Title  Extra 'Recipes' for Text Processing
Version  0.2.2
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
BugReports  https://github.com/tidymodels/textrecipes/issues
Depends  R (>= 2.10), recipes (>= 0.1.4)
Imports  generics, rlang, tokenizers, dplyr, tibble, tidyr, purrr, SnowballC, stopwords, magrittr, Matrix, stringr, vctrs, Rcpp
Suggests  covr, testthat (>= 2.1.0), knitr, text2vec, rmarkdown, textfeatures (>= 0.3.3), modeldata, spacyr
VignetteBuilder  knitr
Encoding  UTF-8
LazyData  true
RoxygenNote  7.1.0.9000
SystemRequirements  GNU make, C++11
LinkingTo  Rcpp
NeedsCompilation  yes
Author  Emil Hvitfeldt [aut, cre] (<https://orcid.org/0000-0002-0679-1945>)
Maintainer  Emil Hvitfeldt <emilhhvitfeldt@gmail.com>
Repository  CRAN
Date/Publication  2020-05-10 07:00:02 UTC
R topics documented:

rcpp_ngram .......................... 2
step_lda .................................. 3
step_lemma .................................. 5
step_ngram .................................. 7
step_pos_filter .............................. 8
step_sequence_onehot ....................... 10
step_stem .................................. 12
step_stopwords ............................. 14
step_textfeature ............................ 16
step_texthash ................................ 18
step_tf ....................................... 20
step_tfidf ................................... 23
step_tokenfilter ............................ 25
step_tokenize ................................ 27
step_tokenmerge ............................. 30
step_untokenize ............................. 31
step_word_embeddings ....................... 33
tokenlist .................................... 35

Index .......................... 37

rcpp_ngram

description

ngram generator

usage

rcpp_ngram(x, n, delim)

arguments

x list of character vectors
n number of grams
delim delimiter
**step_lda**

Calculates LDA dimension estimates

**Description**

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

**Usage**

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

## S3 method for class 'step_lda'

tidy(x, ...)

**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 variables. For `step_lda`, this indicates the variables to be encoded into a `tokenlist`. See `recipes::selections()` for more details. For the tidy method, these are not currently used.

- **role**
  
  For model terms created by this step, what analysis role should they be assigned to? 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 recipe has been baked.

- **columns**
  
  A list of tibble results that define the encoding. 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, it will train its model based on the training data. Look at the examples for how to fit a WarpLDA model.

- **num_topics**
  
  An integer desired number of latent topics.

- **prefix**
  
  A prefix for generated column names, default to "lda".
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 = TRUE` as it may affect the computations for subsequent operations.

id

A character string that is unique to this step to identify it

x

A `step_lda` object.

Value

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

Source

https://arxiv.org/abs/1301.3781

See Also

Other character to numeric steps: `step_sequence_onehot()`, `step_textfeature()`

Examples

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

library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_lda(essay0)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj) %>%
  slice(1:2)
tidy(okc_rec, number = 1)
tidy(okc_obj, number = 1)

# Changing the number of topics.
recipe(~ ., data = okc_text) %>%
  step_lda(essay0, essay1, num_topics = 20) %>%
  prep() %>%
  juice() %>%
  slice(1:2)

# Supplying a pre-trained LDA model trained using text2vec
library(text2vec)
tokens <- word_tokenizer(tolower(okc_text$essay5))
it <- itoken(tokens, ids = seq_along(okc_text$essay5))
v <- create_vocabulary(it)
```
```r
dtm <- create_dtm(it, vocab_vectorizer(v))
lda_model <- LDA$new(n_topics = 15)

