function computes a list of matrices, where each matrix corresponds to a unique group specified by the by argument. Each matrix represents the predictability of every token in the input text (x) based on preceding context, as evaluated by a causal transformer model.

Usage

```
causal_pred_mats(
  x,
  by = rep(1, length(x)),
  sep = " ",
  log.p = getOption("pangoling.log.p"),
  sorted = FALSE,
  model = getOption("pangoling.causal.default"),
  checkpoint = NULL,
  add_special_tokens = NULL,
  decode = FALSE,
  config_model = NULL,
  config_tokenizer = NULL,
  batch_size = 1,
  ...
)
```

Arguments

x	A character vector of words, phrases, or texts to evaluate.
by	A grouping variable indicating how texts are split into groups.
sep	A string specifying how words are separated within contexts or groups. Default is " ". For languages that don't have spaces between words (e.g., Chinese), set sep = "".
log.p	Base of the logarithm used for the output predictability values. If TRUE (default), the natural logarithm (base *e*) is used. If FALSE, the raw probabilities are returned. Alternatively, log.p can be set to a numeric value specifying the base of the logarithm (e.g., 2 for base-2 logarithms). To get surprisal in bits (rather than predictability), set log.p = 1/2.
sorted	When default FALSE it will retain the order of groups we are splitting by. When TRUE then sorted (according to by) list(s) are returned.
model	Name of a pre-trained model or folder. One should be able to use models based on "gpt2". See hugging face website.
checkpoint	Folder of a checkpoint.

`add_special_tokens`
: Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.

`decode`
: Logical. If `TRUE`, decodes the tokens into human-readable strings, handling special characters and diacritics. Default is `FALSE`.

`config_model`
: List with other arguments that control how the model from Hugging Face is accessed.

`config_tokenizer`
: List with other arguments that control how the tokenizer from Hugging Face is accessed.

`batch_size`
: Maximum number of sentences/texts processed in parallel. Larger batches increase speed but use more memory. Since all texts in a batch must have the same length, shorter ones are padded with placeholder tokens.

`...`
: Currently not in use.

Details

The function splits the input `x` into groups specified by the `by` argument and processes each group independently. For each group, the model computes the predictability of each token in its vocabulary based on preceding context.

Each matrix contains:

- Rows representing the model's vocabulary.
- Columns corresponding to tokens in the group (e.g., a sentence or paragraph).
- By default, values in the matrices are the natural logarithm of word probabilities.

Value

A list of matrices with tokens in their columns and the vocabulary of the model in their rows

More details about causal models

A causal language model (also called GPT-like, auto-regressive, or decoder model) is a type of large language model usually used for text-generation that can predict the next word (or more accurately in fact token) based on a preceding context.

If not specified, the causal model used will be the one set in the global option `pangoling.causal.default`, this can be accessed via `getOption("pangoling.causal.default")` (by default "gpt2"). To change the default option use `options(pangoling.causal.default = "newcausalmodel")`.

A list of possible causal models can be found in Hugging Face website.

Using the `config_model` and `config_tokenizer` arguments, it's possible to control how the model and tokenizer from Hugging Face is accessed, see the Python method `from_pretrained` for details.

In case of errors when a new model is run, check the status of `https://status.huggingface.co/`

See Also

Other causal model functions: `causal_next_tokens_pred_tbl()`, `causal_words_pred()`

Examples

```
data("df_sent")
df_sent
list_of_mats <- causal_pred_mats(
                        x = df_sent$word,
                        by = df_sent$sent_n,
                        model = "gpt2"
                )

# View the structure of the resulting list
list_of_mats |> str()

# Inspect the last rows of the first matrix
list_of_mats[[1]] |> tail()

# Inspect the last rows of the second matrix
list_of_mats[[2]] |> tail()
```

`causal_preload` *Preloads a causal language model*

Description

Preloads a causal language model to speed up next runs.

Usage

```
causal_preload(
  model = getOption("pangoling.causal.default"),
  checkpoint = NULL,
  add_special_tokens = NULL,
  config_model = NULL,
  config_tokenizer = NULL
)
```

Arguments

`model`	Name of a pre-trained model or folder. One should be able to use models based on "gpt2". See hugging face website.
`checkpoint`	Folder of a checkpoint.
`add_special_tokens`	Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.
`config_model`	List with other arguments that control how the model from Hugging Face is accessed.
`config_tokenizer`	List with other arguments that control how the tokenizer from Hugging Face is accessed.

