

Package ‘textrecipes’

July 2, 2022

Title Extra 'Recipes' for Text Processing

Version 1.0.0

Description Converting text to numerical features requires specifically created procedures, which are implemented as steps according to the 'recipes' package. These steps allows for tokenization, filtering, counting (tf and tfidf) and feature hashing.

License MIT + file LICENSE

URL <https://github.com/tidymodels/textrecipes>,
<https://textrecipes.tidymodels.org>,
<https://textrecipes.tidymodels.org/>

BugReports <https://github.com/tidymodels/textrecipes/issues>

Depends R (>= 3.4), recipes (>= 1.0.0)

Imports lifecycle, dplyr, generics (>= 0.1.0), magrittr, Matrix, purrr, Rcpp, rlang, SnowballC, tibble, tokenizers, vctrs, glue

Suggests covr, janitor, knitr, modeldata, rmarkdown, sentencepiece, spacyr, stopwords, stringi, testthat (>= 3.0.0), text2vec, textfeatures (>= 0.3.3), tokenizers.bpe, udpipe, wordpiece

LinkingTo Rcpp

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate, reticulate

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.0.9000

SystemRequirements GNU make, C++11

NeedsCompilation yes

Author Emil Hvitfeldt [aut, cre] (<<https://orcid.org/0000-0002-0679-1945>>)

Maintainer Emil Hvitfeldt <emilhhvitfeldt@gmail.com>

Repository CRAN

Date/Publication 2022-07-02 07:10:02 UTC

R topics documented:

all_tokenized	2
emoji_samples	3
show_tokens	3
step_clean_levels	4
step_clean_names	6
step_dummy_hash	7
step_lda	10
step_lemma	12
step_ngram	14
step_pos_filter	16
step_sequence_onehot	18
step_stem	20
step_stopwords	22
step_textfeature	24
step_texthash	26
step_text_normalization	29
step_tf	30
step_tfidf	33
step_tokenfilter	35
step_tokenize	38
step_tokenize_bpe	43
step_tokenize_sentencepiece	45
step_tokenize_wordpiece	47
step_tokenmerge	48
step_untokenize	50
step_word_embeddings	52
tidy.step_clean_levels	54
tokenlist	56

Index	58
--------------	-----------

all_tokenized	<i>Role Selection</i>
---------------	-----------------------

Description

all_tokenized() selects all [token](#) variables, all_tokenized_predictors() selects all predictor [token](#) variables.

Usage

```
all_tokenized()
```

```
all_tokenized_predictors()
```

See Also[recipes::has_role\(\)](#)

emoji_samples	<i>Sample sentences with emojis</i>
---------------	-------------------------------------

Description

This data set is primarily used for examples.

Usage

```
emoji_samples
```

Format

tibble with 1 column

show_tokens	<i>Show token output of recipe</i>
-------------	------------------------------------

Description

Returns the tokens as a list of character vector of a recipe. This function can be useful for diagnostics doing recipe construction but should not be used in final recipe steps. Note that this function will both `prep()` and `bake()` the recipe it is used on.

Usage

```
show_tokens(rec, var, n = 6L)
```

Arguments

rec	A recipe object
var	name of variable
n	Number of elements to return.

Value

A list of character vectors

Examples

```

text_tibble <- tibble(text = c("This is words", "They are nice!"))

recipe(~text, data = text_tibble) %>%
  step_tokenize(text) %>%
  show_tokens(text)

library(modeldata)
data(tate_text)

recipe(~., data = tate_text) %>%
  step_tokenize(medium) %>%
  show_tokens(medium)

```

step_clean_levels *Clean Categorical Levels*

Description

step_clean_levels creates a *specification* of a recipe step that will clean nominal data (character or factor) so the levels consist only of letters, numbers, and the underscore.

Usage

```

step_clean_levels(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  clean = NULL,
  skip = FALSE,
  id = rand_id("clean_levels")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
clean	A named character vector to clean and recode categorical levels. This is NULL until computed by recipes::prep.recipe() . Note that if the original variable is a character vector, it will be converted to a factor.

skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

The new levels are cleaned and then reset with `dplyr::recode_factor()`. When data to be processed contains novel levels (i.e., not contained in the training set), they are converted to missing.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected), `original` (the original levels) and `value` (the cleaned levels) is returned.

Case weights

The underlying operation does not allow for case weights.

See Also

[step_clean_names\(\)](#), [recipes::step_factor2string\(\)](#), [recipes::step_string2factor\(\)](#), [recipes::step_regex\(\)](#), [recipes::step_unknown\(\)](#), [recipes::step_novel\(\)](#), [recipes::step_other\(\)](#)

Other Steps for Text Cleaning: [step_clean_names\(\)](#)

Examples

```
library(recipes)
library(modeldata)
data(Smithsonian)

smith_tr <- Smithsonian[1:15, ]
smith_te <- Smithsonian[16:20, ]

rec <- recipe(~., data = smith_tr)

if (requireNamespace("janitor", quietly = TRUE)) {
  rec <- rec %>%
    step_clean_levels(name)
  rec <- prep(rec, training = smith_tr)

  cleaned <- bake(rec, smith_tr)

  tidy(rec, number = 1)

  # novel levels are replaced with missing
```

```
bake(rec, smith_te)
}
```

step_clean_names	<i>Clean Variable Names</i>
------------------	-----------------------------

Description

step_clean_names creates a *specification* of a recipe step that will clean variable names so the names consist only of letters, numbers, and the underscore.

Usage

```
step_clean_names(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  clean = NULL,
  skip = FALSE,
  id = rand_id("clean_names")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
clean	A named character vector to clean variable names. This is NULL until computed by recipes::prep.recipe() .
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = FALSE.
id	A character string that is unique to this step to identify it.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you [tidy\(\)](#) this step, a tibble with columns terms (the new clean variable names) and value (the original variable names).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_clean_levels\(\)](#), [recipes::step_factor2string\(\)](#), [recipes::step_string2factor\(\)](#), [recipes::step_regex\(\)](#), [recipes::step_unknown\(\)](#), [recipes::step_novel\(\)](#), [recipes::step_other\(\)](#)

Other Steps for Text Cleaning: [step_clean_levels\(\)](#)

Examples

```
library(recipes)
data(airquality)

air_tr <- tibble(airquality[1:100, ])
air_te <- tibble(airquality[101:153, ])

rec <- recipe(~., data = air_tr)

if (requireNamespace("janitor", quietly = TRUE)) {
  rec <- rec %>%
    step_clean_names(all_predictors())
  rec <- prep(rec, training = air_tr)
  tidy(rec, number = 1)

  bake(rec, air_tr)
  bake(rec, air_te)
}
```

step_dummy_hash

Indicator Variables via Feature Hashing

Description

`step_dummy_hash` creates a *specification* of a recipe step that will convert factors or character columns into a series of binary (or signed binary) indicator columns.

