

Package: textrecipes (via r-universe)

October 15, 2026

Title Extra 'Recipes' for Text Processing

Version 1.1.0.9000

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/>

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

Depends R (≥ 4.1), recipes ($\geq 1.4.0$)

Imports cli, dplyr, generics ($\geq 0.1.0$), glue, lifecycle, magrittr, Matrix, purrr, rlang ($\geq 1.1.0$), SnowballC, sparsevctrs ($\geq 0.3.0$), tibble, tokenizers, vctrs

Suggests covr, dials ($\geq 1.2.0$), hardhat, httr2 ($\geq 1.0.0$), janitor, knitr, modeldata, reticulate, rmarkdown, sentencepiece, spacyr, stopwords, stringi, testthat ($\geq 3.0.0$), text2vec, tokenizers.bpe, udpipe, withr, wordpiece

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate, reticulate

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-24

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

SystemRequirements ``GNU make``

Config/roxygen2/version 8.1.0

Config/pak/sysreqs make libicu-dev

Repository <https://tidymodels.r-universe.dev>

Date/Publication 2026-10-15 03:37:33 UTC

RemoteUrl <https://github.com/tidymodels/textrecipes>

RemoteRef HEAD

RemoteSha e45444833b3040df3c2236d7e76859bcef924418

Contents

all_tokenized	3
count_functions	3
embedding_custom	5
embedding_ollama	6
emoji_samples	8
show_tokens	8
step_clean_levels	9
step_clean_names	11
step_dummy_hash	13
step_lda	16
step_lemma	19
step_ngram	21
step_pos_filter	23
step_sentence_embeddings	25
step_sequence_onehot	28
step_stem	31
step_stopwords	33
step_text_normalization	36
step_textfeature	38
step_texthash	40
step_tf	43
step_tfidf	46
step_tokenfilter	49
step_tokenize	52
step_tokenize_bpe	57
step_tokenize_sentencepiece	60
step_tokenize_wordpiece	62
step_tokenmerge	64
step_untokenize	66
step_word_embeddings	68
tokenlist	71

Index	72
--------------	-----------

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()
```

count_functions	<i>List of all feature counting functions</i>
-----------------	---

Description

List of all feature counting functions

Usage

```
count_functions
```

Format

Named list of all ferature counting functions

n_words Number of words.

n_uq_words Number of unique words.

n_charS Number of characters. Not counting urls, hashtags, mentions or white spaces.

n_uq_charS Number of unique characters. Not counting urls, hashtags, mentions or white spaces.

n_digits Number of digits.

n_hashtags Number of hashtags, word preceded by a '#'.

n_uq_hashtags Number of unique hashtags, word preceded by a '#'.

n_mentions Number of mentions, word preceded by a '@'.

n_uq_mentions Number of unique mentions, word preceded by a '@'.

n_commas Number of commas.

n_periods Number of periods.

n_exclaims Number of exclamation points.

n_extraspaces Number of times 2 or more consecutive whitespace characters (spaces, tabs, or newlines) have been used in a row.

n_caps Number of upper case characters.

n_loweres Number of lower case characters.

n_urls Number of urls.

n_uq_urls Number of unique urls.

n_nonasciis Number of non ascii characters.

n_puncts Number of punctuations characters, not including exclamation points, periods and commas.

first_person Number of "first person" words.

first_personp Number of "first person plural" words.

second_person Number of "second person" words.

second_personp Number of "second person plural" words.

third_person Number of "third person" words.

to_be Number of "to be" words.

prepositions Number of preposition words.

Details

In this function we refer to "first person", "first person plural" and so on. This list describes what words are contained in each group.

first person I, me, myself, my, mine, this.

first person plural we, us, our, ours, these.

second person you, yours, your, yourself.

second person plural he, she, it, its, his, hers.

third person they, them, theirs, their, they're, their's, those, that.

to be am, is, are, was, were, being, been, be, were, be.

prepositions about, below, excepting, off, toward, above, beneath, on, under, across, from, onto, underneath, after, between, in, out, until, against, beyond, outside, up, along, but, inside, over, upon, among, by, past, around, concerning, regarding, with, at, despite, into, since, within, down, like, through, without, before, during, near, throughout, behind, except, of, to, for.

All of the functions below propagate missing values: if an input element is **NA**, the corresponding output element is **NA** as well.

embedding_custom	<i>Embedding Engines for step_sentence_embeddings()</i>
------------------	---

Description

These functions create the **engine** object passed to `step_sentence_embeddings()`. An engine describes how character vectors are turned into numeric embeddings. Engines only hold a reference to the model (a function, or later a model name and server settings), never the model itself, so a prepped recipe stays small.

Usage