Value

Nothing.

More details about causal models

A causal language model (also called GPT-like, auto-regressive, or decoder model) is a type of large language model usually used for text-generation that can predict the next word (or more accurately in fact token) based on a preceding context.

If not specified, the causal model used will be the one set in the global option `pangoling.causal.default`, this can be accessed via `getOption("pangoling.causal.default")` (by default "gpt2"). To change the default option use `options(pangoling.causal.default = "newcausalmodel")`.

A list of possible causal models can be found in Hugging Face website.

Using the `config_model` and `config_tokenizer` arguments, it's possible to control how the model and tokenizer from Hugging Face is accessed, see the Python method `from_pretrained` for details.

In case of errors when a new model is run, check the status of `https://status.huggingface.co/`

See Also

Other causal model helper functions: `causal_config()`

Examples

```
causal_preload(model = "gpt2")
```

`causal_words_pred` *Compute predictability using a causal transformer model*

Description

These functions calculate the predictability of words, phrases, or tokens using a causal transformer model.

Usage

```
causal_words_pred(
  x,
  by = rep(1, length(x)),
  word_n = NULL,
  sep = " ",
  log.p = getOption("pangoling.log.p"),
  ignore_regex = "",
  model = getOption("pangoling.causal.default"),
  checkpoint = NULL,
  add_special_tokens = NULL,
  config_model = NULL,
```

```
  config_tokenizer = NULL,
  batch_size = 1,
  ...
)

causal_tokens_pred_lst(
  texts,
  log.p = getOption("pangoling.log.p"),
  model = getOption("pangoling.causal.default"),
  checkpoint = NULL,
  add_special_tokens = NULL,
  config_model = NULL,
  config_tokenizer = NULL,
  batch_size = 1
)

causal_targets_pred(
  contexts,
  targets,
  sep = " ",
  log.p = getOption("pangoling.log.p"),
  ignore_regex = "",
  model = getOption("pangoling.causal.default"),
  checkpoint = NULL,
  add_special_tokens = NULL,
  config_model = NULL,
  config_tokenizer = NULL,
  batch_size = 1,
  ...
)
```

Arguments

x	A character vector of words, phrases, or texts to evaluate.
by	A grouping variable indicating how texts are split into groups.
word_n	Word order, by default this is the word order of the vector x.
sep	A string specifying how words are separated within contexts or groups. Default is `" "`. For languages that don't have spaces between words (e.g., Chinese), set `sep = ""`.
log.p	Base of the logarithm used for the output predictability values. If `TRUE` (default), the natural logarithm (base *e*) is used. If `FALSE`, the raw probabilities are returned. Alternatively, `log.p` can be set to a numeric value specifying the base of the logarithm (e.g., `2` for base-2 logarithms). To get surprisal in bits (rather than predictability), set `log.p = 1/2`.
ignore_regex	Can ignore certain characters when calculating the log probabilities. For example `^[[:punct:]]$` will ignore all punctuation that stands alone in a token.
model	Name of a pre-trained model or folder. One should be able to use models based on "gpt2". See hugging face website.

checkpoint Folder of a checkpoint.

add_special_tokens
Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.

config_model List with other arguments that control how the model from Hugging Face is accessed.

config_tokenizer
List with other arguments that control how the tokenizer from Hugging Face is accessed.

batch_size Maximum number of sentences/texts processed in parallel. Larger batches increase speed but use more memory. Since all texts in a batch must have the same length, shorter ones are padded with placeholder tokens.

... Currently not in use.

texts A vector or list of sentences or paragraphs.

contexts A character vector of contexts corresponding to each target.

targets A character vector of target words or phrases.

Details

These functions calculate the predictability (by default the natural logarithm of the word probability) of words, phrases or tokens using a causal transformer model:

- `causal_targets_pred()`: Evaluates specific target words or phrases based on their given contexts. Use when you have explicit context-target pairs to evaluate, with each target word or phrase paired with a single preceding context.
- `causal_words_pred()`: Computes predictability for all elements of a vector grouped by a specified variable. Use when working with words or phrases split into groups, such as sentences or paragraphs, where predictability is computed for every word or phrase in each group.
- `causal_tokens_pred_lst()`: Computes the predictability of each token in a sentence (or group of sentences) and returns a list of results for each sentence. Use when you want to calculate the predictability of every token in one or more sentences.