Usage

```
step_dummy_hash(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  signed = TRUE,
  num_terms = 32L,
  collapse = FALSE,
```

```

  prefix = "dummyhash",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("dummy_hash")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
signed	A logical, indicating whether to use a signed hash-function (generating values of -1, 0, or 1), to reduce collisions when hashing. Defaults to TRUE.
num_terms	An integer, the number of variables to output. Defaults to 32.
collapse	A logical; should all of the selected columns be collapsed into a single column to create a single set of hashed features?
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

Feature hashing, or the hashing trick, is a transformation of a text variable into a new set of numerical variables. This is done by applying a hashing function over the values of the factor levels and using the hash values as feature indices. This allows for a low memory representation of the data and can be very helpful when a qualitative predictor has many levels or is expected to have new levels during prediction. This implementation is done using the MurmurHash3 method.

The argument `num_terms` controls the number of indices that the hashing function will map to. This is the tuning parameter for this transformation. Since the hashing function can map two different tokens to the same index, will a higher value of `num_terms` result in a lower chance of collision.

The new components will have names that begin with `prefix`, then the name of the variable, followed by the tokens all separated by `-`. The variable names are padded with zeros. For example if `prefix = "hash"`, and if `num_terms < 10`, their names will be `hash1 - hash9`. If `num_terms = 101`, their names will be `hash001 - hash101`.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected), `value` (whether a signed hashing was performed), `num_terms` (number of terms), and `collapse` (where columns collapsed).

Case weights

The underlying operation does not allow for case weights.

References

Kilian Weinberger; Anirban Dasgupta; John Langford; Alex Smola; Josh Attenberg (2009).

Kuhn and Johnson (2019), Chapter 7, <https://bookdown.org/max/FES/encoding-predictors-with-many-categories.html>

See Also

[recipes::step_dummy\(\)](#)

Other Steps for Numeric Variables From Characters: [step_sequence_onehot\(\)](#), [step_textfeature\(\)](#)

Examples

```
if (requireNamespace("text2vec", quietly = TRUE)) {  
  library(recipes)  
  library(modeldata)  
  data(grants)  
  
  grants_rec <- recipe(~sponsor_code, data = grants_other) %>%  
    step_dummy_hash(sponsor_code)  
  
  grants_obj <- grants_rec %>%  
    prep()  
  
  bake(grants_obj, grants_test)  
  
  tidy(grants_rec, number = 1)  
  tidy(grants_obj, number = 1)  
}
```

step_lda

Calculate LDA Dimension Estimates of Tokens

Description

step_lda creates a *specification* of a recipe step that will return the lda dimension estimates of a text variable.

Usage

```
step_lda(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  lda_models = NULL,
  num_topics = 10L,
  prefix = "lda",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("lda")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
lda_models	A WarpLDA model object from the text2vec package. If left to NULL, the default, will it train its model based on the training data. Look at the examples for how to fit a WarpLDA model.
num_topics	integer desired number of latent topics.
prefix	A prefix for generated column names, default to "lda".
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.

skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `num_topics` (number of topics).

Case weights

The underlying operation does not allow for case weights.

Source

<https://arxiv.org/abs/1301.3781>

See Also

Other Steps for Numeric Variables From Tokens: `step_texthash()`, `step_tfidf()`, `step_tf()`, `step_word_embeddings()`

Examples

```
if (requireNamespace("text2vec", quietly = TRUE)) {

  library(recipes)
  library(modeldata)
  data(tate_text)

  tate_rec <- recipe(~., data = tate_text) %>%
    step_tokenize(medium) %>%
    step_lda(medium)

  tate_obj <- tate_rec %>%
    prep()

  bake(tate_obj, new_data = NULL) %>%
    slice(1:2)
  tidy(tate_rec, number = 2)
  tidy(tate_obj, number = 2)

  # Changing the number of topics.
  recipe(~., data = tate_text) %>%
    step_tokenize(medium, artist) %>%
```

```

step_lda(medium, artist, num_topics = 20) %>%
  prep() %>%
  bake(new_data = NULL) %>%
  slice(1:2)

# Supplying A pre-trained LDA model trained using text2vec
library(text2vec)
tokens <- word_tokenizer(tolower(tate_text$medium))
it <- itoken(tokens, ids = seq_along(tate_text$medium))
v <- create_vocabulary(it)
dtm <- create_dtm(it, vocab_vectorizer(v))
lda_model <- LDA$new(n_topics = 15)

recipe(~., data = tate_text) %>%
  step_tokenize(medium, artist) %>%
  step_lda(medium, artist, lda_models = lda_model) %>%
  prep() %>%
  bake(new_data = NULL) %>%
  slice(1:2)
}

```

step_lemma

Lemmatization of Token Variables

Description

step_lemma creates a *specification* of a recipe step that will extract the lemmatization of a [token](#) variable.

Usage

```

step_lemma(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("lemma")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.

role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> .
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

This stem doesn't perform lemmatization by itself, but rather lets you extract the lemma attribute of the `token` variable. To be able to use `step_lemma` you need to use a tokenization method that includes lemmatization. Currently using the "spacyr" engine in `step_tokenize()` provides lemmatization and works well with `step_lemma`.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected).

Case weights

The underlying operation does not allow for case weights.

See Also

`step_tokenize()` to turn characters into `tokens`

Other Steps for Token Modification: `step_ngram()`, `step_pos_filter()`, `step_stem()`, `step_stopwords()`, `step_tokenfilter()`, `step_tokenmerge()`

Examples

```
## Not run:
library(recipes)

short_data <- data.frame(text = c(
  "This is a short tale,",
  "With many cats and ladies."
))

rec_spec <- recipe(~text, data = short_data) %>%
  step_tokenize(text, engine = "spacyr") %>%
  step_lemma(text) %>%
  step_tf(text)
```

```

rec_prepped <- prep(rec_spec)

bake(rec_prepped, new_data = NULL)

## End(Not run)

```

step_ngram

Generate n-grams From Token Variables

Description

step_ngram creates a *specification* of a recipe step that will convert a [token](#) variable into a [token](#) variable of ngrams.

Usage

```

step_ngram(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  num_tokens = 3L,
  min_num_tokens = 3L,
  delim = "_",
  skip = FALSE,
  id = rand_id("ngram")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
num_tokens	The number of tokens in the n-gram. This must be an integer greater than or equal to 1. Defaults to 3.
min_num_tokens	The minimum number of tokens in the n-gram. This must be an integer greater than or equal to 1 and smaller than n. Defaults to 3.
delim	The separator between words in an n-gram. Defaults to "_".

skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

The use of this step will leave the ordering of the tokens meaningless. If `min_num_tokens < num_tokens` then the tokens order in increasing fashion with respect to the number of tokens in the n-gram. If `min_num_tokens = 1` and `num_tokens = 3` then the output contains all the 1-grams followed by all the 2-grams followed by all the 3-grams.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_tokenize\(\)](#) to turn characters into `tokens`

Other Steps for Token Modification: [step_lemma\(\)](#), [step_pos_filter\(\)](#), [step_stem\(\)](#), [step_stopwords\(\)](#), [step_tokenfilter\(\)](#), [step_tokenmerge\(\)](#)

Examples

```
library(recipes)
library(modeldata)
data(tate_text)

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(medium) %>%
  step_ngram(medium)

tate_obj <- tate_rec %>%
  prep()

bake(tate_obj, new_data = NULL, medium) %>%
  slice(1:2)

bake(tate_obj, new_data = NULL) %>%
  slice(2) %>%
  pull(medium)
```

```
tidy(tate_rec, number = 2)
tidy(tate_obj, number = 2)
```

step_pos_filter	<i>Part of Speech Filtering of Token Variables</i>
-----------------	--

Description

`step_pos_filter` creates a *specification* of a recipe step that will filter a `token` variable based on part of speech tags.

Usage

```
step_pos_filter(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  keep_tags = "NOUN",
  skip = FALSE,
  id = rand_id("pos_filter")
)
```

Arguments

<code>recipe</code>	A recipe object. The step will be added to the sequence of operations for this recipe.
<code>...</code>	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
<code>role</code>	Not used by this step since no new variables are created.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character string of variable names that will be populated (eventually) by the <code>terms</code> argument. This is <code>NULL</code> until the step is trained by recipes::prep.recipe() .
<code>keep_tags</code>	Character variable of part of speech tags to keep. See details for complete list of tags. Defaults to "NOUN".
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
<code>id</code>	A character string that is unique to this step to identify it.