```
embedding_custom(fun)
```

Arguments

fun	A function that takes a character vector and returns a numeric matrix with one row per element and one column per embedding dimension. A formula such as <code>~ my_embedder(.x)</code> is converted to a function with <code>rlang::as_function()</code> .
------------	---

Details

`embedding_custom()` is the escape hatch for any backend that doesn't have a dedicated engine. The function is called with the unique, non-missing strings that haven't been embedded yet in the current R session, and must return a numeric matrix with exactly one row per string.

The function is stored in the recipe. If it captures a live external object, such as a Python model through reticulate, that object will not survive `saveRDS()`. Have the function load its model lazily, from a path or an identifier, instead of capturing it.

Value

An object of class `sentence_embedding_engine`.

See Also

`step_sentence_embeddings()`

Other Embedding Engines: `embedding_ollama()`

Examples

```
# A toy embedder that returns the length and number of vowels of each string
toy_embedder <- function(x) {
  cbind(
    nchar = nchar(x),
    vowels = nchar(gsub("[^aeiou]", "", x))
  )
}
```

```
}

embedding_custom(toy_embedder)
```

embedding_ollama	<i>Ollama Embedding Engine</i>
------------------	--------------------------------

Description

`embedding_ollama()` creates an engine for `step_sentence_embeddings()` that gets embeddings from a running **Ollama** server. No Python environment or API key is needed, the engine sends HTTP requests to the server, which is usually running on your own machine.

Usage

```
embedding_ollama(
    model,
    host = "http://localhost:11434",
    text_prefix = NULL,
    normalize = FALSE,
    batch_size = 64
)
```

Arguments

model	A single string giving the name of the Ollama model, such as "all-minilm". The model must already be available on the server, which you can do with <code>ollama pull <model></code> . There is no default, so the model is always visible in your code.
host	A single string giving the address of the Ollama server. Defaults to "http://localhost:11434".
text_prefix	A single string added to the start of every text before it is embedded, or NULL for no prefix. Some models are trained with prefixes, for example "query: " or "passage: ". Defaults to NULL.
normalize	A single logical. If TRUE, each embedding is scaled to have unit length. Defaults to FALSE, which returns the vectors as the server returns them.
batch_size	A single positive whole number giving the maximum number of texts sent in one request. Defaults to 64.

Details

Embeddings are requested from the `/api/embed` endpoint. Requests that fail with a transient error (too many requests, or the server being unavailable) are retried a few times. Any other failure stops `bake()` with an error, so partial results are never returned. Texts longer than the context length of the model are an error rather than being silently truncated.

Before any text is embedded, the engine checks that the server can be reached and that `model` is available on it, and gives an error that says how to fix it if not. The text is sent

to the server at `host`. For a server you don't control, keep in mind that this is where your data goes.

The engine stores the model name and the server address, not the model. A prepped recipe therefore needs the same model to be available on the server when it is baked, and Ollama model names, such as "all-minilm", can point to different model versions over time. The engine doesn't pin a version.

Requires the `httr2` package.

Value

An object of class `sentence_embedding_engine`.

Setting up Ollama

The engine needs an Ollama server that is running and has the model.

1. Install Ollama from <https://ollama.com/download>. See the [Ollama documentation](#) for your system.
2. Make sure the server is running. The desktop apps start it when you open them. Otherwise run `ollama serve` in a terminal and leave it running.
3. Pull the model once, for example `ollama pull all-minilm`. Models are listed at <https://ollama.com/library>.

You can check which models are on the server with `ollama list`.

See Also

[step_sentence_embeddings\(\)](#)

Other Embedding Engines: [embedding_custom\(\)](#)

Examples

```
# Needs a running Ollama server with the model pulled:
# ollama pull all-minilm
library(recipes)

sample_data <- tibble(
  text = c("The cat sat.", "A dog ran.", "Stocks fell sharply today.")
)

rec <- recipe(~text, data = sample_data) |>
  step_sentence_embeddings(
    text,
    engine = embedding_ollama("all-minilm")
  )

rec |>
  prep() |>
  bake(new_data = NULL)
```

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 vectors of a recipe. This function can be useful for diagnostics during 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 <code>recipes::recipe()</code> object.
var	The name of the variable to show tokens for, as a bare name.
n	The 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, n = 2)

library(modeldata)
data(tate_text)

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

step_clean_levels	<i>Clean Categorical Levels</i>
-------------------	---------------------------------

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 recipes::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 <code>recipes::prep.recipe()</code> . 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()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

The levels are cleaned with `janitor::make_clean_names()`, which is the function responsible for the cleaning, 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 is returned with columns `terms`, `original`, `value`, and `id`:

terms character, the selectors or variables selected

original character, the original levels

value character, the cleaned levels

id character, id of this step

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)

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

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> 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.

clean	A named character vector to clean variable names. This is NULL until computed by <code>recipes::prep.recipe()</code> .
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

The variable names are cleaned with `janitor::make_clean_names()`, which is the function responsible for the cleaning.

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 is returned with columns `terms`, `value`, and `id`:

terms character, the new clean variable names

value character, the original variable names

id character, id of this step

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)

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	<i>Indicator Variables via Feature Hashing</i>
-----------------	--

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",
  sparse = "auto",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("dummy_hash")
)

```

Arguments

recipe	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> for more details.
role	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.