See the online article in pangoling website for more examples.

Value

For `causal_targets_pred()` and `causal_words_pred()`, a named numeric vector of predictability scores. For `causal_tokens_pred_lst()`, a list of named numeric vectors, one for each sentence or group.

More details about causal models

A causal language model (also called GPT-like, auto-regressive, or decoder model) is a type of large language model usually used for text-generation that can predict the next word (or more accurately in fact token) based on a preceding context.

If not specified, the causal model used will be the one set in the global option `pangoling.causal.default`, this can be accessed via `getOption("pangoling.causal.default")` (by default "gpt2"). To change the default option use `options(pangoling.causal.default = "newcausalmodel")`.

A list of possible causal models can be found in Hugging Face website.

Using the `config_model` and `config_tokenizer` arguments, it's possible to control how the model and tokenizer from Hugging Face is accessed, see the Python method `from_pretrained` for details.

In case of errors when a new model is run, check the status of `https://status.huggingface.co/`

See Also

Other causal model functions: `causal_next_tokens_pred_tbl()`, `causal_pred_mats()`

Examples

```
# Using causal_targets_pred
causal_targets_pred(
  contexts = c("The apple doesn't fall far from the",
               "Don't judge a book by its"),
  targets = c("tree.", "cover."),
  model = "gpt2"
)

# Using causal_words_pred
causal_words_pred(
  x = df_sent$word,
  by = df_sent$sent_n,
  model = "gpt2"
)

# Using causal_tokens_pred_lst
preds <- causal_tokens_pred_lst(
  texts = c("The apple doesn't fall far from the tree.",
            "Don't judge a book by its cover."),
  model = "gpt2"
)
preds

# Convert the output to a tidy table
suppressPackageStartupMessages(library(tidytable))
map2_dfr(preds, seq_along(preds),
~ data.frame(tokens = names(.x), pred = .x, id = .y))
```

---

df_jaeger14 *Self-Paced Reading Dataset on Chinese Relative Clauses*

Description

This dataset contains data from a self-paced reading experiment on Chinese relative clause comprehension. It is structured to support analysis of reaction times, comprehension accuracy, and surprisal values across various experimental conditions in a 2x2 fully crossed factorial design:

Usage

```
data(df_jaeger14)
```

Format

A tibble with 8,624 rows and 15 variables:

subject Participant identifier, a character vector.

item Trial item number, an integer.

cond Experimental condition, a character vector indicating variations in sentence structure (e.g., "a", "b", "c", "d").

word Chinese word presented in each trial, a character vector.

wordn Position of the word within the sentence, an integer.

rt Reaction time in milliseconds for reading each word, an integer.

region Sentence region or phrase type (e.g., "hd1", "Det+CL"), a character vector.

question Comprehension question associated with the trial, a character vector.

accuracy Binary accuracy score for the comprehension question (1 = correct, 0 = incorrect).

correct_answer Expected correct answer for the comprehension question, a character vector ("Y" or "N").

question_type Type of comprehension question, a character vector.

experiment Name of the experiment, indicating self-paced reading, a character vector.

list Experimental list number, for counterbalancing item presentation, an integer.

sentence Full sentence used in the trial with words marked for analysis, a character vector.

surprisal Model-derived surprisal values for each word, a numeric vector.

Region codes in the dataset (column `region`):

- **N**: Main clause subject (in object-modifications only)
- **V**: Main clause verb (in object-modifications only)
- **Det+CL**: Determiner+classifier
- **Adv**: Adverb
- **VN**: RC-verb+RC-object (subject relatives) or RC-subject+RC-verb (object relatives)
  - Note: These two words were merged into one region after the experiment; they were presented as separate regions during the experiment.
- **FreqP**: Frequency phrase/durational phrase
- **DE**: Relativizer "de"
- **head**: Relative clause head noun
- **hd1**: First word after the head noun
- **hd2**: Second word after the head noun
- **hd3**: Third word after the head noun
- **hd4**: Fourth word after the head noun (only in subject-modifications)

- **hd5**: Fifth word after the head noun (only in subject-modifications)

Notes on reading times (column `rt`):

- The reading time of the relative clause region (e.g., "V-N" or "N-V") was computed by summing up the reading times of the relative clause verb and noun.
- The verb and noun were presented as two separate regions during the experiment.