Details

Possible part of speech tags for spacyr engine are: "ADJ", "ADP", "ADV", "AUX", "CONJ", "CCONJ", "DET", "INTJ", "NOUN", "NUM", "PART", "PRON", "PROPN", "PUNCT", "SCONJ", "SYM", "VERB", "X" and "SPACE". For more information look here <https://github.com/explosion/spaCy/blob/master/spacy/glossary.py>.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `num_topics` (number of topics).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_tokenize\(\)](#) to turn characters into `tokens`

Other Steps for Token Modification: [step_lemma\(\)](#), [step_ngram\(\)](#), [step_stem\(\)](#), [step_stopwords\(\)](#), [step_tokenfilter\(\)](#), [step_tokenmerge\(\)](#)

Examples

```
## Not run:
library(recipes)

short_data <- data.frame(text = c(
  "This is a short tale,",
  "With many cats and ladies."
))

rec_spec <- recipe(~text, data = short_data) %>%
  step_tokenize(text, engine = "spacyr") %>%
  step_pos_filter(text, keep_tags = "NOUN") %>%
  step_tf(text)

rec_prepped <- prep(rec_spec)

bake(rec_prepped, new_data = NULL)

## End(Not run)
```

step_sequence_onehot *Positional One-Hot encoding of Tokens*

Description

step_sequence_onehot creates a *specification* of a recipe step that will take a string and do one hot encoding for each character by position.

Usage

```
step_sequence_onehot(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  sequence_length = 100,
  padding = "pre",
  truncating = "pre",
  vocabulary = NULL,
  prefix = "seq1hot",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("sequence_onehot")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
sequence_length	A numeric, number of characters to keep before discarding. Defaults to 100.
padding	'pre' or 'post', pad either before or after each sequence. defaults to 'pre'.
truncating	'pre' or 'post', remove values from sequences larger than sequence_length either in the beginning or in the end of the sequence. Defaults too 'pre'.
vocabulary	A character vector, characters to be mapped to integers. Characters not in the vocabulary will be encoded as 0. Defaults to letters.

prefix	A prefix for generated column names, default to "seq1hot".
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

The string will be capped by the `sequence_length` argument, strings shorter than `sequence_length` will be padded with empty characters. The encoding will assign an integer to each character in the vocabulary, and will encode accordingly. Characters not in the vocabulary will be encoded as 0.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected), `vocabulary` (index) and `token` (text corresponding to the index).

Case weights

The underlying operation does not allow for case weights.

Source

<https://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf>

See Also

Other Steps for Numeric Variables From Characters: `step_dummy_hash()`, `step_textfeature()`

Examples

```
library(recipes)
library(modeldata)
data(tate_text)

tate_rec <- recipe(~medium, data = tate_text) %>%
  step_tokenize(medium) %>%
  step_tokenfilter(medium) %>%
  step_sequence_onehot(medium)

tate_obj <- tate_rec %>%
  prep()
```

```
bake(tate_obj, new_data = NULL)

tidy(tate_rec, number = 3)
tidy(tate_obj, number = 3)
```

step_stem

Stemming of Token Variables

Description

step_stem creates a *specification* of a recipe step that will convert a [token](#) variable to have its stemmed version.

Usage

```
step_stem(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  options = list(),
  custom_stemmer = NULL,
  skip = FALSE,
  id = rand_id("stem")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
options	A list of options passed to the stemmer function.
custom_stemmer	A custom stemming function. If none is provided it will default to "SnowballC".
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = FALSE.
id	A character string that is unique to this step to identify it.

Details

Words tend to have different forms depending on context, such as organize, organizes, and organizing. In many situations it is beneficial to have these words condensed into one to allow for a smaller pool of words. Stemming is the act of chopping off the end of words using a set of heuristics.

Note that the stemming will only be done at the end of the word and will therefore not work reliably on ngrams or sentences.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `is_custom_stemmer` (indicate if custom stemmer was used).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_tokenize\(\)](#) to turn characters into `tokens`

Other Steps for Token Modification: [step_lemma\(\)](#), [step_ngram\(\)](#), [step_pos_filter\(\)](#), [step_stopwords\(\)](#), [step_tokenfilter\(\)](#), [step_tokenmerge\(\)](#)

Examples

```
library(recipes)
library(modeldata)
data(tate_text)

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(medium) %>%
  step_stem(medium)

tate_obj <- tate_rec %>%
  prep()

bake(tate_obj, new_data = NULL, medium) %>%
  slice(1:2)

bake(tate_obj, new_data = NULL) %>%
  slice(2) %>%
  pull(medium)

tidy(tate_rec, number = 2)
tidy(tate_obj, number = 2)

# Using custom stemmer. Here a custom stemmer that removes the last letter
```

```

# if it is a "s".
remove_s <- function(x) gsub("s$", "", x)

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(medium) %>%
  step_stem(medium, custom_stemmer = remove_s)

tate_obj <- tate_rec %>%
  prep()

bake(tate_obj, new_data = NULL, medium) %>%
  slice(1:2)

bake(tate_obj, new_data = NULL) %>%
  slice(2) %>%
  pull(medium)

```

step_stopwords

Filtering of Stop Words for Tokens Variables

Description

step_stopwords creates a *specification* of a recipe step that will filter a [token](#) variable for stop words.

Usage

```

step_stopwords(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  language = "en",
  keep = FALSE,
  stopword_source = "snowball",
  custom_stopword_source = NULL,
  skip = FALSE,
  id = rand_id("stopwords")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.

trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> .
language	A character to indicate the language of stop words by ISO 639-1 coding scheme.
keep	A logical. Specifies whether to keep the stop words or discard them.
stopword_source	A character to indicate the stop words source as listed in <code>stopwords::stopwords_getsources</code> .
custom_stopword_source	A character vector to indicate a custom list of words that cater to the users specific problem.
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

Stop words are words which sometimes are remove before natural language processing tasks. While stop words usually refers to the most common words in the language there is no universal stop word list.

The argument `custom_stopword_source` allows you to pass a character vector to filter against. With the `keep` argument one can specify to keep the words instead of removing thus allowing you to select words with a combination of these two arguments.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected), `value` (name of stop word list), and `keep` (whether stop words are removed or kept).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_tokenize\(\)](#) to turn characters into `tokens`

Other Steps for Token Modification: [step_lemma\(\)](#), [step_ngram\(\)](#), [step_pos_filter\(\)](#), [step_stem\(\)](#), [step_tokenfilter\(\)](#), [step_tokenmerge\(\)](#)

Examples

```

library(recipes)
library(modeldata)
data(tate_text)

if (requireNamespace("stopwords", quietly = TRUE)) {
  tate_rec <- recipe(~., data = tate_text) %>%
    step_tokenize(medium) %>%
    step_stopwords(medium)

  tate_obj <- tate_rec %>%
    prep()

  bake(tate_obj, new_data = NULL, medium) %>%
    slice(1:2)

  bake(tate_obj, new_data = NULL) %>%
    slice(2) %>%
    pull(medium)

  tidy(tate_rec, number = 2)
  tidy(tate_obj, number = 2)
}

# With a custom stop words list

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(medium) %>%
  step_stopwords(medium, custom_stopword_source = c("twice", "upon"))
tate_obj <- tate_rec %>%
  prep(traimomg = tate_text)

bake(tate_obj, new_data = NULL) %>%
  slice(2) %>%
  pull(medium)

```

step_textfeature

Calculate Set of Text Features

Description

step_textfeature creates a *specification* of a recipe step that will extract a number of numeric features of a text column.

Usage

```

step_textfeature(
  recipe,
  ...,

```



```

    role = "predictor",
    trained = FALSE,
    columns = NULL,
    extract_functions = textfeatures::count_functions,
    prefix = "textfeature",
    keep_original_cols = FALSE,
    skip = FALSE,
    id = rand_id("textfeature")
  )

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
extract_functions	A named list of feature extracting functions. default to count_functions from the textfeatures package. See details for more information.
prefix	A prefix for generated column names, default to "textfeature".
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = FALSE.
id	A character string that is unique to this step to identify it.

Details

This step will take a character column and returns a number of numeric columns equal to the number of functions in the list passed to the `extract_functions` argument. The default is a list of functions from the textfeatures package.

All the functions passed to `extract_functions` must take a character vector as input and return a numeric vector of the same length, otherwise an error will be thrown.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `functions` (name of feature functions).

Case weights

The underlying operation does not allow for case weights.

