Package 'sbo'

October 14, 2022

Type Package

Title Text Prediction via Stupid Back-Off N-Gram Models

Version 0.5.0

Author Valerio Gherardi

Maintainer Valerio Gherardi <vgherard@sissa.it>

Description Utilities for training and evaluating text predictors based on Stupid Back-Off N-gram models (Brants et al., 2007, <https://www.aclweb.org/anthology/D07-1090/>).

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1.9000

Depends R (>= 3.5.0)

LinkingTo Rcpp, testthat

Imports Rcpp, rlang, tidyr, dplyr, utils, stats, graphics

Suggests ggplot2, knitr, rmarkdown, cli, testthat, covr

SystemRequirements C++11

URL https://vgherard.github.io/sbo/, https://github.com/vgherard/sbo

BugReports https://github.com/vgherard/sbo/issues

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-12-05 19:50:02 UTC

R topics documented:

`as_sbo_dictionary` *Coerce to dictionary*

Description

Coerce objects to `sbo_dictionary` class.

Usage

```
as_sbo_dictionary(x, ...)

## S3 method for class 'character'
as_sbo_dictionary(x, .preprocess = identity, EOS = "", ...)
```

Arguments

`x`	object to be coerced.
`...`	further arguments passed to or from other methods.
`.preprocess`	a function for corpus preprocessing.
`EOS`	a length one character vector listing all (single character) end-of-sentence tokens.

Details

This function is an S3 generic for coercing existing objects to `sbo_dictionary` class objects. Currently, only a method for character vectors is implemented, and this will be described below.

Character vector input: Calling `as_sbo_dictionary(x)` simply decorates the character vector `x` with the class `sbo_dictionary` attribute, and with customizable `.preprocess` and `EOS` attributes.

Value

A sbo_dictionary object.

Author(s)

Valerio Gherardi

Examples

```
dict <- as_sbo_dictionary(c("a","b","c"), .preprocess = tolower, EOS = ".")
```

---

babble *Babble!*

Description

Generate random text based on Stupid Back-off language model.

Usage

```
babble(model, input = NA, n_max = 100L, L = attr(model, "L"))
```

Arguments

model	a sbo_predictor object.
input	a length one character vector. Starting point for babbling! If NA, as by default, a random word is sampled from the model's dictionary.
n_max	a length one integer. Maximum number of words to generate.
L	a length one integer. Number of next-word suggestions from which to sample (see details).

Details

This function generates random text from a Stupid Back-off language model. babble randomly samples one of the top L next word predictions. Text generation stops when an End-Of-Sentence token is encountered, or when the number of generated words exceeds n_max.

Value

A character vector of length one.

Author(s)

Valerio Gherardi

Examples

```
# Babble!
p <- sbo_predictor(twitter_predtable)
set.seed(840) # Set seed for reproducibility
babble(p)
```

`eval_sbo_predictor` *Evaluate Stupid Back-off next-word predictions*

Description

Evaluate next-word predictions based on Stupid Back-off N-gram model on a test corpus.

Usage

```
eval_sbo_predictor(model, test, L = attr(model, "L"))
```

Arguments

`model`	a `sbo_predictor` object.
`test`	a character vector. Perform a single prediction on each entry of this vector (see details).
`L`	Maximum number of predictions for each input sentence (maximum allowed is `attr(model, "L")`)

Details

This function allows to obtain information on the quality of Stupid Back-off model predictions, such as next-word prediction accuracy, or the word-rank distribution of correct prediction, by direct test against a test set corpus. For a reasonable estimate of prediction accuracy, the different entries of the `test` vector should be uncorrelated documents (e.g. separate tweets, as in the `twitter_test` example dataset).

More in detail, `eval_sbo_predictor` performs the following operations:

1. Sample a single sentence from each entry of the character vector `test`.
2. Sample a single N-gram from each sentence obtained in the previous step.
3. Predict next words from the $(N-1)$-gram prefix.
4. Return all predictions, together with the true word completions.

Value

A tibble, containing the input $(N-1)$-grams, the true completions, the predicted completions and a column indicating whether one of the predictions were correct or not.