Details

- **Factor I**: Modification type (subject modification; object modification)
- **Factor II**: Relative clause type (subject relative; object relative)

Condition labels:

- a) subject modification; subject relative
- b) subject modification; object relative
- c) object modification; subject relative
- d) object modification; object relative

Source

Jäger, L., Chen, Z., Li, Q., Lin, C.-J. C., & Vasishth, S. (2015). *The subject-relative advantage in Chinese: Evidence for expectation-based processing*. Journal of Memory and Language, 79–80, 97-120. doi:10.1016/j.jml.2014.10.005

See Also

Other datasets: `df_sent`

Examples

```
# Basic exploration
head(df_jaeger14)

# Summarize reaction times by region
 library(tidytable)
df_jaeger14 |>
  group_by(region) |>
  summarize(mean_rt = mean(rt, na.rm = TRUE))
```

df_sent *Example dataset: Two word-by-word sentences*

Description

This dataset contains two example sentences, split word-by-word. It is structured to demonstrate the use of the pangoling package for processing text data.

Usage

```
df_sent
```

Format

A data frame with 15 rows and 2 columns:

sent_n (integer) Sentence number, indicating which sentence each word belongs to.

word (character) Words from the sentences.

See Also

Other datasets: df_jaeger14

Examples

```
# Load the dataset
data("df_sent")
df_sent
```

installed_py_pangoling *Check if the required Python dependencies for* pangoling *are installed*

Description

This function verifies whether the necessary Python modules (transformers and torch) are available in the current Python environment.

Usage

```
installed_py_pangoling()
```

Value

A logical value: TRUE if both transformers and torch are installed and accessible, otherwise FALSE.

See Also

Other helper functions: `install_py_pangoling()`, `set_cache_folder()`

Examples

```
## Not run:
if (installed_py_pangoling()) {
 message("Python dependencies are installed.")
} else {
 warning("Python dependencies are missing. Please install `torch` and `transformers`.")
}

## End(Not run)
```

---

install_py_pangoling *Install the Python packages needed for* pangoling

---

Description

`install_py_pangoling` function facilitates the installation of Python packages needed for using `pangoling` within an R environment, utilizing the `reticulate` package for managing Python environments. It supports various installation methods, environment settings, and Python versions.

Usage

```
install_py_pangoling(method = c("auto", "virtualenv", "conda"),
                     conda = "auto",
                     version = "default",
                     envname = "r-pangoling",
                     restart_session = TRUE,
                     conda_python_version = NULL,
                     ...,
                     pip_ignore_installed = FALSE,
                     new_env = identical(envname, "r-pangoling"),
                     python_version = NULL)
```

Arguments

`method`	A character vector specifying the environment management method. Options are 'auto', 'virtualenv', and 'conda'. Default is 'auto'.
`conda`	Specifies the conda binary to use. Default is 'auto'.
`version`	The Python version to use. Default is 'default', automatically selected.
`envname`	Name of the virtual environment. Default is 'r-pangoling'.
`restart_session`	Logical, whether to restart the R session after installation. Default is TRUE.

conda_python_version
: Python version for conda environments.

...
: Additional arguments passed to `reticulate::py_install`.

pip_ignore_installed
: Logical, whether to ignore already installed packages. Default is FALSE.

new_env
: Logical, whether to create a new environment if `envname` is 'r-pangoling'. Default is the identity of `envname`.

python_version
: Specifies the Python version for the environment.

Details

This function automatically selects the appropriate method for environment management and Python installation, with a focus on virtual and conda environments. It ensures flexibility in dependency management and Python version control. If a new environment is created, existing environments with the same name are removed.

Value

The function returns `NULL` invisibly, but outputs a message on successful installation.

See Also

Other helper functions: `installed_py_pangoling()`, `set_cache_folder()`

Examples

```
# Install with default settings:
## Not run:
 install_py_pangoling()

## End(Not run)
```

---

masked_config *Returns the configuration of a masked model*

Description

Returns the configuration of a masked model.

Usage

```
masked_config(
  model = getOption("pangoling.masked.default"),
  config_model = NULL
)
```

Arguments

model	Name of a pre-trained model or folder. One should be able to use models based on "bert". See hugging face website.
config_model	List with other arguments that control how the model from Hugging Face is accessed.

Details

A masked language model (also called BERT-like, or encoder model) is a type of large language model that can be used to predict the content of a mask in a sentence.