See Also

Other Steps for Numeric Variables From Characters: [step_dummy_hash\(\)](#), [step_sequence_onehot\(\)](#)

Examples

```
if (requireNamespace("textfeatures", quietly = TRUE)) {
  library(recipes)
  library(modeldata)
  data(tate_text)

  tate_rec <- recipe(~., data = tate_text) %>%
    step_textfeature(medium)

  tate_obj <- tate_rec %>%
    prep()

  bake(tate_obj, new_data = NULL) %>%
    slice(1:2)

  bake(tate_obj, new_data = NULL) %>%
    pull(textfeature_medium_n_words)

  tidy(tate_rec, number = 1)
  tidy(tate_obj, number = 1)

  # Using custom extraction functions
  nchar_round_10 <- function(x) round(nchar(x) / 10) * 10

  recipe(~., data = tate_text) %>%
    step_textfeature(medium,
      extract_functions = list(nchar10 = nchar_round_10)
    ) %>%
    prep() %>%
    bake(new_data = NULL)
}
```

Description

step_texthash creates a *specification* of a recipe step that will convert a [token](#) variable into multiple numeric variables using the hashing trick.

Usage

```
step_texthash(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  signed = TRUE,
  num_terms = 1024L,
  prefix = "texthash",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("texthash")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
signed	A logical, indicating whether to use a signed hash-function to reduce collisions when hashing. Defaults to TRUE.
num_terms	An integer, the number of variables to output. Defaults to 1024.
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = FALSE.
id	A character string that is unique to this step to identify it.

Details

Feature hashing, or the hashing trick, is a transformation of a text variable into a new set of numerical variables. This is done by applying a hashing function over the tokens and using the hash values as feature indices. This allows for a low memory representation of the text. This implementation is done using the MurmurHash3 method.

The argument `num_terms` controls the number of indices that the hashing function will map to. This is the tuning parameter for this transformation. Since the hashing function can map two different tokens to the same index, will a higher value of `num_terms` result in a lower chance of collision.

The new components will have names that begin with `prefix`, then the name of the variable, followed by the tokens all separated by `-`. The variable names are padded with zeros. For example if `prefix = "hash"`, and if `num_terms < 10`, their names will be `hash1 - hash9`. If `num_terms = 101`, their names will be `hash001 - hash101`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `value` (number of terms).

@template case-weights-not-supported

References

Kilian Weinberger; Anirban Dasgupta; John Langford; Alex Smola; Josh Attenberg (2009).

See Also

`step_tokenize()` to turn characters into `tokens` `step_text_normalization()` to perform text normalization.

Other Steps for Numeric Variables From Tokens: `step_lda()`, `step_tfidf()`, `step_tf()`, `step_word_embeddings()`

Examples

```
if (requireNamespace("text2vec", quietly = TRUE)) {
  library(recipes)
  library(modeldata)
  data(tate_text)

  tate_rec <- recipe(~., data = tate_text) %>%
    step_tokenize(medium) %>%
    step_tokenfilter(medium, max_tokens = 10) %>%
    step_texthash(medium)

  tate_obj <- tate_rec %>%
    prep()

  bake(tate_obj, tate_text)
```

```

  tidy(tate_rec, number = 3)
  tidy(tate_obj, number = 3)
}

```

step_text_normalization

Normalization of Character Variables

Description

step_text_normalization creates a *specification* of a recipe step that will perform Unicode Normalization on chracter variables.

Usage

```

step_text_normalization(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  normalization_form = "nfc",
  skip = FALSE,
  id = rand_id("text_normalization")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
normalization_form	A single character string determining the Unicode Normalization. Must be one of "nfc", "nfd", "nfkd", "nfkc", or "nfkc_casefold". Defaults to "nfc". See stringi::stri_trans_nfc() for more details.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = FALSE.
id	A character string that is unique to this step to identify it.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `normalization_form` (type of normalization).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_texthash\(\)](#) for feature hashing.

Examples

```
if (requireNamespace("stringi", quietly = TRUE)) {
  library(recipes)

  sample_data <- tibble(text = c("sch\u00f6n", "scho\u00308n"))

  rec <- recipe(~., data = sample_data) %>%
    step_text_normalization(text)

  prepped <- rec %>%
    prep()

  bake(prepped, new_data = NULL, text) %>%
    slice(1:2)

  bake(prepped, new_data = NULL) %>%
    slice(2) %>%
    pull(text)

  tidy(rec, number = 1)
  tidy(prepped, number = 1)
}
```

step_tf

Term frequency of Tokens

Description

`step_tf` creates a *specification* of a recipe step that will convert a `token` variable into multiple variables containing the token counts.

Usage

```

step_tf(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  weight_scheme = "raw count",
  weight = 0.5,
  vocabulary = NULL,
  res = NULL,
  prefix = "tf",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("tf")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
weight_scheme	A character determining the weighting scheme for the term frequency calculations. Must be one of "binary", "raw count", "term frequency", "log normalization" or "double normalization". Defaults to "raw count".
weight	A numeric weight used if weight_scheme is set to "double normalization". Defaults to 0.5.
vocabulary	A character vector of strings to be considered.
res	The words that will be used to calculate the term frequency will be stored here once this preprocessing step has be trained by prep.recipe() .
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = FALSE.
id	A character string that is unique to this step to identify it.

Details

It is strongly advised to use [step_tokenfilter](#) before using [step_tf](#) to limit the number of variables created, otherwise you might run into memory issues. A good strategy is to start with a low token count and go up according to how much RAM you want to use.

Term frequency is a weight of how many times each token appear in each observation. There are different ways to calculate the weight and this step can do it in a couple of ways. Setting the argument `weight_scheme` to "binary" will result in a set of binary variables denoting if a token is present in the observation. "raw count" will count the times a token is present in the observation. "term frequency" will divide the count with the total number of words in the document to limit the effect of the document length as longer documents tends to have the word present more times but not necessarily at a higher percentage. "log normalization" takes the log of 1 plus the count, adding 1 is done to avoid taking log of 0. Finally "double normalization" is the raw frequency divided by the raw frequency of the most occurring term in the document. This is then multiplied by `weight` and `weight` is added to the result. This is again done to prevent a bias towards longer documents.

The new components will have names that begin with `prefix`, then the name of the variable, followed by the tokens all separated by `-`. The variable names are padded with zeros. For example if `prefix = "hash"`, and if `num_terms < 10`, their names will be `hash1 - hash9`. If `num_terms = 101`, their names will be `hash001 - hash101`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `value` (the weighting scheme).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_tokenize\(\)](#) to turn characters into `tokens`

Other Steps for Numeric Variables From Tokens: [step_lda\(\)](#), [step_texthash\(\)](#), [step_tfidf\(\)](#), [step_word_embeddings\(\)](#)

Examples

```
library(recipes)
library(modeldata)
data(tate_text)

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(terms) %>%
  step_tf(terms)
```



```
tate_obj <- tate_rec %>%
  prep()

bake(tate_obj, tate_text)

tidy(tate_rec, number = 2)
tidy(tate_obj, number = 2)
```

step_tfidf

Term Frequency-Inverse Document Frequency of Tokens

Description

step_tfidf creates a *specification* of a recipe step that will convert a [token](#) variable into multiple variables containing the term frequency-inverse document frequency of tokens.

Usage

```
step_tfidf(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  vocabulary = NULL,
  res = NULL,
  smooth_idf = TRUE,
  norm = "l1",
  sublinear_tf = FALSE,
  prefix = "tfidf",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("tfidf")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.

columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> .
vocabulary	A character vector of strings to be considered.
res	The words that will be used to calculate the term frequency will be stored here once this preprocessing step has been trained by <code>prep.recipe()</code> .
smooth_idf	TRUE smooth IDF weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. This prevents division by zero.
norm	A character, defines the type of normalization to apply to term vectors. "l1" by default, i.e., scale by the number of words in the document. Must be one of c("l1", "l2", "none").
sublinear_tf	A logical, apply sublinear term-frequency scaling, i.e., replace the term frequency with $1 + \log(\text{TF})$. Defaults to FALSE.
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

It is strongly advised to use `step_tokenfilter` before using `step_tfidf` to limit the number of variables created; otherwise you may run into memory issues. A good strategy is to start with a low token count and increase depending on how much RAM you want to use.