recipe(~ ., data = okc_text) %>%
  step_lda(essay0, essay1, lda_models = lda_model) %>%
  prep() %>%
  juice() %>%
  slice(1:2)
```

---

**step_lemma**

Lemmatization of `tokenlist` variables

### Description

`step_lemma` creates a specification of a recipe step that will extract the lemmatization of a tokenlist.

### Usage

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

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

### 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 variables. For `step_lemma`, this indicates the variables to be encoded into a `tokenlist`. See `recipes::selections()` for more details. For the tidy method, these are not currently used.
- `role` Not used by this step since no new variables are created.
- `trained` A logical to indicate if the recipe has been baked.
- `columns` A list of tibble results that define the encoding. This is `NULL` until the step is trained 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 = TRUE` as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

x A `step_lemma` object.

Details
This stem doesn't perform lemmatization by itself, but rather lets you extract the lemma attribute of the tokenlist. To be able to use `step_lemma` you need to use a tokenization method that includes lemmatization. Currently using the "spacy" 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).

See Also

- `step_tokenize()` to turn character into tokenlist.
- Other tokenlist to tokenlist steps: `step_ngram()`, `step_pos_filter()`, `step_stem()`, `step_stopwords()`, `step_tokenfilter()`, `step_tokenmerge()`

Examples
```r
## Not run:
library(recipes)
short_data <- data.frame(text = c("This is a short tale,",
    "With many cats and ladies."))

okc_rec <- recipe(~ text, data = short_data) %>%
  step_tokenize(text, engine = "spacy") %>%
  step_lemma(text) %>%
  step_tf(text)

okc_obj <- prep(okc_rec)
juice(okc_obj)

## End(Not run)
```
step_ngram  

Generate ngrams from tokenlist

Description

step_ngram creates a specification of a recipe step that will convert a tokenlist into a list of ngram of tokens.

Usage

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

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

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 variables. For step_ngram, this indicates the variables to be encoded into a tokenlist. See recipes::selections() for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the recipe has been baked.
- **columns**: A list of tibble results that define the encoding. 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.
- **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 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 = TRUE as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A step_ngram object.
Value

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

See Also

step_tokenize() to turn character into tokenlist.

Other tokenlist to tokenlist steps: step_lemma(), step_pos_filter(), step_stem(), step_stopwords(), step_tokenfilter(), step_tokenmerge()

Examples

library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0) %>%
  step_ngram(essay0)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj, essay0) %>%
  slice(1:2)

juice(okc_obj) %>%
  slice(2) %>%
  pull(essay0)

tidy(okc_rec, number = 2)
tidy(okc_obj, number = 2)

---

**step_pos_filter**

Part of speech filtering of tokenlist variables

Description

step_pos_filter creates a specification of a recipe step that will filter a tokenlist based on part of speech tags.

Usage

step_pos_filter(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  columns = NULL,
### Details


### Value

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

### See Also

- `step_tokenize()` to turn character into tokenlist.

Other tokenlist to tokenlist steps: `step_lemma()`, `step_ngram()`, `step_stem()`, `step_stopwords()`, `step_tokenfilter()`, `step_tokenmerge()`
Examples

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

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

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

okc_obj <- prep(okc_rec)

juice(okc_obj)

## End(Not run)
```

---

**step_sequence_onehot**  
*Generate the basic set of text features*

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

```r
step_sequence_onehot(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  string_length = 100,
  integer_key = letters,
  prefix = "seq1hot",
  skip = FALSE,
  id = rand_id("sequence_onehot")
)
```

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

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 variables. For `step_sequence_onehot`, this indicates the variables to be encoded into a tokenlist. See `recipes::selections()` for more details. For the tidy method, these are not currently used.

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.

A logical to indicate if the recipe has been baked.

A list of tibble results that define the encoding. This is NULL until the step is trained by `recipes::prep.recipe()`.

A numeric, number of characters to keep before discarding. Defaults to 100.

A character vector, characters to be mapped to integers. Characters not in the integer_key will be encoded as 0. Defaults to letters.

A prefix for generated column names, default to "seq1hot".

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 = TRUE` as it may affect the computations for subsequent operations.

A character string that is unique to this step to identify it

A step_sequence_onehot object.

The string will be capped by the string_length argument, strings shorter then string_length will be padded with empty characters. The encoding will assign a integer to each character in the integer_key, and will encode accordingly. Characters not in the integer_key will be encoded as 0.

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


Other character to numeric steps: `step_lda()`, `step_textfeature()`

library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
step_stem

```
step_sequence_onehot(essay0)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj)

tidy(okc_rec, number = 1)
tidy(okc_obj, number = 1)
```

---

**step_stem**

**Stemming of tokenlist variables**

**Description**

step_stem creates a *specification* of a recipe step that will convert a *tokenlist* to have its tokens stemmed.