Author(s)

Valerio Gherardi

Examples

```
# Evaluating next-word predictions from a Stupid Back-off N-gram model
if (suppressMessages(require(dplyr) && require(ggplot2))) {
        p <- sbo_predictor(twitter_predtable)
        set.seed(840) # Set seed for reproducibility
        test <- sample(twitter_test, 500)
        eval <- eval_sbo_predictor(p, test)

        ## Compute three-word accuracies
        eval %>% summarise(accuracy = sum(correct)/n()) # Overall accuracy
        eval %>% # Accuracy for in-sentence predictions
                filter(true != "<EOS>") %>%
                summarise(accuracy = sum(correct) / n())

        ## Make histogram of word-rank distribution for correct predictions
        dict <- attr(twitter_predtable, "dict")
        eval %>%
                filter(correct, true != "<EOS>") %>%
                transmute(rank = match(true, table = dict)) %>%
                ggplot(aes(x = rank)) + geom_histogram(binwidth = 30)
}
```

`kgram_freqs` *k-gram frequency tables*

Description

Get k-gram frequency tables from a training corpus.

Usage

```
kgram_freqs(corpus, N, dict, .preprocess = identity, EOS = "")

sbo_kgram_freqs(corpus, N, dict, .preprocess = identity, EOS = "")

kgram_freqs_fast(corpus, N, dict, erase = "", lower_case = FALSE, EOS = "")

sbo_kgram_freqs_fast(corpus, N, dict, erase = "", lower_case = FALSE, EOS = "")
```

Arguments

`corpus`	a character vector. The training corpus from which to extract k-gram frequencies.
`N`	a length one integer. The maximum order of k-grams for which frequencies are to be extracted.

dict	either a sbo_dictionary object, a character vector, or a formula (see details). The language model dictionary.
.preprocess	a function to apply before k-gram tokenization.
EOS	a length one character vector listing all (single character) end-of-sentence tokens.
erase	a length one character vector. Regular expression matching parts of text to be erased from input. The default removes anything not alphanumeric, white space, apostrophes or punctuation characters (i.e. ".?!:;").
lower_case	a length one logical vector. If TRUE, puts everything to lower case.

Details

These functions extract all k-gram frequency tables from a text corpus up to a specified k-gram order N. These are the building blocks to train any N-gram model. The functions sbo_kgram_freqs() and sbo_kgram_freqs_fast() are aliases for kgram_freqs() and kgram_freqs_fast(), respectively.

The optimized version kgram_freqs_fast(erase = x, lower_case = y) is equivalent to kgram_freqs(.preprocess = preprocess(erase = x, lower_case = y)), but more efficient (both from the speed and memory point of view).

Both kgram_freqs() and kgram_freqs_fast() employ a fixed (user specified) dictionary: any out-of-vocabulary word gets effectively replaced by an "unknown word" token. This is specified through the argument dict, which accepts three types of arguments: a sbo_dictionary object, a character vector (containing the words of the dictionary) or a formula. In this last case, valid formulas can be either max_size ~ V or target ~ f, where V and f represent a dictionary size and a corpus word coverage fraction (of corpus), respectively. This usage of the dict argument allows to build the model dictionary 'on the fly'.

The return value is a "sbo_kgram_freqs" object, i.e. a list of N tibbles, storing frequency counts for each k-gram observed in the training corpus, for k = 1, 2, ..., N. In these tables, words are represented by integer numbers corresponding to their position in the reference dictionary. The special codes 0, length(dictionary)+1 and length(dictionary)+2 correspond to the "Begin-Of-Sentence", "End-Of-Sentence" and "Unknown word" tokens, respectively.

Furthermore, the returned objected has the following attributes:

- N: The highest order of N-grams.
- dict: The reference dictionary, sorted by word frequency.
- .preprocess: The function used for text preprocessing.
- EOS: A length one character vector listing all (single character) end-of-sentence tokens employed in k-gram tokenization.

The .preprocess argument of kgram_freqs allows the user to apply a custom transformation to the training corpus, before kgram tokenization takes place.

The algorithm for k-gram tokenization considers anything separated by (any number of) white spaces (i.e. " ") as a single word. Sentences are split according to end-of-sentence (single character) tokens, as specified by the EOS argument. Additionally text belonging to different entries of the preprocessed input vector which are understood to belong to different sentences.

Nota Bene: It is useful to keep in mind that the function passed through the `.preprocess` argument also captures its enclosing environment, which is by default the environment in which the former was defined. If, for instance, `.preprocess` was defined in the global environment, and the latter binds heavy objects, the resulting `sbo_kgram_freqs` will contain bindings to the same objects. If `sbo_kgram_freqs` is stored out of memory and recalled in another R session, these objects will also be reloaded in memory. For this reason, for non interactive use, it is advisable to avoid using preprocessing functions defined in the global environment (for instance, `base::identity` is preferred to `function(x) x`).