<code>columns</code>	A character vector of the selected variable names. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>signed</code>	A logical, indicating whether to use a signed hash-function (generating values of -1, 0, or 1), to reduce collisions when hashing. Defaults to <code>TRUE</code> .
<code>num_terms</code>	An integer, the number of variables to output. Defaults to 32.
<code>collapse</code>	A logical. Should all of the selected columns be collapsed into a single column to create a single set of hashed features? Defaults to <code>FALSE</code> .
<code>prefix</code>	A character string that will be the prefix to the resulting new variables. See notes below.
<code>sparse</code>	A single string. Should the columns produced be sparse vectors? Can take the values "yes", "no", and "auto". If <code>sparse = "auto"</code> then workflows can determine the best option. Defaults to "auto".
<code>keep_original_cols</code>	A logical to keep the original variables in the output. Defaults to <code>FALSE</code> .
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	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 via `text2vec::hash_vectorizer()`.

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 columns will have names that begin with `prefix`, then the name of the variable, then the index of the hash, all separated by `_`. The indices are padded with zeros to the same width. For example, with the default `prefix = "dummyhash"` and a variable named `text`, `num_terms = 12` gives `dummyhash_text_01` to `dummyhash_text_12`. If `collapse = TRUE`, the variable name is replaced by the names of all selected variables joined by `_`, such as `dummyhash_text_g_01` when `text` and `g` are selected.

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 is returned with columns `terms`, `value`, `num_terms`, `collapse`, and `id`:

terms character, the selectors or variables selected

value logical, whether a signed hashing was performed

num_terms integer, number of terms

collapse logical, were the columns collapsed

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- **signed**: Signed Hash Value (type: logical, default: TRUE)
- **num_terms**: # Hash Features (type: integer, default: 32)

Sparse data

This step produces sparse columns if `sparse = "yes"` is being set. The default value `"auto"` won't trigger production of sparse columns if a recipe is `recipes::prep()`ed, but allows for a workflow to toggle to `"yes"` or `"no"` depending on whether the model supports `recipes::sparse_data` and if the model is expected to run faster with the data.

The mechanism for determining how much sparsity is produced isn't perfect, and there will be times when you want to manually overwrite by setting `sparse = "yes"` or `sparse = "no"`.

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_sentence_embeddings()`, `step_sequence_onehot()`, `step_textfeature()`

Examples

```
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 `recipes::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 <code>recipes::selections()</code> for more details.
<code>role</code>	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>lda_models</code>	A <code>WarpLDA</code> model object from <code>text2vec::LDA</code> . If <code>NULL</code> , the default, it will train its model based on the training data. Look at the examples for how to fit a <code>WarpLDA</code> model.
<code>num_topics</code>	A whole number, the desired number of latent topics. Defaults to 10.
<code>prefix</code>	A character string that will be the prefix to the resulting new variables. See notes below.
<code>keep_original_cols</code>	A logical to keep the original variables in the output. Defaults to <code>FALSE</code> .
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	A character string that is unique to this step to identify it.

Details

The new columns will have names that begin with `prefix`, then the name of the variable, then the index of the topic, all separated by `_`. For example, with the default `prefix = "lda"` and a variable named `text`, the columns will be named `lda_text_1`, `lda_text_2`, and so on up to `num_topics`.

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 is returned with columns `terms`, `num_topics`, and `id`:

terms character, the selectors or variables selected

num_topics integer, number of topics

id character, id of this step

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_tf\(\)](#), [step_tfidf\(\)](#), [step_word_embeddings\(\)](#)

Examples

```
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

*Lemma*ization 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

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> 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 vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	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 is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

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

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> 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 vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>num_tokens</code>	The number of tokens in the n-gram. This must be an integer greater than or equal to 1. Defaults to 3.
<code>min_num_tokens</code>	The minimum number of tokens in the n-gram. This must be an integer greater than or equal to 1 and smaller than or equal to <code>num_tokens</code> . Defaults to 3.
<code>delim</code>	A single string, the separator between words in an n-gram. Defaults to <code>"_"</code> .

skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
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 will be ordered 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 will contain 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 is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 1 tuning parameters:

- `num_tokens`: Number of tokens (type: integer, default: 3)

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",
  remove_tags = NULL,
  skip = FALSE,
  id = rand_id("pos_filter")
)

```

Arguments

<code>recipe</code>	A recipes::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 vector of the selected variable names. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>keep_tags</code>	A character vector of part of speech tags to keep. See details for the complete list of tags. Defaults to "NOUN". Can't be used together with <code>remove_tags</code> .
<code>remove_tags</code>	A character vector of part of speech tags to remove, keeping all other tokens. See details for the complete list of tags. Defaults to NULL. Can't be used together with <code>keep_tags</code> .
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<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 is returned with columns `terms` and `id`:

`terms` character, the selectors or variables selected

`id` character, id of this step

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_sentence_embeddings

Sentence Embeddings of Text

Description

`step_sentence_embeddings()` creates a *specification* of a recipe step that will convert a character variable into dense embedding dimensions by running each string through an embedding model.

Usage

```
step_sentence_embeddings(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  engine,
  prefix = "sentembed",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("sentence_embeddings")
)
```

Arguments

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> for more details.
<code>role</code>	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>engine</code>	An engine object created by one of the embedding engine functions, such as <code>embedding_custom()</code> . The engine defines how strings are turned into vectors and holds the settings needed to do so.
<code>prefix</code>	A single string used as the prefix of the new columns. See Details for how the column names are built. Defaults to <code>"sentembed"</code> .
<code>keep_original_cols</code>	A logical to keep the original variables in the output. Defaults to <code>FALSE</code> .
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	A character string that is unique to this step to identify it.

Details

Unlike `step_word_embeddings()`, which looks up and aggregates the vectors of tokens, this step takes a character (or factor) column directly and returns one dense vector per row, as produced by the embedding model. It doesn't use `step_tokenize()`.

The new columns have names that begin with `prefix`, then the name of the variable being embedded, then the position of the dimension. For example, using the default prefix and a variable named `text`, the new columns are `sentembed_text_1`, `sentembed_text_2`, etc.

Value

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

What is stored in the recipe

Embedding models are large, so the step never stores the model. It stores the engine, which is a reference to the model, together with the number of dimensions seen when the recipe

was prepped. This keeps prepped recipes small, but it also means a prepped recipe can only be baked where the engine can run. A recipe that is moved to another machine needs whatever the engine depends on to be available there.

If the engine returns a different number of dimensions when baking than it did when prepping, `bake()` errors instead of returning columns the downstream model wasn't trained on.

Missing values and caching

Missing values stay missing: a row with NA text gets NA in all of its embedding columns, and is never sent to the engine. Empty strings are sent to the engine as they are.

Embedding is usually the slowest part of a recipe, and `bake()` embeds new data every time it is called. Within an R session, embeddings are cached per engine and string, so repeated strings, and the training data that is embedded both by `prep()` and by `bake()`, are only embedded once. The cache lives outside the recipe and is cleared when the R session ends.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms`, `engine`, `model`, `dimension`, and `id`:

terms character, the selectors or variables selected

engine character, the name of the engine

model character, the name of the model, NA if the engine doesn't have one

dimension integer, the number of embedding dimensions, NA until the recipe is prepped

id character, id of this step

Case weights

The underlying operation does not allow for case weights.

See Also

`embedding_custom()` for the available engines, and `step_word_embeddings()` for embeddings that are aggregated from tokens.

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

Examples

```
library(recipes)

# A toy embedder, replace it with a call to a real embedding model
toy_embedder <- function(x) {
  cbind(
    nchar = nchar(x),
    vowels = nchar(gsub("[^aeiou]", "", x))
  )
}
```

```

sample_data <- tibble(
  text = c("The cat.", "The cat ran.", "A longer sentence here."),
  text_label = c("short", "short", "long")
)

rec <- recipe(text_label ~ ., data = sample_data) |>
  step_sentence_embeddings(text, engine = embedding_custom(toy_embedder))

obj <- rec |>
  prep()

bake(obj, new_data = NULL)