Term frequency-inverse document frequency is the product of two statistics: the term frequency (TF) and the inverse document frequency (IDF).

Term frequency measures how many times each token appears in each observation.

Inverse document frequency is a measure of how informative a word is, e.g., how common or rare the word is across all the observations. If a word appears in all the observations it might not give that much insight, but if it only appears in some it might help differentiate between observations.

The IDF is defined as follows: $\text{idf} = \log(1 + (\# \text{ documents in the corpus}) / (\# \text{ documents where the term appears}))$

The new components will have names that begin with `prefix`, then the name of the variable, followed by the tokens all separated by `-`. The variable names are padded with zeros. For example if `prefix = "hash"`, and if `num_terms < 10`, their names will be `hash1 - hash9`. If `num_terms = 101`, their names will be `hash001 - hash101`.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected), `token` (name of the tokens), `weight` (the calculated IDF weight) is returned.

Case weights

The underlying operation does not allow for case weights.

See Also

[step_tokenize\(\)](#) to turn characters into `tokens`

Other Steps for Numeric Variables From Tokens: [step_lda\(\)](#), [step_texthash\(\)](#), [step_tf\(\)](#), [step_word_embeddings\(\)](#)

Examples

```
library(recipes)
library(modeldata)
data(tate_text)

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(medium) %>%
  step_tfidf(medium)

tate_obj <- tate_rec %>%
  prep()

bake(tate_obj, tate_text)

tidy(tate_rec, number = 2)
tidy(tate_obj, number = 2)
```

step_tokenfilter

Filter Tokens Based on Term Frequency

Description

`step_tokenfilter` creates a *specification* of a recipe step that will convert a `token` variable to be filtered based on frequency.

Usage

```

step_tokenfilter(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  max_times = Inf,
  min_times = 0,
  percentage = FALSE,
  max_tokens = 100,
  filter_fun = NULL,
  res = NULL,
  skip = FALSE,
  id = rand_id("tokenfilter")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
max_times	An integer. Maximal number of times a word can appear before getting removed.
min_times	An integer. Minimum number of times a word can appear before getting removed.
percentage	A logical. Should max_times and min_times be interpreted as a percentage instead of count.
max_tokens	An integer. Will only keep the top max_tokens tokens after filtering done by max_times and min_times. Defaults to 100.
filter_fun	A function. This function should take a vector of characters, and return a logical vector of the same length. This function will be applied to each observation of the data set. Defaults to NULL. All other arguments will be ignored if this argument is used.
res	The words that will be keep will be stored here once this preprocessing step has been trained by prep.recipe() .
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = FALSE.
id	A character string that is unique to this step to identify it.

Details

This step allow you to limit the tokens you are looking at by filtering on their occurrence in the corpus. You are able to exclude tokens if they appear too many times or too few times in the data. It can be specified as counts using `max_times` and `min_times` or as percentages by setting `percentage` as `TRUE`. In addition one can filter to only use the top `max_tokens` used tokens. If `max_tokens` is set to `Inf` then all the tokens will be used. This will generally lead to very large data sets when then tokens are words or trigrams. A good strategy is to start with a low token count and go up according to how much RAM you want to use.

It is strongly advised to filter before using [step_tf](#) or [step_tfidf](#) to limit the number of variables created.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `value` (number of unique tokens).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_tokenize\(\)](#) to turn characters into `tokens`

Other Steps for Token Modification: [step_lemma\(\)](#), [step_ngram\(\)](#), [step_pos_filter\(\)](#), [step_stem\(\)](#), [step_stopwords\(\)](#), [step_tokenmerge\(\)](#)

Examples

```
library(recipes)
library(modeldata)
data(tate_text)

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(medium) %>%
  step_tokenfilter(medium)

tate_obj <- tate_rec %>%
  prep()

bake(tate_obj, new_data = NULL, medium) %>%
  slice(1:2)

bake(tate_obj, new_data = NULL) %>%
  slice(2) %>%
  pull(medium)
```

```
tidy(tate_rec, number = 2)
tidy(tate_obj, number = 2)
```

step_tokenize	<i>Tokenization of Character Variables</i>
---------------	--

Description

`step_tokenize()` creates a *specification* of a recipe step that will convert a character predictor into a `token` variable.

Usage

```
step_tokenize(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  training_options = list(),
  options = list(),
  token = "words",
  engine = "tokenizers",
  custom_token = NULL,
  skip = FALSE,
  id = rand_id("tokenize")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the <code>terms</code> argument. This is NULL until the step is trained by recipes::prep.recipe() .
training_options	A list of options passed to the tokenizer when it is being trained. Only applicable for <code>engine == "tokenizers.bpe"</code> .
options	A list of options passed to the tokenizer.
token	Unit for tokenizing. See details for options. Defaults to "words".
engine	Package that will be used for tokenization. See details for options. Defaults to "tokenizers".

custom_token	User supplied tokenizer. Use of this argument will overwrite the token and engine arguments. Must take a character vector as input and output a list of character vectors.
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

Tokenization is the act of splitting a character string into smaller parts to be further analyzed. This step uses the `tokenizers` package which includes heuristics on how to split the text into paragraphs tokens, word tokens, among others. `textrecipes` keeps the tokens as a `token` variable and other steps will do their tasks on those `token` variable before transforming them back to numeric variables.

Working with `textrecipes` will almost always start by calling `step_tokenize` followed by modifying and filtering steps. This is not always the case as you sometimes want to do apply pre-tokenization steps, this can be done with `recipes::step_mutate()`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

Engines

The choice of engine determines the possible choices of `token`.

The following is some small example data used in the following examples

```
text_tibble <- tibble(
  text = c("This is words", "They are nice!")
)
```

tokenizers:

The `tokenizers` package is the default engine and it comes with the following unit of `token`. All of these options correspond to a function in the `tokenizers` package.

- "words" (default)
- "characters"
- "character_shingles"
- "ngrams"
- "skip_ngrams"
- "sentences"
- "lines"
- "paragraphs"
- "regex"
- "tweets"

- "ptb" (Penn Treebank)
- "skip_ngrams"
- "word_stems"

The default tokenizer is "word" which splits the text into a series of words. By using `step_tokenize()` without setting any arguments you get word tokens

```
recipe(~ text, data = text_tibble) %>%
  step_tokenize(text) %>%
  show_tokens(text)
```

```
## [[1]]
## [1] "this" "is" "words"
##
## [[2]]
## [1] "they" "are" "nice"
```

This tokenizer has arguments that change how the tokenization occurs and can be accessed using the `options` argument by passing a named list. Here we are telling `tokenizers::tokenize_words` that we don't want to turn the words to lowercase

```
recipe(~ text, data = text_tibble) %>%
  step_tokenize(text,
                options = list(lowercase = FALSE)) %>%
  show_tokens(text)
```

```
## [[1]]
## [1] "This" "is" "words"
##
## [[2]]
## [1] "They" "are" "nice"
```

We can also stop removing punctuation.

```
recipe(~ text, data = text_tibble) %>%
  step_tokenize(text,
                options = list(strip_punct = FALSE,
                               lowercase = FALSE)) %>%
  show_tokens(text)
```

```
## [[1]]
## [1] "This" "is" "words"
##
## [[2]]
## [1] "They" "are" "nice" "!"
```

The tokenizer can be changed by setting a different token. Here we change it to return character tokens.

```
recipe(~ text, data = text_tibble) %>%
  step_tokenize(text, token = "characters") %>%
  show_tokens(text)
```



```
## [[1]]
## [1] "t" "h" "i" "s" "i" "s" "w" "o" "r" "d" "s"
##
## [[2]]
## [1] "t" "h" "e" "y" "a" "r" "e" "n" "i" "c" "e"
```

It is worth noting that not all these token methods are appropriate but are included for completeness.

spacyr:

- "words"

tokenizers.bpe:

The tokenizers.bpe engine performs Byte Pair Encoding Text Tokenization.