Value

A `sbo_kgram_freqs` object, containing the k-gram frequency tables for k = 1, 2, ..., N.

Author(s)

Valerio Gherardi

Examples

```
# Obtain k-gram frequency table from corpus
## Get k-gram frequencies, for k <= N = 3.
## The dictionary is built on the fly, using the most frequent 1000 words.
freqs <- kgram_freqs(corpus = twitter_train, N = 3, dict = max_size ~ 1000,
                     .preprocess = preprocess, EOS = ".?!:;")
freqs
## Using a predefined dictionary
freqs <- kgram_freqs_fast(twitter_train, N = 3, dict = twitter_dict,
                          erase = "[^.?!:;'\\w\\s]", lower_case = TRUE,
                          EOS = ".?!:;")
freqs
## 2-grams, no preprocessing, use a dictionary covering 50% of corpus
freqs <- kgram_freqs(corpus = twitter_train, N = 2, dict = target ~ 0.5,
                     EOS = ".?!:;")
freqs


# Obtain k-gram frequency table from corpus
freqs <- kgram_freqs_fast(twitter_train, N = 3, dict = twitter_dict)
## Print result
freqs
```

`plot.word_coverage` *Plot method for word_coverage objects*

Description

Plot cumulative corpus coverage fraction of a dictionary.

Usage

```
## S3 method for class 'word_coverage'
plot(
  x,
  include_EOS = FALSE,
  show_limit = TRUE,
  type = "l",
  xlim = c(0, length(x)),
  ylim = c(0, 1),
  xticks = seq(from = 0, to = length(x), by = length(x)/5),
  yticks = seq(from = 0, to = 1, by = 0.25),
  xlab = "Rank",
  ylab = "Covered fraction",
  title = "Cumulative corpus coverage fraction of dictionary",
  subtitle = "_default_",
  ...
)
```

Arguments

x	a word_coverage object.
include_EOS	length one logical. Should End-Of-Sentence tokens be considered in the computation of coverage fraction?
show_limit	length one logical. If TRUE, plots an horizontal line corresponding to the total coverage fraction.
type	what type of plot should be drawn, as detailed in ?plot.
xlim	length two numeric. Extremes of the x-range.
ylim	length two numeric. Extremes of the y-range.
xticks	numeric vector. position of the x-axis ticks.
yticks	numeric vector. position of the y-axis ticks.
xlab	length one character. The x-axis label.
ylab	length one character. The y-axis label.
title	length one character. Plot title.
subtitle	length one character. Plot subtitle; if "*default*", prints dictionary length and total covered fraction.
...	further arguments passed to or from other methods.

Details

This function generates nice plots of cumulative corpus coverage fractions. The x coordinate in the resulting plot is the word rank in the underlying dictionary; the y coordinate at x is the cumulative coverage fraction for rank <= x.

Author(s)

Valerio Gherardi

Examples

```
c <- word_coverage(twitter_dict, twitter_test)
plot(c)
```

`predict.sbo_kgram_freqs`

Predict method for k-gram frequency tables

Description

Predictive text based on Stupid Back-off N-gram model.

Usage

```
## S3 method for class 'sbo_kgram_freqs'
predict(object, input, lambda = 0.4, ...)
```

Arguments

`object`	a `sbo_kgram_freqs` object.
`input`	a length one character vector, containing the input for next-word prediction.
`lambda`	a numeric vector of length one. The back-off penalization in Stupid Back-off algorithm.
`...`	further arguments passed to or from other methods.

Value

A tibble containing the next-word probabilities for all words in the dictionary.

Author(s)

Valerio Gherardi

Examples

```
predict(twitter_freqs, "i love")
```

`predict.sbo_predictor` *Predict method for Stupid Back-off text predictor*

Description

Predictive text based on Stupid Back-off N-gram model.

Usage

```
## S3 method for class 'sbo_predictor'
predict(object, input, ...)
```

Arguments

`object`	a `sbo_predictor` object.
`input`	a character vector, containing the input for next-word prediction.
`...`	further arguments passed to or from other methods.

Details

This method returns the top `L` next-word predictions from a text predictor trained with Stupid Back-Off.

Trying to predict from a `sbo_predtable` results into an error. Instead, one should load a `sbo_predictor` object and use this one to predict(), as shown in the example below.

Value

A character vector if `length(input) == 1`, otherwise a character matrix.