**Usage**

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

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

**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 variables. For step_stem, this indicates the variables to be encoded into a *tokenlist*. See `recipes::selections()` for more details. For the tidy method, these are not currently used.
- **role**
  Not used by this step since no new variables are created.
- **trained**
  A logical to indicate if the recipe has been baked.
- **columns**
  A list of tibble results that define the encoding. This is NULL until the step is trained by `recipes::prep.recipe()`.
Step Stem

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 = TRUE as it may affect the computations for subsequent operations.
id A character string that is unique to this step to identify it.
x A step_stem object.

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).

See Also

step_tokenize() to turn character into tokenlist.

Other tokenlist to tokenlist steps: step_lemma(), step_ngram(), step_pos_filter(), step_stopwords(), step_tokenfilter(), step_tokenmerge()

Examples

library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0) %>%
  step_stem(essay0)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj, essay0) %>%
  slice(1:2)

juice(okc_obj) %>%
  slice(2) %>%
  pull(essay0)

tidy(okc_rec, number = 2)
tidy(okc_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)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0) %>%
  step_stem(essay0, custom_stemmer = remove_s)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj, essay0) %>%
  slice(1:2)

juice(okc_obj) %>%
  slice(2) %>%
  pull(essay0)

---

**step_stopwords**  
*Filtering of stopwords from a tokenlist variable*

**Description**

*step_stopwords* creates a specification of a recipe step that will filter a tokenlist for stopwords (keep or remove).

**Usage**

```r
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")
)
```

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

```r
step_stopwords
```
step_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 variables. For `step_stopwords`, this indicates the variables to be encoded into a tokenlist. See `recipes::selections()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the recipe has been baked.
- **columns**: A list of tibble results that define the encoding. This is NULL until the step is trained by `recipes::prep.recipe()`.
- **language**: A character to indicate the language of stopwords by ISO 639-1 coding scheme.
- **keep**: A logical. Specifies whether to keep the stopwords or discard them.
- **stopword_source**: A character to indicate the stopwords source as listed in `stopwords::stopwords_getsources()`.
- **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 `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 = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_stopwords` object.

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).

See Also

- `step_tokenize()` to turn character into tokenlist.
- Other tokenlist to tokenlist steps: `step_lemma()`, `step_ngram()`, `step_pos_filter()`, `step_stem()`, `step_tokenfilter()`, `step_tokenmerge()`
Examples

```r
library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0) %>%
  step_stopwords(essay0)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj, essay0) %>%
  slice(1:2)

juice(okc_obj) %>%
  slice(2) %>%
  pull(essay0)

tidy(okc_rec, number = 2)
tidy(okc_obj, number = 2)
# With a custom stopwords list

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0) %>%
  step_stopwords(essay0, custom_stopword_source = c("twice", "upon"))
okc_obj <- okc_rec %>%
  prep(trainmg = okc_text)

juice(okc_obj) %>%
  slice(2) %>%
  pull(essay0)
```

**step_textfeature**

Generate the basic 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**

```r
step_textfeature(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  columns = NULL,
```
extract_functions = textfeatures::count_functions,
prefix = "textfeature",
skip = FALSE,
id = rand_id("textfeature")
}

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

**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 variables. For `step_textfeature`, this indicates the variables to be encoded into a tokenlist. See `recipes::selections()` for more details. For the `tidy` method, these are not currently used.
- `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 recipe has been baked.
- `columns` A list of tibble results that define the encoding. 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".
- `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 = TRUE` as it may affect the computations for subsequent operations.
- `id` A character string that is unique to this step to identify it
- `x` A `step_textfeature` object.