- "words"

This tokenizer is trained on the training set and will thus need to be passed training arguments. These are passed to the `training_options` argument and the most important one is `vocab_size`. This determines the number of unique tokens the tokenizer will produce. It is generally set to a much higher value, typically in the thousands, but is set to 22 here for demonstration purposes.

```
recipe(~ text, data = text_tibble) %>%
  step_tokenize(
    text,
    engine = "tokenizers.bpe",
    training_options = list(vocab_size = 22)
  ) %>%
  show_tokens(text)

## [[1]]
## [1] "_Th" "is" "_" "is" "_" "w" "o" "r" "d" "s"
##
## [[2]]
## [1] "_Th" "e" "y" "_" "a" "r" "e" "_" "n" "i" "c" "e"
## [13] "!"
```

udpipe:

- "words"

custom_token:

Sometimes you need to perform tokenization that is not covered by the supported engines. In that case you can use the `custom_token` argument to pass a function in that performs the tokenization you want.

Below is an example of a very simple space tokenization. This is a very fast way of tokenizing.

```
space_tokenizer <- function(x) {
  strsplit(x, " ")
}

recipe(~ text, data = text_tibble) %>%
```

```

step_tokenize(
  text,
  custom_token = space_tokenizer
) %>%
show_tokens(text)

## [[1]]
## [1] "This" "is" "words"
##
## [[2]]
## [1] "They" "are" "nice!"

```

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `value` (unit of tokenization).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_untokenize\(\)](#) to untokenize.

Other Steps for Tokenization: [step_tokenize_bpe\(\)](#), [step_tokenize_sentencepiece\(\)](#), [step_tokenize_wordpiece\(\)](#)

Examples

```

library(recipes)
library(modeldata)
data(tate_text)

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(medium)

tate_obj <- tate_rec %>%
  prep()

bake(tate_obj, new_data = NULL, medium) %>%
  slice(1:2)

bake(tate_obj, new_data = NULL) %>%
  slice(2) %>%
  pull(medium)

tidy(tate_rec, number = 1)
tidy(tate_obj, number = 1)

tate_obj_chars <- recipe(~., data = tate_text) %>%
  step_tokenize(medium, token = "characters") %>%
  prep()

```

```
bake(tate_obj, new_data = NULL) %>%
  slice(2) %>%
  pull(medium)
```

step_tokenize_bpe *BPE Tokenization of Character Variables*

Description

`step_tokenize_bpe()` creates a *specification* of a recipe step that will convert a character predictor into a `token` variable using Byte Pair Encoding.

Usage

```
step_tokenize_bpe(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  vocabulary_size = 1000,
  options = list(),
  res = NULL,
  skip = FALSE,
  id = rand_id("tokenize_bpe")
)
```

Arguments

<code>recipe</code>	A recipe object. The step will be added to the sequence of operations for this recipe.
<code>...</code>	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
<code>role</code>	Not used by this step since no new variables are created.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character string of variable names that will be populated (eventually) by the <code>terms</code> argument. This is <code>NULL</code> until the step is trained by recipes::prep.recipe() .
<code>vocabulary_size</code>	Integer, indicating the number of tokens in the final vocabulary. Defaults to 1000. Highly encouraged to be tuned.
<code>options</code>	A list of options passed to the tokenizer.
<code>res</code>	The fitted tokenizers.bpe::bpe() model tokenizer will be stored here once this preprocessing step has been trained by prep.recipe() .

skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_untokenize\(\)](#) to untokenize.

Other Steps for Tokenization: [step_tokenize_sentencepiece\(\)](#), [step_tokenize_wordpiece\(\)](#), [step_tokenize\(\)](#)

Examples

```
if (requireNamespace("tokenizers.bpe", quietly = TRUE)) {
  library(recipes)
  library(modeldata)
  data(tate_text)

  tate_rec <- recipe(~., data = tate_text) %>%
    step_tokenize_bpe(medium)

  tate_obj <- tate_rec %>%
    prep()

  bake(tate_obj, new_data = NULL, medium) %>%
    slice(1:2)

  bake(tate_obj, new_data = NULL) %>%
    slice(2) %>%
    pull(medium)

  tidy(tate_rec, number = 1)
  tidy(tate_obj, number = 1)
}
```

 step_tokenize_sentencepiece

Sentencepiece Tokenization of Character Variables

Description

`step_tokenize_sentencepiece()` creates a *specification* of a recipe step that will convert a character predictor into a **token** variable using SentencePiece tokenization.

Usage

```
step_tokenize_sentencepiece(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  vocabulary_size = 1000,
  options = list(),
  res = NULL,
  skip = FALSE,
  id = rand_id("tokenize_sentencepiece")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
vocabulary_size	Integer, indicating the number of tokens in the final vocabulary. Defaults to 1000. Highly encouraged to be tuned.
options	A list of options passed to the tokenizer.
res	The fitted sentencepiece::sentencepiece() model tokenizer will be stored here once this preprocessing step has been trained by prep.recipe() .
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

If you are running into errors, you can investigate the progress of the compiled code by setting `options = list(verbose = TRUE)`. This can reveal if sentencepiece ran correctly or not.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_untokenize\(\)](#) to untokenize.

Other Steps for Tokenization: [step_tokenize_bpe\(\)](#), [step_tokenize_wordpiece\(\)](#), [step_tokenize\(\)](#)

Examples

```
if (requireNamespace("sentencepiece", quietly = TRUE)) {
  library(recipes)
  library(modeldata)
  data(tate_text)

  tate_rec <- recipe(~., data = tate_text) %>%
    step_tokenize_sentencepiece(medium)

  tate_obj <- tate_rec %>%
    prep()

  bake(tate_obj, new_data = NULL, medium) %>%
    slice(1:2)

  bake(tate_obj, new_data = NULL) %>%
    slice(2) %>%
    pull(medium)

  tidy(tate_rec, number = 1)
  tidy(tate_obj, number = 1)
}
```

 step_tokenize_wordpiece

Wordpiece Tokenization of Character Variables

Description

`step_tokenize_wordpiece()` creates a *specification* of a recipe step that will convert a character predictor into a `token` variable using WordPiece tokenization.

Usage

```
step_tokenize_wordpiece(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  vocab = wordpiece::wordpiece_vocab(),
  unk_token = "[UNK]",
  max_chars = 100,
  skip = FALSE,
  id = rand_id("tokenize_wordpiece")
)
```

Arguments

<code>recipe</code>	A recipe object. The step will be added to the sequence of operations for this recipe.
<code>...</code>	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
<code>role</code>	Not used by this step since no new variables are created.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character string of variable names that will be populated (eventually) by the <code>terms</code> argument. This is <code>NULL</code> until the step is trained by recipes::prep.recipe() .
<code>vocab</code>	Character or Character vector of vocabulary tokens. Defaults to <code>wordpiece_vocab()</code> .
<code>unk_token</code>	Token to represent unknown words. Defaults to <code>"[UNK]"</code> .
<code>max_chars</code>	Integer, Maximum length of word recognized. Defaults to 100.
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
<code>id</code>	A character string that is unique to this step to identify it.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_untokenize\(\)](#) to untokenize.

Other Steps for Tokenization: [step_tokenize_bpe\(\)](#), [step_tokenize_sentencepiece\(\)](#), [step_tokenize\(\)](#)

Examples

```
if (requireNamespace("wordpiece", quietly = TRUE)) {
  library(recipes)
  library(modeldata)
  data(tate_text)

  tate_rec <- recipe(~., data = tate_text) %>%
    step_tokenize_wordpiece(medium)

  tate_obj <- tate_rec %>%
    prep()

  bake(tate_obj, new_data = NULL, medium) %>%
    slice(1:2)

  bake(tate_obj, new_data = NULL) %>%
    slice(2) %>%
    pull(medium)

  tidy(tate_rec, number = 1)
  tidy(tate_obj, number = 1)
}
```

step_tokenmerge

Combine Multiple Token Variables Into One

Description

`step_tokenmerge` creates a *specification* of a recipe step that will take multiple `token` variables and combine them into one `token` variable.

Usage