If not specified, the masked model that will be used is the one set in specified in the global option `pangoling.masked.default`, this can be accessed via `getOption("pangoling.masked.default")` (by default "bert-base-uncased"). To change the default option use `options(pangoling.masked.default = "newmaskedmodel")`.

A list of possible masked can be found in Hugging Face website

Using the `config_model` and `config_tokenizer` arguments, it's possible to control how the model and tokenizer from Hugging Face is accessed, see the python method `from_pretrained` for details. In case of errors check the status of `https://status.huggingface.co/`

Value

A list with the configuration of the model.

See Also

Other masked model helper functions: `masked_preload()`

Examples

```
masked_config(model = "bert-base-uncased")
```

---

masked_preload *Preloads a masked language model*

Description

Preloads a masked language model to speed up next runs.

Usage

```
masked_preload(
  model = getOption("pangoling.masked.default"),
  add_special_tokens = NULL,
  config_model = NULL,
  config_tokenizer = NULL
)
```

Arguments

model	Name of a pre-trained model or folder. One should be able to use models based on "bert". See hugging face website.
add_special_tokens	Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.
config_model	List with other arguments that control how the model from Hugging Face is accessed.
config_tokenizer	List with other arguments that control how the tokenizer from Hugging Face is accessed.

Details

A masked language model (also called BERT-like, or encoder model) is a type of large language model that can be used to predict the content of a mask in a sentence.

If not specified, the masked model that will be used is the one set in specified in the global option `pangoling.masked.default`, this can be accessed via `getOption("pangoling.masked.default")` (by default "bert-base-uncased"). To change the default option use `options(pangoling.masked.default = "newmaskedmodel")`.

A list of possible masked can be found in Hugging Face website

Using the `config_model` and `config_tokenizer` arguments, it's possible to control how the model and tokenizer from Hugging Face is accessed, see the python method `from_pretrained` for details. In case of errors check the status of `https://status.huggingface.co/`

Value

Nothing.

See Also

Other masked model helper functions: `masked_config()`

Examples

```
causal_preload(model = "bert-base-uncased")
```

---

masked_targets_pred *Get the predictability of a target word (or phrase) given a left and right context*

Description

Get the predictability (by default the natural logarithm of the word probability) of a vector of target words (or phrase) given a vector of left and of right contexts using a masked transformer.

Usage

```
masked_targets_pred(
  prev_contexts,
  targets,
  after_contexts,
  log.p = getOption("pangoling.log.p"),
  ignore_regex = "",
  model = getOption("pangoling.masked.default"),
  checkpoint = NULL,
  add_special_tokens = NULL,
  config_model = NULL,
  config_tokenizer = NULL
)
```

Arguments

`prev_contexts`	Left context of the target word in left-to-right written languages.
`targets`	Target words.
`after_contexts`	Right context of the target in left-to-right written languages.
`log.p`	Base of the logarithm used for the output predictability values. If `TRUE` (default), the natural logarithm (base *e*) is used. If `FALSE`, the raw probabilities are returned. Alternatively, `log.p` can be set to a numeric value specifying the base of the logarithm (e.g., `2` for base-2 logarithms). To get surprisal in bits (rather than predictability), set `log.p = 1/2`.
`ignore_regex`	Can ignore certain characters when calculating the log probabilities. For example `^[[:punct:]]$` will ignore all punctuation that stands alone in a token.
`model`	Name of a pre-trained model or folder. One should be able to use models based on "bert". See hugging face website.
`checkpoint`	Folder of a checkpoint.
`add_special_tokens`	Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.
`config_model`	List with other arguments that control how the model from Hugging Face is accessed.
`config_tokenizer`	List with other arguments that control how the tokenizer from Hugging Face is accessed.

Details

A masked language model (also called BERT-like, or encoder model) is a type of large language model that can be used to predict the content of a mask in a sentence.

If not specified, the masked model that will be used is the one set in specified in the global option `pangoling.masked.default`, this can be accessed via `getOption("pangoling.masked.default")` (by default "bert-base-uncased"). To change the default option use `options(pangoling.masked.default = "newmaskedmodel")`.

A list of possible masked can be found in Hugging Face website

Using the `config_model` and `config_tokenizer` arguments, it's possible to control how the model and tokenizer from Hugging Face is accessed, see the python method `from_pretrained` for details. In case of errors check the status of `https://status.huggingface.co/`

Value

A named vector of predictability values (by default the natural logarithm of the word probability).