**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).
See Also

Other character to numeric steps: `step_lda()`, `step_sequence_onehot()`

Examples

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

  okc_rec <- recipe(~ ., data = okc_text) %>%
    step_textfeature(essay0)

  okc_obj <- okc_rec %>%
    prep()

  juice(okc_obj) %>%
    slice(1:2)

  juice(okc_obj) %>%
    pull(textfeature_essay0_n_words)

  tidy(okc_rec, number = 1)
  tidy(okc_obj, number = 1)

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

  recipe(~ ., data = okc_text) %>%
    step_textfeature(essay0,
      extract_functions = list(nchar10 = nchar_round_10)) %>%
    prep() %>%
    juice()
}
```

---

**step_texthash**

Term frequency of tokens

**Description**

`step_texthash` creates a specification of a recipe step that will convert a tokenlist into multiple variables using the hashing trick.

**Usage**

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

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

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 variables. For step_texthash, this indicates the variables to be encoded into a tokenlist. See recipes::selections() for more details. For the tidy method, these are not currently used.

role For model terms created by this step, what analysis role should they be assigned to? 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 recipe has been baked.

columns A list of tibble results that define the encoding. 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.

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 = TRUE as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

x A step_texthash object.

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.
step_tf

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, a higher value of `num_terms` will 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 `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).

References

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

See Also

`step_tokenize()` to turn character into tokenlist.

Other tokenlist to numeric steps: `step_tfidf()`, `step_tf()`, `step_word_embeddings()`

Examples

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

  okc_rec <- recipe(~ ., data = okc_text) %>%
    step_tokenize(essay0) %>%
    step_tokenfilter(essay0, max_tokens = 10) %>%
    step_texthash(essay0)

  okc_obj <- okc_rec %>%
    prep()

  bake(okc_obj, okc_text)

  tidy(okc_rec, number = 2)
  tidy(okc_obj, number = 2)
}
```

---

**step_tf**

*Term frequency of tokens*

Description

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

Usage

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

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

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 variables. For step_tf, this indicates the variables to be encoded into a tokenlist. See recipes::selections() for more details. For the tidy method, these are not currently used.

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 recipe has been baked.

columns A list of tibble results that define the encoding. 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

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 = TRUE as it may affect the computations for subsequent operations.

**id**
A character string that is unique to this step to identify it.

**x**
A step_tf object.

**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 new variables will be created alphabetically according to token.

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

**See Also**

step_tokenize() to turn character into tokenlist.

Other tokenlist to numeric steps: step_texthash(), step_tfidf(), step_word_embeddings()

**Examples**

```r
library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0) %>%
  step_tf(essay0)

okc_obj <- okc_rec %>%
  prep()

bake(okc_obj, okc_text)
```
**step_tfidf**

```r
tidy(okc_rec, number = 2)
tidy(okc_obj, number = 2)
```

---

**Description**

`step_tfidf` creates a *specification* of a recipe step that will convert a *tokenlist* into multiple variables containing the term frequency-inverse document frequency of tokens.

**Usage**

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

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

**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 variables. For `step_tfidf`, this indicates the variables to be encoded into a *tokenlist*. See `recipes::selections()` for more details. For the tidy method, these are not currently used.
- `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 recipe has been baked.
- `columns` A list of tibble results that define the encoding. This is NULL until the step is trained by `recipes::prep.recipe()`.
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 prep.recipe().
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(TF). Defaults to FALSE.
prefix: A character string that will be the prefix to the resulting new variables. See notes below.
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 = TRUE as it may affect the computations for subsequent operations.
id: A character string that is unique to this step to identify it.
x: A step_tfidf object.

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 new variables will be created alphabetically according to token.

Value

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

See Also

step_tokenize() to turn character into tokenlist.
Other tokenlist to numeric steps: step_texthash(), step_tf(), step_word_embeddings()
### Examples

```r
library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0) %>%
  step_tfidf(essay0)

okc_obj <- okc_rec %>%
  prep()

bake(okc_obj, okc_text)

tidy(okc_rec, number = 2)
tidy(okc_obj, number = 2)
```

---

**step_tokenfilter**  
*Filter the tokens based on term frequency*

**Description**

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

**Usage**

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

```r
## S3 method for class 'step_tokenfilter'
tidy(x, ...)
```
**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 variables. For `step_tokenfilter`, this indicates the variables to be encoded into a tokenlist. See `recipes::selections()` for more details. For the tidy method, these are not currently used.

- **role**
  Not used by this step since no new variables are created.

- **trained**
  A logical to indicate if the recipe has been baked.