```
step_tokenmerge(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  prefix = "tokenmerge",
  skip = FALSE,
  id = rand_id("tokenmerge")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
prefix	A prefix for generated column names, default to "tokenmerge".
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe() ? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = FALSE.
id	A character string that is unique to this step to identify it.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you [tidy\(\)](#) this step, a tibble with columns terms (the selectors or variables selected).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_tokenize\(\)](#) to turn characters into [tokens](#)

Other Steps for Token Modification: [step_lemma\(\)](#), [step_ngram\(\)](#), [step_pos_filter\(\)](#), [step_stem\(\)](#), [step_stopwords\(\)](#), [step_tokenfilter\(\)](#)

Examples

```

library(recipes)
library(modeldata)
data(tate_text)

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(medium, artist) %>%
  step_tokenmerge(medium, artist)

tate_obj <- tate_rec %>%
  prep()

bake(tate_obj, new_data = NULL)

tidy(tate_rec, number = 2)
tidy(tate_obj, number = 2)

```

step_untokenize	<i>Untokenization of Token Variables</i>
-----------------	--

Description

step_untokenize creates a *specification* of a recipe step that will convert a [token](#) variable into a character predictor.

Usage

```

step_untokenize(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  sep = " ",
  skip = FALSE,
  id = rand_id("untokenize")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.

columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> .
sep	a character to determine how the tokens should be separated when pasted together. Defaults to " ".
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

This steps will turn a `token` vector back into a character vector. This step is calling `paste` internally to put the tokens back together to a character.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected) and `value` (separator used for collapsing).

Case weights

The underlying operation does not allow for case weights.

See Also

`step_tokenize()` to turn characters into `tokens`

Examples

```
library(recipes)
library(modeldata)
data(tate_text)

tate_rec <- recipe(~., data = tate_text) %>%
  step_tokenize(terms) %>%
  step_untokenize(terms)

tate_obj <- tate_rec %>%
  prep()

bake(tate_obj, new_data = NULL, terms) %>%
  slice(1:2)

bake(tate_obj, new_data = NULL) %>%
  slice(2) %>%
```

```

pull(medium)

tidy(tate_rec, number = 2)
tidy(tate_obj, number = 2)

```

step_word_embeddings *Pretrained Word Embeddings of Tokens*

Description

step_word_embeddings creates a *specification* of a recipe step that will convert a [token](#) variable into word-embedding dimensions by aggregating the vectors of each token from a pre-trained embedding.

Usage

```

step_word_embeddings(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  embeddings,
  aggregation = c("sum", "mean", "min", "max"),
  aggregation_default = 0,
  prefix = "wordembed",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("word_embeddings")
)

```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables are affected by the step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variable names that will be populated (eventually) by the terms argument. This is NULL until the step is trained by recipes::prep.recipe() .
embeddings	A tibble of pre-trained word embeddings, such as those returned by the embedding_glove function from the textdata package. The first column should contain tokens, and additional columns should contain embeddings vectors.

aggregation	A character giving the name of the aggregation function to use. Must be one of "sum", "mean", "min", and "max". Defaults to "sum".
aggregation_default	A numeric denoting the default value for case with no words are matched in embedding. Defaults to 0.
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake.recipe()</code> ? While all operations are baked when <code>recipes::prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = FALSE</code> .
id	A character string that is unique to this step to identify it.

Details

Word embeddings map words (or other tokens) into a high-dimensional feature space. This function maps pre-trained word embeddings onto the tokens in your data.

The argument `embeddings` provides the pre-trained vectors. Each dimension present in this tibble becomes a new feature column, with each column aggregated across each row of your text using the function supplied in the `aggregation` argument.

The new components will have names that begin with `prefix`, then the name of the aggregation function, then the name of the variable from the embeddings tibble (usually something like "d7"). For example, using the default "wordembedding" prefix, and the GloVe embeddings from the `textdata` package (where the column names are d1, d2, etc), new columns would be `wordembedding_d1`, `wordembedding_d1`, etc.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

Tidying

When you `tidy()` this step, a tibble with columns `terms` (the selectors or variables selected), `embedding_rows` (number of rows in embedding), and `aggregation` (the aggregation method).

Case weights

The underlying operation does not allow for case weights.

See Also

[step_tokenize\(\)](#) to turn characters into `tokens`

Other Steps for Numeric Variables From Tokens: [step_lda\(\)](#), [step_texthash\(\)](#), [step_tfidf\(\)](#), [step_tf\(\)](#)

Examples

```
library(recipes)

embeddings <- tibble(
  tokens = c("the", "cat", "ran"),
  d1 = c(1, 0, 0),
  d2 = c(0, 1, 0),
  d3 = c(0, 0, 1)
)

sample_data <- tibble(
  text = c(
    "The.",
    "The cat.",
    "The cat ran."
  ),
  text_label = c("fragment", "fragment", "sentence")
)

rec <- recipe(text_label ~ ., data = sample_data) %>%
  step_tokenize(text) %>%
  step_word_embeddings(text, embeddings = embeddings)

obj <- rec %>%
  prep()

bake(obj, sample_data)

tidy(rec, number = 2)
tidy(obj, number = 2)
```

tidy.step_clean_levels

Tidy the Result of a Recipe

Description

tidy will return a data frame that contains information regarding a recipe or operation within the recipe (when a tidy method for the operation exists). See [recipes::tidy.recipe](#) for more information.

Usage

```
## S3 method for class 'step_clean_levels'
tidy(x, ...)

## S3 method for class 'step_clean_names'
tidy(x, ...)

## S3 method for class 'step_word_embeddings'
```

```
tidy(x, ...)  
  
## S3 method for class 'step_texthash'  
tidy(x, ...)  
  
## S3 method for class 'step_dummy_hash'  
tidy(x, ...)  
  
## S3 method for class 'step_lda'  
tidy(x, ...)  
  
## S3 method for class 'step_lemma'  
tidy(x, ...)  
  
## S3 method for class 'step_ngram'  
tidy(x, ...)  
  
## S3 method for class 'step_pos_filter'  
tidy(x, ...)  
  
## S3 method for class 'step_sequence_onehot'  
tidy(x, ...)  
  
## S3 method for class 'step_stem'  
tidy(x, ...)  
  
## S3 method for class 'step_stopwords'  
tidy(x, ...)  
  
## S3 method for class 'step_text_normalization'  
tidy(x, ...)  
  
## S3 method for class 'step_textfeature'  
tidy(x, ...)  
  
## S3 method for class 'step_tf'  
tidy(x, ...)  
  
## S3 method for class 'step_tfidf'  
tidy(x, ...)  
  
## S3 method for class 'step_tokenfilter'  
tidy(x, ...)  
  
## S3 method for class 'step_tokenize'  
tidy(x, ...)  
  
## S3 method for class 'step_tokenize_bpe'
```

```

tidy(x, ...)