tidy(rec, number = 1)
tidy(obj, number = 1)

```

`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

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> for more details.

role	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
sequence_length	A whole number, the number of characters to keep before discarding. Defaults to 100.
padding	A single string, either "pre" or "post", determining whether to pad before or after each sequence. Defaults to "pre".
truncating	A single string, either "pre" or "post", determining whether to remove values from the beginning or the end of sequences longer than <code>sequence_length</code> . Defaults to "pre".
vocabulary	A character vector of the characters to be mapped to integers. Characters not in the vocabulary will be encoded as 0. If <code>NULL</code> , the default, all characters found in the training data are used.
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 <code>FALSE</code> .
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
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.

The new columns will have names that begin with `prefix`, then the name of the variable, then the position in the sequence, all separated by `_`. For example, with the default `prefix = "seq1hot"`, a variable named `text`, and `sequence_length = 3`, the columns will be named `seq1hot_text_1`, `seq1hot_text_2`, and `seq1hot_text_3`.

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 is returned with columns `terms`, `vocabulary`, `token`, and `id`:

terms character, the selectors or variables selected

vocabulary integer, index

token character, text corresponding to the index

id character, id of this step

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_sentence_embeddings()`, `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

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> 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 vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>options</code>	A list of options passed to the stemmer function. Note that the arguments <code>X</code> and <code>FUN</code> should not be passed here. Defaults to <code>list()</code> .
<code>custom_stemmer</code>	A custom stemming function. If <code>NULL</code> , the default, <code>SnowballC::wordStem()</code> is used.
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	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 is returned with columns `terms`, `is_custom_stemmer`, and `id`:

terms character, the selectors or variables selected

is_custom_stemmer logical, indicate if custom stemmer was used

id character, id of this step

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

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> 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 vector of the selected variable names. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>language</code>	A single string indicating the language of the stop words by ISO 639-1 coding scheme. Ignored if <code>custom_stopword_source</code> is used. Defaults to "en".
<code>keep</code>	A logical. Should the stop words be kept instead of discarded? Defaults to FALSE.
<code>stopword_source</code>	A single string indicating the stop words source, as listed in <code>stopwords::stopwords_getsources</code> . Ignored if <code>custom_stopword_source</code> is used. Defaults to "snowball".
<code>custom_stopword_source</code>	A character vector of words to use as a custom list of stop words, which cater to the user's specific problem. If NULL, the default, the list is taken from <code>stopword_source</code> and <code>language</code> . If used, <code>language</code> and <code>stopword_source</code> are ignored.
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	A character string that is unique to this step to identify it.

Details

Stop words are words which sometimes are removed 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 words to keep 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 is returned with columns `terms`, `value`, `keep`, and `id`:

terms character, the selectors or variables selected

value character, name of stop word list

keep logical, whether stop words are removed or kept

id character, id of this step

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)
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_text_normalization
```

Normalization of Character Variables

Description

`step_text_normalization()` creates a *specification* of a recipe step that will perform Unicode Normalization on character variables.

Usage

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

Arguments

<code>recipe</code>	A recipes::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 vector of the selected variable names. This is <code>NULL</code> until the step is trained by recipes::prep.recipe() , and should not be set by hand.
<code>normalization_form</code>	A single 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.
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by recipes::bake() ? While all operations are baked when recipes::prep() 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 = TRUE</code> , as it may affect the computations for subsequent operations.

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 is returned with columns **terms**, **normalization_form**, and **id**:

terms character, the selectors or variables selected

normalization_form character, type of normalization

id character, id of this step

Case weights

The underlying operation does not allow for case weights.

See Also

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

Examples

```
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_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 = count_functions,
  prefix = "textfeature",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("textfeature")
)
```

Arguments

<code>recipe</code>	A recipes::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>	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character vector of the selected variable names. This is NULL until the step is trained by recipes::prep.recipe() , and should not be set by hand.
<code>extract_functions</code>	A named list of feature extracting functions. Each function should take a character vector and return a numeric vector of the same length. Defaults to <code>count_functions</code> . See details for more information.
<code>prefix</code>	A character string that will be the prefix to the resulting new variables. See notes below.
<code>keep_original_cols</code>	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()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
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.

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.

The new columns will have names that begin with `prefix`, then the name of the variable, then the name of the function in `extract_functions`, all separated by `_`. For example, with the default `prefix = "textfeature"`, a variable named `text`, and the default `count_functions`, the columns will include `textfeature_text_n_words` and `textfeature_text_n_uq_words`.

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 is returned with columns `terms`, `functions`, and `id`:

terms character, the selectors or variables selected

functions character, name of feature functions

id character, id of this step

Case weights

The underlying operation does not allow for case weights.