- **columns**
  A list of tibble results that define the encoding. 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.

- **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 = TRUE` as it may affect the computations for subsequent operations.

- **id**
  A character string that is unique to this step to identify it.

- **x**
  A `step_tokenfilter` object.

**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 fews 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 datasets 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).
See Also

`step_tokenize()` to turn character into tokenlist.

Other tokenlist to tokenlist steps: `step_lemma()`, `step_ngram()`, `step_pos_filter()`, `step_stem()`, `step_stopwords()`, `step_tokenmerge()`

Examples

```r
library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0) %>%
  step_tokenfilter(essay0)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj, essay0) %>%
  slice(1:2)

juice(okc_obj) %>%
  slice(2) %>%
  pull(essay0)

tidy(okc_rec, number = 2)
tidy(okc_obj, number = 2)
```

---

### step_tokenize

**Tokenization of character variables**

**Description**

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

**Usage**

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

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

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 variables. For step_tokenize(), this indicates the variables to be encoded into a tokenlist. See recipes::selections() for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the recipe has been baked.
- **columns**: A list of tibble results that define the encoding. This is NULL until the step is trained by recipes::prep.recipe().
- **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 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 = TRUE as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it
- **x**: A step_tokenize object.

Details

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

The choice of engine determines the possible choices of token.

If engine = "tokenizers":

- "words" (default)
- "characters"
- "character_shingles"


- "ngrams"
- "skip_ngrams"
- "sentences"
- "lines"
- "paragraphs"
- "regex"
- "tweets"
- "ptb" (Penn Treebank)
- "skip_ngrams"
- "word_stems"

if engine = "spacyr"
- "words"

Working will 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).

See Also

`step_untokenize()` to untokenize.

Examples

```r
library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj, essay0) %>%
  slice(1:2)

juice(okc_obj) %>
  slice(2) %>
  pull(essay0)

tidy(okc_rec, number = 1)
tidy(okc_obj, number = 1)
```
okc_obj_chars <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0, token = "characters") %>%
  prep()

juice(okc_obj_chars) %>%
  slice(2) %>%
  pull(essay0)

---

**step_tokenmerge**

*Generate the basic set of text features*

**Description**

`step_tokenmerge` creates a *specification* of a recipe step that will take multiple tokenlists and combine them into one tokenlist.

**Usage**

```r
step_tokenmerge(
  recipe,
  ...,  # One or more selector functions to choose variables. For step_tokenmerge, this indicates the variables to be encoded into a tokenlist. See recipes::selections() for more details. For the tidy method, these are not currently used.
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  prefix = "tokenmerge",
  skip = FALSE,
  id = rand_id("tokenmerge")
)
```

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

**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 variables. For step_tokenmerge, this indicates the variables to be encoded into a tokenlist. See recipes::selections() for more details. For the tidy method, these are not currently used.
- `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 recipe has been baked.
- `columns` A list of tibble results that define the encoding. 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 \texttt{recipes::bake.recipe()}? While all operations are baked when \texttt{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 \texttt{skip = TRUE} as it may affect the computations for subsequent operations.

id  A character string that is unique to this step to identify it

x  A \texttt{step_tokenmerge} object.

Value

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

See Also

\texttt{step_tokenize()} to turn character into tokenlist.

Other tokenlist to tokenlist steps: \texttt{step_lemma()}, \texttt{step_ngram()}, \texttt{step_pos_filter()}, \texttt{step_stem()}, \texttt{step_stopwords()}, \texttt{step_tokenfilter()}

Examples

```
library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0, essay1) %>%
  step_tokenmerge(essay0, essay1)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj)

tidy(okc_rec, number = 1)
tidy(okc_obj, number = 1)
```

---

**step_untokenize**

Untokenization of tokenlist variables

Description

\texttt{step_untokenize} creates a \textit{specification} of a recipe step that will convert a \textit{tokenlist} into a character predictor.
### Usage

```r
code
# A recipe object. The step will be added to the sequence of operations for this recipe.
...

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

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

### 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 variables. For `step_untokenize`, this indicates the variables to be encoded into a tokenlist. See `recipes::selections()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the recipe has been baked.
- **columns**: A list of tibble results that define the encoding. This is NULL until the step is trained by `recipes::prep.recipe()`.
- **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 `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 = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_untokenize` object.

### Details

This step will turn a tokenlist 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).
See Also

step_tokenize() to turn character into tokenlist.

Examples