Author(s)

Valerio Gherardi

Examples

```
p <- sbo_predictor(twitter_predtable)
x <- predict(p, "i love")
x
x <- predict(p, "you love")
x
#N.B. the top predictions here are x[1], followed by x[2] and x[3].
predict(p, c("i love", "you love")) # Behaviour with length()>1 input.
```

preprocess *Preprocess text corpus*

Description

A simple text preprocessing utility.

Usage

```
preprocess(input, erase = "[^.?!:;'\\w\\s]", lower_case = TRUE)
```

Arguments

input	a character vector.
erase	a length one character vector. Regular expression matching parts of text to be erased from input. The default removes anything not alphanumeric, white space, apostrophes or punctuation characters (i.e. ".?!:;").
lower_case	a length one logical vector. If TRUE, puts everything to lower case.

Value

a character vector containing the processed output.

Author(s)

Valerio Gherardi

Examples

```
preprocess("Hi @ there! I'm using `sbo`.")
```

prune *Prune k-gram objects*

Description

Prune M-gram frequency tables or Stupid Back-Off prediction tables for an M-gram model to a smaller order N.

Usage

```
prune(object, N, ...)

## S3 method for class 'sbo_kgram_freqs'
prune(object, N, ...)

## S3 method for class 'sbo_predtable'
prune(object, N, ...)
```

Arguments

object	A kgram_freqs or a sbo_predtable class object.
N	a length one positive integer. N-gram order of the new object.
...	further arguments passed to or from other methods.

Details

This generic function provides a helper to prune M-gram frequency tables or M-gram models, represented by sbo_kgram_freqs and sbo_predtable objects respectively, to objects of a smaller N-gram order, N < M. For k-gram frequency objects, frequency tables for k > N are simply dropped. For sbo_predtable's, the predictions coming from the nested N-gram model are instead retained. In both cases, all other other attributes besides k-gram order (such as the corpus preprocessing function, or the lambda penalty in Stupid Back-Off training) are left unchanged.

Value

an object of the same class of the input object.

Author(s)

Valerio Gherardi

Examples

```
# Drop k-gram frequencies for k > 2
freqs <- twitter_freqs
summary(freqs)
freqs <- prune(freqs, N = 2)
summary(freqs)
# Extract a 2-gram model from a larger 3-gram model
pt <- twitter_predtable
summary(pt)
pt <- prune(pt, N = 2)
summary(pt)
```

---

sbo_dictionary *Dictionaries*

Description

Build dictionary from training corpus.

Usage

```
sbo_dictionary(
  corpus,
  max_size = Inf,
  target = 1,
  .preprocess = identity,
  EOS = ""
)

dictionary(
  corpus,
  max_size = Inf,
  target = 1,
  .preprocess = identity,
  EOS = ""
)
```

Arguments

`corpus`	a character vector. The training corpus from which to extract the dictionary.
`max_size`	a length one numeric. If less than `Inf`, only the most frequent `max_size` words are retained in the dictionary.
`target`	a length one numeric between `0` and `1`. If less than one, retains only as many words as needed to cover a fraction `target` of the training corpus.
`.preprocess`	a function for corpus preprocessing. Takes a character vector as input and returns a character vector.
`EOS`	a length one character vector listing all (single character) end-of-sentence tokens.

Details

The function `dictionary()` is an alias for `sbo_dictionary()`.

This function builds a dictionary using the most frequent words in a training corpus. Two pruning criterions can be applied:

1. Dictionary size, as implemented by the `max_size` argument.
2. Target coverage fraction, as implemented by the `target` argument.

If both these criterions imply non-trivial cuts, the most restrictive critierion applies.

The `.preprocess` argument allows the user to apply a custom transformation to the training corpus, before word tokenization. The `EOS` argument allows to specify a set of characters to be identified as End-Of-Sentence tokens (and thus not part of words).

The returned object is a `sbo_dictionary` object, which is a character vector containing words sorted by decreasing corpus frequency. Furthermore, the object stores as attributes the original values of `.preprocess` and `EOS` (i.e. the function used in corpus preprocessing and the End-Of-Sentence characters for sentence tokenization).

Value

A sbo_dictionary object.