See Also

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

Examples

```
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)

```

step_texthash

Feature Hashing of Tokens

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",
  sparse = "auto",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("texthash")
)

```

Arguments

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> for more details.
<code>role</code>	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>signed</code>	A logical, indicating whether to use a signed hash-function to reduce collisions when hashing. Defaults to <code>TRUE</code> .
<code>num_terms</code>	An integer, the number of variables to output. Defaults to 1024.
<code>prefix</code>	A character string that will be the prefix to the resulting new variables. See notes below.
<code>sparse</code>	A single string. Should the columns produced be sparse vectors? Can take the values "yes", "no", and "auto". If <code>sparse = "auto"</code> then workflows can determine the best option. Defaults to "auto".
<code>keep_original_cols</code>	A logical to keep the original variables in the output. Defaults to <code>FALSE</code> .
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	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 via `text2vec::hash_vectorizer()`.

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 columns will have names that begin with `prefix`, then the name of the variable, then the index of the hash, all separated by `_`. The indices are padded with zeros to the

same width. For example, with the default `prefix = "texthash"` and a variable named `text`, `num_terms = 12` gives `texthash_text_01` to `texthash_text_12`, and `num_terms = 101` gives `texthash_text_001` to `texthash_text_101`.

Value

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

Tidying

When you run `tidy()` this step, a tibble is returned with columns `terms`, `value` and `id`:

terms character, the selectors or variables selected

value logical, is it signed?

length integer, number of terms

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- **signed**: Signed Hash Value (type: logical, default: TRUE)
- **num_terms**: # Hash Features (type: integer, default: 1024)

Sparse data

This step produces sparse columns if `sparse = "yes"` is being set. The default value `"auto"` won't trigger production of sparse columns if a recipe is `recipes::prep()`ed, but allows for a workflow to toggle to `"yes"` or `"no"` depending on whether the model supports `recipes::sparse_data` and if the model is expected to run faster with the data.

The mechanism for determining how much sparsity is produced isn't perfect, and there will be times when you want to manually overwrite by setting `sparse = "yes"` or `sparse = "no"`.

Case weights

The underlying operation does not allow for case weights.

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_tf()`, `step_tfidf()`, `step_word_embeddings()`

Examples

```
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_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",
  sparse = "auto",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("tf")
)
```

Arguments

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> for more details.
<code>role</code>	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>weight_scheme</code>	A single string determining the weighting scheme for the term frequency calculations. Must be one of "binary", "raw count", "term frequency", "log normalization", or "double normalization". See details for what each scheme does. Note that <code>sparse = "yes"</code> has no effect with "double normalization", as it doesn't produce sparse data. Defaults to "raw count".
<code>weight</code>	A single numeric weight used if <code>weight_scheme</code> is set to "double normalization". Defaults to 0.5.
<code>vocabulary</code>	A character vector of the tokens to create variables for. If <code>NULL</code> , the default, all tokens found in the training data are used.
<code>res</code>	A named list of the tokens used for each selected variable. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> and should not be set by hand.
<code>prefix</code>	A character string that will be the prefix to the resulting new variables. See notes below.
<code>sparse</code>	A single string. Should the columns produced be sparse vectors? Can take the values "yes", "no", and "auto". If <code>sparse = "auto"</code> then workflows can determine the best option. Defaults to "auto".
<code>keep_original_cols</code>	A logical to keep the original variables in the output. Defaults to <code>FALSE</code> .
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	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 appears 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 by the total number of in-vocabulary tokens 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 columns will have names that begin with `prefix`, then the name of the variable, then the token, all separated by `_`. For example, with the default `prefix = "tf"`, a variable named `text`, and the token `"word"`, the new column will be named `tf_text_word`.

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 is returned with columns `terms`, `value`, and `id`:

terms character, the selectors or variables selected

value character, the weighting scheme

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- **weight_scheme**: Term Frequency Weight Method (type: character, default: raw count)
- **weight**: Weight (type: double, default: 0.5)

Sparse data

This step produces sparse columns if `sparse = "yes"` is being set. The default value "auto" won't trigger production of sparse columns if a recipe is `recipes::prep()`ed, but allows for a workflow to toggle to "yes" or "no" depending on whether the model supports `recipes::sparse_data` and if the model is expected to run faster with the data.

The mechanism for determining how much sparsity is produced isn't perfect, and there will be times when you want to manually overwrite by setting `sparse = "yes"` or `sparse = "no"`.

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(medium) |>
  step_tf(medium)

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",
    sparse = "auto",
    keep_original_cols = FALSE,
    skip = FALSE,
    id = rand_id("tfidf")
  )

```

Arguments

recipe	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> for more details.
role	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
vocabulary	A character vector of the tokens to create variables for. If <code>NULL</code> , the default, all tokens found in the training data are used.
res	A named list of numeric vectors holding the IDF weight of each token, one element per selected variable. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> and should not be set by hand.
smooth_idf	A logical. Should the IDF weights be smoothed by adding one to the ratio of the number of documents to the number of documents containing the token? This prevents tokens that appear in every document from getting a weight of zero. Defaults to <code>TRUE</code> .
norm	A single string determining the type of normalization applied to the term frequencies of each document. Must be one of: • "l1": scale by the sum of the term frequencies in the document. • "l2": scale by the square root of the sum of the squared term frequencies in the document. • "none": don't normalize. Defaults to "l1".
sublinear_tf	A logical. Should sublinear term-frequency scaling be applied, replacing the term frequency with $1 + \log(\text{TF})$? Defaults to <code>FALSE</code> .
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
sparse	A single string. Should the columns produced be sparse vectors? Can take the values "yes", "no", and "auto". If <code>sparse = "auto"</code> then workflows can determine the best option. Defaults to "auto".