## S3 method for class 'step_tokenize_sentencepiece'
tidy(x, ...)

## S3 method for class 'step_tokenize_wordpiece'
tidy(x, ...)

## S3 method for class 'step_tokenmerge'
tidy(x, ...)

## S3 method for class 'step_untokenize'
tidy(x, ...)

```

Arguments

x	A step_untokenize object.
...	Not currently used.

tokenlist	<i>Create Token Object</i>
-----------	----------------------------

Description

A [tokenlist](#) object is a thin wrapper around a list of character vectors, with a few attributes.

Usage

```
tokenlist(tokens = list(), lemma = NULL, pos = NULL)
```

Arguments

tokens	List of character vectors
lemma	List of character vectors, must be same size and shape as x.
pos	List of character vectors, must be same size and shape as x.

Value

a [tokenlist](#) object.

Examples

```

abc <- list(letters, LETTERS)
tokenlist(abc)

unclass(tokenlist(abc))

```



```
tibble(text = tokenlist(abc))

library(tokenizers)
library(modeldata)
data(tate_text)
tokens <- tokenize_words(as.character(tate_text$medium))

tokenlist(tokens)
```

Index

- * **Steps for Numeric Variables From Characters**
 - step_dummy_hash, 7
 - step_sequence_onehot, 18
 - step_textfeature, 24
- * **Steps for Numeric Variables From Tokens**
 - step_lda, 10
 - step_texthash, 26
 - step_tf, 30
 - step_tfidf, 33
 - step_word_embeddings, 52
- * **Steps for Text Cleaning**
 - step_clean_levels, 4
 - step_clean_names, 6
- * **Steps for Text Normalization**
 - step_text_normalization, 29
- * **Steps for Token Modification**
 - step_lemma, 12
 - step_ngram, 14
 - step_pos_filter, 16
 - step_stem, 20
 - step_stopwords, 22
 - step_tokenfilter, 35
 - step_tokenmerge, 48
- * **Steps for Tokenization**
 - step_tokenize, 38
 - step_tokenize_bpe, 43
 - step_tokenize_sentencepiece, 45
 - step_tokenize_wordpiece, 47
- * **Steps for Un-Tokenization**
 - step_untokenize, 50
- * **datasets**
 - emoji_samples, 3
- all_tokenized, 2
- all_tokenized_predictors
 - (all_tokenized), 2
- count_functions, 25
- dplyr::recode_factor(), 5
- emoji_samples, 3
- prep.recipe(), 31, 34, 36, 43, 45
- recipe, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 25, 27, 29, 31, 33, 36, 38, 43, 45, 47, 49, 50, 52
- recipes::bake.recipe(), 5, 6, 8, 11, 13, 15, 16, 19, 20, 23, 25, 27, 29, 31, 34, 36, 39, 44, 45, 47, 49, 51, 53
- recipes::has_role(), 3
- recipes::prep.recipe(), 4–6, 8, 10, 11, 13–16, 18–20, 23, 25, 27, 29, 31, 34, 36, 38, 39, 43–45, 47, 49, 51–53
- recipes::selections(), 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 25, 27, 29, 31, 33, 36, 38, 43, 45, 47, 49, 50, 52
- recipes::step_dummy(), 9
- recipes::step_factor2string(), 5, 7
- recipes::step_mutate(), 39
- recipes::step_novel(), 5, 7
- recipes::step_other(), 5, 7
- recipes::step_regex(), 5, 7
- recipes::step_string2factor(), 5, 7
- recipes::step_unknown(), 5, 7
- recipes::tidy.recipe, 54
- sentencepiece::sentencepiece(), 45
- show_tokens, 3
- step_clean_levels, 4, 7
- step_clean_levels(), 7
- step_clean_names, 5, 6
- step_clean_names(), 5
- step_dummy_hash, 7, 19, 26
- step_lda, 10, 28, 32, 35, 53
- step_lemma, 12, 15, 17, 21, 23, 37, 49
- step_ngram, 13, 14, 17, 21, 23, 37, 49
- step_pos_filter, 13, 15, 16, 21, 23, 37, 49

- step_sequence_onehot, [9](#), [18](#), [26](#)
- step_stem, [13](#), [15](#), [17](#), [20](#), [23](#), [37](#), [49](#)
- step_stopwords, [13](#), [15](#), [17](#), [21](#), [22](#), [37](#), [49](#)
- step_text_normalization, [29](#)
- step_text_normalization(), [28](#)
- step_textfeature, [9](#), [19](#), [24](#)
- step_texthash, [11](#), [26](#), [32](#), [35](#), [53](#)
- step_texthash(), [30](#)
- step_tf, [11](#), [28](#), [30](#), [32](#), [35](#), [37](#), [53](#)
- step_tfidf, [11](#), [28](#), [32](#), [33](#), [34](#), [37](#), [53](#)
- step_tokenfilter, [13](#), [15](#), [17](#), [21](#), [23](#), [32](#), [34](#),
[35](#), [49](#)
- step_tokenize, [38](#), [44](#), [46](#), [48](#)
- step_tokenize(), [13](#), [15](#), [17](#), [21](#), [23](#), [28](#), [32](#),
[35](#), [37](#), [38](#), [49](#), [51](#), [53](#)
- step_tokenize_bpe, [42](#), [43](#), [46](#), [48](#)
- step_tokenize_bpe(), [43](#)
- step_tokenize_sentencepiece, [42](#), [44](#), [45](#),
[48](#)
- step_tokenize_sentencepiece(), [45](#)
- step_tokenize_wordpiece, [42](#), [44](#), [46](#), [47](#)
- step_tokenize_wordpiece(), [47](#)
- step_tokenmerge, [13](#), [15](#), [17](#), [21](#), [23](#), [37](#), [48](#)
- step_untokenize, [50](#)
- step_untokenize(), [42](#), [44](#), [46](#), [48](#)
- step_word_embeddings, [11](#), [28](#), [32](#), [35](#), [52](#)
- stringi::stri_trans_nfc(), [29](#)

- tidy(), [5](#), [6](#), [9](#), [11](#), [13](#), [15](#), [17](#), [19](#), [21](#), [23](#), [26](#),
[28](#), [30](#), [32](#), [35](#), [37](#), [42](#), [44](#), [46](#), [48](#), [49](#),
[51](#), [53](#)
- tidy.recipe(tidy.step_clean_levels), [54](#)
- tidy.step_clean_levels, [54](#)
- tidy.step_clean_names
(tidy.step_clean_levels), [54](#)
- tidy.step_dummy_hash
(tidy.step_clean_levels), [54](#)
- tidy.step_lda(tidy.step_clean_levels),
[54](#)
- tidy.step_lemma
(tidy.step_clean_levels), [54](#)
- tidy.step_ngram
(tidy.step_clean_levels), [54](#)
- tidy.step_pos_filter
(tidy.step_clean_levels), [54](#)
- tidy.step_sequence_onehot
(tidy.step_clean_levels), [54](#)
- tidy.step_stem
(tidy.step_clean_levels), [54](#)

- tidy.step_stopwords
(tidy.step_clean_levels), [54](#)
- tidy.step_text_normalization
(tidy.step_clean_levels), [54](#)
- tidy.step_textfeature
(tidy.step_clean_levels), [54](#)
- tidy.step_texthash
(tidy.step_clean_levels), [54](#)
- tidy.step_tf(tidy.step_clean_levels),
[54](#)
- tidy.step_tfidf
(tidy.step_clean_levels), [54](#)
- tidy.step_tokenfilter
(tidy.step_clean_levels), [54](#)
- tidy.step_tokenize
(tidy.step_clean_levels), [54](#)
- tidy.step_tokenize_bpe
(tidy.step_clean_levels), [54](#)
- tidy.step_tokenize_sentencepiece
(tidy.step_clean_levels), [54](#)
- tidy.step_tokenize_wordpiece
(tidy.step_clean_levels), [54](#)
- tidy.step_tokenmerge
(tidy.step_clean_levels), [54](#)
- tidy.step_untokenize
(tidy.step_clean_levels), [54](#)
- tidy.step_word_embeddings
(tidy.step_clean_levels), [54](#)
- token, [2](#), [12–14](#), [16](#), [20](#), [22](#), [27](#), [30](#), [33](#), [35](#), [38](#),
[39](#), [43](#), [45](#), [47](#), [48](#), [50–52](#)
- tokenizers.bpe::bpe(), [43](#)
- tokenizers::tokenize_words, [40](#)
- tokenlist, [56](#), [56](#)
- tokens, [13](#), [15](#), [17](#), [21](#), [23](#), [28](#), [32](#), [35](#), [37](#), [49](#),
[51](#), [53](#)