Author(s)

Valerio Gherardi

Examples

```
# Extract dictionary from `twitter_train` corpus (all words)
dict <- sbo_dictionary(twitter_train)
# Extract dictionary from `twitter_train` corpus (top 1000 words)
dict <- sbo_dictionary(twitter_train, max_size = 1000)
# Extract dictionary from `twitter_train` corpus (coverage target = 50%)
dict <- sbo_dictionary(twitter_train, target = 0.5)
```

---

sbo_predictions *Stupid Back-off text predictions*

Description

Train a text predictor via Stupid Back-off

Usage

```
sbo_predictor(object, ...)

predictor(object, ...)

## S3 method for class 'character'
sbo_predictor(
  object,
  N,
  dict,
  .preprocess = identity,
  EOS = "",
  lambda = 0.4,
  L = 3L,
  filtered = "<UNK>",
  ...
)

## S3 method for class 'sbo_kgram_freqs'
sbo_predictor(object, lambda = 0.4, L = 3L, filtered = "<UNK>", ...)
```

```
## S3 method for class 'sbo_predtable'
sbo_predictor(object, ...)

sbo_predtable(object, lambda = 0.4, L = 3L, filtered = "<UNK>", ...)

predtable(object, lambda = 0.4, L = 3L, filtered = "<UNK>", ...)

## S3 method for class 'character'
sbo_predtable(
  object,
  lambda = 0.4,
  L = 3L,
  filtered = "<UNK>",
  N,
  dict,
  .preprocess = identity,
  EOS = "",
  ...
)

## S3 method for class 'sbo_kgram_freqs'
sbo_predtable(object, lambda = 0.4, L = 3L, filtered = "<UNK>", ...)
```

Arguments

object	either a character vector or an object inheriting from classes sbo_kgram_freqs or sbo_predtable. Defines the method to use for training.
...	further arguments passed to or from other methods.
N	a length one integer. Order 'N' of the N-gram model.
dict	a sbo_dictionary, a character vector or a formula. For more details see kgram_freqs.
.preprocess	a function for corpus preprocessing. For more details see kgram_freqs.
EOS	a length one character vector. String listing End-Of-Sentence characters. For more details see kgram_freqs.
lambda	a length one numeric. Penalization in the Stupid Back-off algorithm.
L	a length one integer. Maximum number of next-word predictions for a given input (top scoring predictions are retained).
filtered	a character vector. Words to exclude from next-word predictions. The strings '<UNK>' and '<EOS>' are reserved keywords referring to the Unknown-Word and End-Of-Sentence tokens, respectively.

Details

These functions are generics used to train a text predictor with Stupid Back-Off. The functions predictor() and predtable() are aliases for sbo_predictor() and sbo_predtable(), respectively.

The sbo_predictor data structure carries all information required for prediction in a compact and efficient (upon retrieval) way, by directly storing the top L next-word predictions for each k-gram prefix observed in the training corpus.

The sbo_predictor objects are for interactive use. If the training process is computationally heavy, one can store a "raw" version of the text predictor in a sbo_predtable class object, which can be safely saved out of memory (with e.g. save()). The resulting object can be restored in another R session, and the corresponding sbo_predictor object can be loaded rapidly using again the generic constructor sbo_predictor() (see example below).

The returned objects are a sbo_predictor and a sbo_predtable objects. The latter contains Stupid Back-Off prediction tables, storing next-word prediction for each k-gram prefix observed in the text, whereas the former is an external pointer to an equivalent (but processed) C++ structure.

Both objects have the following attributes:

- N: The order of the underlying N-gram model, "N".
- dict: The model dictionary.
- lambda: The penalization used in the Stupid Back-Off algorithm.
- L: The maximum number of next-word predictions for a given text input.
- .preprocess: The function used for text preprocessing.
- EOS: A length one character vector listing all (single character) end-of-sentence tokens.

Value

A sbo_predictor object for sbo_predictor(), a sbo_predtable object for sbo_predtable().

Author(s)

Valerio Gherardi

See Also

predict.sbo_predictor

Examples

```
# Train a text predictor directly from corpus
p <- sbo_predictor(twitter_train, N = 3, dict = max_size ~ 1000,
                   .preprocess = preprocess, EOS = ".?!:;")


# Train a text predictor from previously computed 'kgram_freqs' object
p <- sbo_predictor(twitter_freqs)


# Load a text predictor from a Stupid Back-Off prediction table
p <- sbo_predictor(twitter_predtable)
```

```
# Predict from Stupid Back-Off text predictor
p <- sbo_predictor(twitter_predtable)
predict(p, "i love")


# Build Stupid Back-Off prediction tables directly from corpus
t <- sbo_predtable(twitter_train, N = 3, dict = max_size ~ 1000,
                   .preprocess = preprocess, EOS = ".?!:;")


# Build Stupid Back-Off prediction tables from kgram_freqs object
t <- sbo_predtable(twitter_freqs)

## Not run:
# Save and reload a 'sbo_predtable' object with base::save()
save(t)
load("t.rda")

## End(Not run)
```

`tokenize_sentences` *Sentence tokenizer*

Description

Get sentence tokens from text

Usage

```
tokenize_sentences(input, EOS = ".?!:;")
```

Arguments

`input`	a character vector.
`EOS`	a length one character vector listing all (single character) end-of-sentence tokens.