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()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
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 when `smooth_idf = TRUE`: $\text{idf} = \log(1 + (\# \text{ documents in the corpus}) / (\# \text{ documents where the term appears}))$. When `smooth_idf = FALSE` the `1 +` is dropped.

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 is returned with columns `terms`, `token`, `weight`, and `id`:

terms character, the selectors or variables selected

token character, name of token

weight numeric, the calculated IDF weight

id character, id of this step

The new columns will have names that begin with `prefix`, then the name of the variable, then the token, all separated by `_`. For example, with the default `prefix = "tfidf"`, a variable named `text`, and the token `"word"`, the new column will be named `tfidf_text_word`.

Sparse data

This step produces sparse columns if `sparse = "yes"` is being set. The default value `"auto"` won't trigger production of sparse columns if a recipe is `recipes::prep()`ed, but allows for a workflow to toggle to `"yes"` or `"no"` depending on whether the model supports `recipes::sparse_data` and if the model is expected to run faster with the data.

The mechanism for determining how much sparsity is produced isn't perfect, and there will be times when you want to manually overwrite by setting `sparse = "yes"` or `sparse = "no"`.

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

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> 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 vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>max_times</code>	A number. Maximal number of times a token can appear before getting removed. An integer count, or a proportion between 0 and 1 if <code>percentage = TRUE</code> . Defaults to <code>Inf</code> .
<code>min_times</code>	A number. Minimum number of times a token must appear to avoid getting removed. An integer count, or a proportion between 0 and 1 if <code>percentage = TRUE</code> . Defaults to 0.
<code>percentage</code>	A logical. Should <code>max_times</code> and <code>min_times</code> be interpreted as a proportion of all token occurrences in the training data instead of a count? For example, with <code>percentage = TRUE</code> and <code>min_times = 0.01</code> , tokens that make up less than 1 percent of all tokens are removed. Defaults to <code>FALSE</code> .
<code>max_tokens</code>	An integer. Will only keep the top <code>max_tokens</code> tokens after filtering done by <code>max_times</code> and <code>min_times</code> . Defaults to 100.
<code>filter_fun</code>	A function. This function should take a vector of characters and return a logical vector of the same length. It will be applied to each observation of the data set. If <code>NULL</code> , the default, tokens are filtered using <code>max_times</code> , <code>min_times</code> , <code>percentage</code> , and <code>max_tokens</code> , which are otherwise ignored.

res	A named list of the tokens that will be kept for each selected variable. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> and should not be set by hand.
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

This step allows 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 is returned with columns `terms`, `value`, and `id`:

terms character, the selectors or variables selected

value integer, number of unique tokens

id character, id of this step

Tuning Parameters

This step has 3 tuning parameters:

- `max_times`: Maximum Token Frequency (type: integer, default: `Inf`)
- `min_times`: Minimum Token Frequency (type: integer, default: 0)
- `max_tokens`: # Retained Tokens (type: integer, default: 100)

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

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> 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 vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>training_options</code>	A list of options passed to the tokenizer when it is being trained. Only applicable when <code>engine = "tokenizers.bpe"</code> or <code>engine = "udpipe"</code> . For <code>engine = "udpipe"</code> the list must contain <code>model</code> , a loaded <code>udpipe</code> model as returned by <code>udpipe::udpipe_load_model()</code> . Note that the argument <code>x</code> should not be passed here. Defaults to <code>list()</code> .
<code>options</code>	A list of options passed to the tokenizer. Note that the argument <code>x</code> should not be passed here. For <code>engine = "tokenizers.bpe"</code> these options only apply when the tokenizer is trained (at <code>prep()</code> time); they have no effect on already-trained models at <code>bake()</code> time. For <code>engine = "spacyr"</code> the options are applied each time the data is tokenized, including at <code>bake()</code> time. Defaults to <code>list()</code> .
<code>token</code>	A single string giving the unit for tokenizing. See details for options. Defaults to <code>"words"</code> .
<code>engine</code>	A single string giving the package that will be used for tokenization. See details for options. Defaults to <code>"tokenizers"</code> .
<code>custom_token</code>	A user supplied tokenizer function. Must take a character vector as input and output a list of character vectors. If <code>NULL</code> , the default, <code>token</code> and <code>engine</code> determine the tokenizer; otherwise they are overwritten.
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	A character string that is unique to this step to identify it.