More examples

See the online article in pangoling website for more examples.

See Also

Other masked model functions: `masked_tokens_pred_tbl()`

Examples

```
masked_targets_pred(
  prev_contexts = c("The", "The"),
  targets = c("apple", "pear"),
  after_contexts = c(
    "doesn't fall far from the tree.",
    "doesn't fall far from the tree."
  ),
  model = "bert-base-uncased"
)
```

---

masked_tokens_pred_tbl

Get the possible tokens and their log probabilities for each mask in a sentence

---

Description

For each mask, indicated with `[MASK]`, in a sentence, get the possible tokens and their predictability (by default the natural logarithm of the word probability) using a masked transformer.

Usage

```
masked_tokens_pred_tbl(
  masked_sentences,
  log.p = getOption("pangoling.log.p"),
  model = getOption("pangoling.masked.default"),
  checkpoint = NULL,
  add_special_tokens = NULL,
```

```
  config_model = NULL,
  config_tokenizer = NULL
)
```

Arguments

masked_sentences	Masked sentences.
log.p	Base of the logarithm used for the output predictability values. If TRUE (default), the natural logarithm (base *e*) is used. If FALSE, the raw probabilities are returned. Alternatively, log.p can be set to a numeric value specifying the base of the logarithm (e.g., 2 for base-2 logarithms). To get surprisal in bits (rather than predictability), set log.p = 1/2.
model	Name of a pre-trained model or folder. One should be able to use models based on "bert". See hugging face website.
checkpoint	Folder of a checkpoint.
add_special_tokens	Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.
config_model	List with other arguments that control how the model from Hugging Face is accessed.
config_tokenizer	List with other arguments that control how the tokenizer from Hugging Face is accessed.

Details

A masked language model (also called BERT-like, or encoder model) is a type of large language model that can be used to predict the content of a mask in a sentence.

If not specified, the masked model that will be used is the one set in specified in the global option pangoling.masked.default, this can be accessed via getOption("pangoling.masked.default") (by default "bert-base-uncased"). To change the default option use options(pangoling.masked.default = "newmaskedmodel").

A list of possible masked can be found in Hugging Face website

Using the config_model and config_tokenizer arguments, it's possible to control how the model and tokenizer from Hugging Face is accessed, see the python method from_pretrained for details. In case of errors check the status of https://status.huggingface.co/

Value

A table with the masked sentences, the tokens (token), predictability (pred), and the respective mask number (mask_n).

More examples

See the online article in pangoling website for more examples.

See Also

Other masked model functions: masked_targets_pred()

Examples

```
masked_tokens_pred_tbl("The [MASK] doesn't fall far from the tree.",
  model = "bert-base-uncased"
)
```

---

ntokens *The number of tokens in a string or vector of strings*

Description

The number of tokens in a string or vector of strings

Usage

```
ntokens(
  x,
  model = getOption("pangoling.causal.default"),
  add_special_tokens = NULL,
  config_tokenizer = NULL
)
```

Arguments

x	character input
model	Name of a pre-trained model or folder. One should be able to use models based on "gpt2". See hugging face website.
add_special_tokens	Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.
config_tokenizer	List with other arguments that control how the tokenizer from Hugging Face is accessed.

Value

The number of tokens in a string or vector of words.

See Also

Other token-related functions: tokenize_lst(), transformer_vocab()

Examples

```
ntokens(x = c("The apple doesn't fall far from the tree."), model = "gpt2")
```

`perplexity_calc` *Calculates perplexity*

Description

Calculates the perplexity of a vector of (log-)probabilities.

Usage

```
perplexity_calc(x, na.rm = FALSE, log.p = TRUE)
```

Arguments

`x`	A vector of log-probabilities.
`na.rm`	Should missing values (including NaN) be removed?
`log.p`	If TRUE (default), x are assumed to be log-transformed probabilities with base e, if FALSE x are assumed to be raw probabilities, alternatively log.p can be the base of other logarithmic transformations.

Details

If x are raw probabilities (NOT the default), then perplexity is calculated as follows:

$$\left(\prod_n x_n\right)^{\frac{1}{N}}$$

Value

The perplexity.