Value

a character vector, each entry of which corresponds to a single sentence.

Author(s)

Valerio Gherardi

Examples

```
tokenize_sentences("Hi there! I'm using `sbo`.")
```

twitter_dict *Top 1000 dictionary from Twitter training set*

Description

Top 1000 dictionary from Twitter training set

Usage

```
twitter_dict
```

Format

A character vector. Contains the 1000 most frequent words from the example training set `twitter_train`, sorted by word rank.

See Also

`twitter_train`

Examples

```
head(twitter_dict, 10)
```

twitter_freqs *k-gram frequencies from Twitter training set*

Description

k-gram frequencies from Twitter training set

Usage

```
twitter_freqs
```

Format

A `sbo_kgram_freqs` object. Contains k-gram frequencies from the example training set `twitter_train`.

See Also

`twitter_train`

twitter_predtable *Next-word prediction tables from 3-gram model trained on Twitter training set*

Description

Next-word prediction tables from 3-gram model trained on Twitter training set

Usage

```
twitter_predtable
```

Format

A `sbo_predtable` object. Contains prediction tables of a 3-gram Stupid Back-off model trained on the example training set `twitter_train`.

See Also

`twitter_train`

twitter_test *Twitter test set*

Description

Twitter test set

Usage

```
twitter_test
```

Format

A collection of 10'000 Twitter posts in English.

Source

https://www.kaggle.com/crmercado/tweets-blogs-news-swiftkey-dataset-4million

See Also

`twitter_train`

Examples

```
head(twitter_test)
```

twitter_train *Twitter training set*

Description

Twitter training set

Usage

```
twitter_train
```

Format

A collection of 50'000 Twitter posts in English.

Source

https://www.kaggle.com/crmercado/tweets-blogs-news-swiftkey-dataset-4million

See Also

twitter_test, twitter_dict, twitter_freqs, twitter_predtable

Examples

```
head(twitter_train)
```

word_coverage *Word coverage fraction*

Description

Compute total and cumulative corpus coverage fraction of a dictionary.

Usage

```
word_coverage(object, corpus, ...)

## S3 method for class 'sbo_dictionary'
word_coverage(object, corpus, ...)

## S3 method for class 'character'
word_coverage(object, corpus, .preprocess = identity, EOS = "", ...)

## S3 method for class 'sbo_kgram_freqs'
word_coverage(object, corpus, ...)
```

```
## S3 method for class 'sbo_predictions'
word_coverage(object, corpus, ...)
```

Arguments

object	either a character vector, or an object inheriting from one of the classes sbo_dictionary, sbo_kgram_freqs, sbo_predtable or sbo_predictor. The object storing the dictionary for which corpus coverage is to be computed.
corpus	a character vector.
...	further arguments passed to or from other methods.
.preprocess	preprocessing function for training corpus. See kgram_freqs and sbo_dictionary for further details.
EOS	a length one character vector. String containing End-Of-Sentence characters, see kgram_freqs and sbo_dictionary for further details.

Details

This function computes the corpus coverage fraction of a dictionary, that is the fraction of words appearing in corpus which are contained in the original dictionary.

This function is a generic, accepting as object argument any object storing a dictionary, along with a preprocessing function and a list of End-Of-Sentence characters. This includes all sbo main classes: sbo_dictionary, sbo_kgram_freqs, sbo_predtable and sbo_predictor. When object is a character vector, the preprocessing function and the End-Of-Sentence characters must be specified explicitly.

The coverage fraction is computed cumulatively, and the dependence of coverage with respect to maximal rank can be explored through plot() (see examples below)

Value

a word_coverage object.

Author(s)

Valerio Gherardi

See Also

predict.sbo_predictor

Examples

```
c <- word_coverage(twitter_dict, twitter_train)
print(c)
summary(c)
# Plot coverage fraction, including the End-Of-Sentence in word counts.
plot(c, include_EOS = TRUE)
```

Index