```r
library(recipes)
library(modeldata)
data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0) %>%
  step_untokenize(essay0)

okc_obj <- okc_rec %>%
  prep()

juice(okc_obj, essay0) %>%
  slice(1:2)

juice(okc_obj) %>%
  slice(2) %>%
  pull(essay0)

tidy(okc_rec, number = 2)
tidy(okc_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 tokenlist into word-embedding dimensions by aggregating the vectors of each token from a pre-trained embedding.

Usage

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

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

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 variables. For step_word_embeddings, this indicates the variables to be encoded into a tokenlist. See recipes::selections() for more details. For the tidy method, these are not currently used.

role For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.

test A logical to indicate if the recipe has been baked.

columns A list of tibble results that define the encoding. 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.

skip A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe()?

While all operations are baked when recipes::bake.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 = TRUE as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

x A step_word_embeddings object.

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 "word_embeddings" prefix, the "sum" aggregation, and the GloVe embeddings from the textdata package (where the column names are d1, d2, etc), new columns would be word_embeddings_sum_d1, word_embeddings_sum_d2, etc.

Value

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

See Also

step_tokenize() to turn character into tokenlist.

Other tokenlist to numeric steps: 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)
)

tokenlist

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)
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(okc_text)
tokens <- tokenize_words(okc_text$essay0)
tokenlist(tokens)
Index

count_functions, 17
prep.recipe(), 21, 24, 26
rcpp_ngram, 2
recipes::bake.recipe(), 4, 6, 7, 9, 11, 13, 15, 17, 19, 21, 24, 26, 28, 31, 32, 34
recipes::prep.recipe(), 3–7, 9, 11–13, 15, 17, 19, 21, 23, 24, 26, 28, 30–32, 34
recipes::selections(), 3, 5, 7, 9, 11, 12, 15, 17, 19, 21, 23, 26, 28, 30, 32, 34
recipes::step_mutate(), 29
step_lda, 3, 11, 18
step_lemma, 5, 8, 9, 13, 15, 27, 31
step_ngram, 6, 7, 9, 13, 15, 27, 31
step_pos_filter, 6, 8, 9, 13, 15, 27, 31
step_sequence_onehot, 4, 10, 18
step_stem, 6, 8, 9, 12, 15, 27, 31
step_stopwords, 6, 8, 9, 13, 14, 27, 31
step_textfeature, 4, 11, 16
step_texthash, 18, 22, 24, 35
step_tf, 20, 22, 24, 26, 35
step_tfidf, 20, 22, 23, 24, 26, 35
step_tokenfilter, 6, 8, 9, 13, 15, 22, 24, 25, 31
step_tokenize, 27
step_tokenize(), 6, 8, 9, 13, 15, 20, 22, 24, 27, 28, 31, 33, 35
step_tokenmerge, 6, 8, 9, 13, 15, 27, 30
step_untokenize, 31
step_untokenize(), 29
step_word_embeddings, 20, 22, 24, 33
tidy.step_lemma (step_lemma), 5
tidy.step_ngram (step_ngram), 7
tidy.step_pos_filter (step_pos_filter), 8
tidy.step_sequence_onehot (step_sequence_onehot), 10
tidy.step_stem (step_stem), 12
tidy.step_stopwords (step_stopwords), 14
tidy.step_textfeature (step_textfeature), 16
tidy.step_texthash (step_texthash), 18
tidy.step_tf (step_tf), 20
tidy.step_tfidf (step_tfidf), 23
tidy.step_tokenfilter (step_tokenfilter), 25
tidy.step_tokenize (step_tokenize), 27
tidy.step_tokenmerge (step_tokenmerge), 30
tidy.step_untokenize (step_untokenize), 31
tidy.step_word_embeddings (step_word_embeddings), 33
tokenlist, 3, 5, 7–9, 11, 12, 14, 15, 17–21, 23, 25–28, 30–34, 35, 36