Examples

```
probs <- c(.3, .5, .6)
perplexity_calc(probs, log.p = FALSE)
lprobs <- log(probs)
perplexity_calc(lprobs, log.p = TRUE)
```

`set_cache_folder` *Set cache folder for HuggingFace transformers*

Description

This function sets the cache directory for HuggingFace transformers. If a path is given, the function checks if the directory exists and then sets the `HF_HOME` environment variable to this path. If no path is provided, the function checks for the existing cache directory in a number of environment variables. If none of these environment variables are set, it provides the user with information on the default cache directory.

Usage

```
set_cache_folder(path = NULL)
```

Arguments

`path`	Character string, the path to set as the cache directory. If NULL, the function will look for the cache directory in a number of environment variables. Default is NULL.

Value

Nothing is returned, this function is called for its side effect of setting the `HF_HOME` environment variable, or providing information to the user.

See Also

Installation docs

Other helper functions: `install_py_pangoling()`, `installed_py_pangoling()`

Examples

```
## Not run:
set_cache_folder("~/new_cache_dir")

## End(Not run)
```

`tokenize_lst` *Tokenize an input*

Description

Tokenize a string or token ids.

Usage

```
tokenize_lst(
  x,
  decode = FALSE,
  model = getOption("pangoling.causal.default"),
  add_special_tokens = NULL,
  config_tokenizer = NULL
)
```

Arguments

`x`	Strings or token ids.
`decode`	Logical. If `TRUE`, decodes the tokens into human-readable strings, handling special characters and diacritics. Default is `FALSE`.
`model`	Name of a pre-trained model or folder. One should be able to use models based on "gpt2". See hugging face website.
`add_special_tokens`	Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.
`config_tokenizer`	List with other arguments that control how the tokenizer from Hugging Face is accessed.

Value

A list with tokens

See Also

Other token-related functions: `ntokens()`, `transformer_vocab()`

Examples

```
tokenize_lst(x = c("The apple doesn't fall far from the tree."),
             model = "gpt2")
```

transformer_vocab *Returns the vocabulary of a model*

Description

Returns the (decoded) vocabulary of a model.

Usage

```
transformer_vocab(
  model = getOption("pangoling.causal.default"),
  add_special_tokens = NULL,
  decode = FALSE,
  config_tokenizer = NULL
)
```

Arguments

model	Name of a pre-trained model or folder. One should be able to use models based on "gpt2". See hugging face website.
add_special_tokens	Whether to include special tokens. It has the same default as the AutoTokenizer method in Python.
decode	Logical. If TRUE, decodes the tokens into human-readable strings, handling special characters and diacritics. Default is FALSE.
config_tokenizer	List with other arguments that control how the tokenizer from Hugging Face is accessed.

Value

A vector with the vocabulary of a model.

See Also

Other token-related functions: ntokens(), tokenize_lst()

Examples

```
transformer_vocab(model = "gpt2") |>
 head()
```

Index

∗ **causal model functions**
  causal_next_tokens_pred_tbl, 4
  causal_pred_mats, 6
  causal_words_pred, 9
∗ **causal model helper functions**
  causal_config, 3
  causal_preload, 8
∗ **datasets**
  df_jaeger14, 12
  df_sent, 15
∗ **general functions**
  perplexity_calc, 24
∗ **helper functions**
  install_py_pangoling, 16
  installed_py_pangoling, 15
  set_cache_folder, 25
∗ **masked model functions**
  masked_targets_pred, 19
  masked_tokens_pred_tbl, 21
∗ **masked model helper functions**
  masked_config, 17
  masked_preload, 18
∗ **token-related functions**
  ntokens, 23
  tokenize_lst, 26
  transformer_vocab, 27

causal_config, 3, *9*
causal_next_tokens_pred_tbl, 4, *7*, *12*
causal_pred_mats, *5*, 6, *12*
causal_preload, *3*, 8
causal_targets_pred
        *(*causal_words_pred*)*, 9
causal_tokens_pred_lst
        *(*causal_words_pred*)*, 9
causal_words_pred, *5*, *7*, 9

df_jaeger14, 12, *15*
df_sent, *14*, 15

install_py_pangoling, *16*, 16, *25*
installed_py_pangoling, 15, *17*, *25*

masked_config, 17, *19*
masked_preload, *18*, 18
masked_targets_pred, 19, *23*
masked_tokens_pred_tbl, *21*, 21

ntokens, 23, *26, 27*

perplexity_calc, 24

set_cache_folder, *16, 17*, 25

tokenize_lst, *23*, 26, *27*
transformer_vocab, *23*, *26*, 27