Details

```
options(width = 55)
```

Tokenization is the act of splitting a character vector 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** variables 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 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"
- "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, n = 2)
```

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, n = 2)
```

We can also stop removing punctuation.

```
recipe(~ text, data = text_tibble) |>
  step_tokenize(text,
    options = list(strip_punct = FALSE,
      lowercase = FALSE)) |>
  show_tokens(text, n = 2)
```

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, n = 2)
```

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, n = 2)
```

```
recipe(~ text, data = text_tibble) |>
  step_tokenize(
    text,
    engine = "tokenizers.bpe",
```

```

    training_options = list(vocab_size = 22)
  ) |>
  show_tokens(text, n = 2) |>
  lapply(function(x) gsub(" ", "_", x))

```

udpipe:

- "words"

The upipe engine requires a pre-trained upipe model. Load one with `udpipe::udpipe_load_model()` (use `udpipe::udpipe_download_model()` to download one first) and pass it as the `model` element of the `training_options` argument.

```

model <- upipe::udpipe_load_model("english-ewt-ud-2.5-191206.udpipe")

recipe(~ text, data = text_tibble) |>
  step_tokenize(
    text,
    engine = "udpipe",
    training_options = list(model = model)
  )

```

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, n = 2)

```

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms`, `value`, and `id`:

terms character, the selectors or variables selected

value character, unit of tokenization

id character, id of this step

```

step <- "step_tokenize"
result <- knitr::knit_child("man/rmd/tunable-args.Rmd")
cat(result)

```

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

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 <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> 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 vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>vocabulary_size</code>	An 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 <code>tokenizers.bpe::bpe()</code> when the tokenizer is trained at <code>prep()</code> time. These options have no effect on already-trained models at <code>bake()</code> time. Note that the arguments <code>x</code> , <code>model_path</code> , and <code>vocab_size</code> should not be passed here, use <code>vocabulary_size</code> instead. Defaults to <code>list()</code> .
<code>res</code>	The fitted <code>tokenizers.bpe::bpe()</code> model tokenizer. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> and should not be set by hand.
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<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 is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 1 tuning parameters:

- `vocabulary_size`: # Unique Tokens in Vocabulary (type: integer, default: 1000)

Case weights

The underlying operation does not allow for case weights.

See Also

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

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

Examples

```
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,
  model = NULL,
  options = list(),
  res = NULL,
  skip = FALSE,
  id = rand_id("tokenize_sentencepiece")
)
```

Arguments

recipe	A recipes::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 vector of the selected variable names. This is NULL until the step is trained by recipes::prep.recipe() , and should not be set by hand.
vocabulary_size	An integer, indicating the number of tokens in the final vocabulary. Defaults to 1000. Highly encouraged to be tuned. Ignored when model is supplied.
model	A string giving the path to a pretrained sentencepiece model file, such as one downloaded with sentencepiece::sentencepiece_download_model() or trained with sentencepiece::sentencepiece() . When supplied, no model is trained at prep() time and the same model is used for all selected columns. The model is read when the step is prepped, so the file is not needed afterwards. Defaults to NULL , which trains a new model.

options	A list of options passed to <code>sentencepiece::sentencepiece()</code> when the tokenizer is trained at <code>prep()</code> time. These options have no effect on already-trained models at <code>bake()</code> time. Note that the arguments <code>x</code> , <code>model_dir</code> , and <code>vocab_size</code> should not be passed here, use <code>vocabulary_size</code> instead. Ignored when <code>model</code> is supplied. Defaults to <code>list()</code> .
res	The fitted <code>sentencepiece::sentencepiece()</code> model tokenizer. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> and should not be set by hand.
skip	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
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 is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Case weights

The underlying operation does not allow for case weights.

See Also

`step_untokenize()` to untokenize.

Other Steps for Tokenization: `step_tokenize()`, `step_tokenize_bpe()`, `step_tokenize_wordpiece()`

Examples

```
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)

# Using a pretrained model
model_dir <- tempfile()
dir.create(model_dir)
file <- tempfile(tmpdir = model_dir)
writeLines(as.character(tate_text$medium), file)
trained <- sentencepiece::sentencepiece(
  file,
  vocab_size = 100,
  model_dir = model_dir
)

recipe(~., data = tate_text) |>
  step_tokenize_sentencepiece(medium, model = trained$model_path) |>
  prep() |>
  bake(new_data = NULL, medium) |>
  slice(1:2)

```

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

recipe	A recipes::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 vector of the selected variable names. This is <code>NULL</code> until the step is trained by recipes::prep.recipe() , and should not be set by hand.
vocab	A character vector of vocabulary tokens. Defaults to wordpiece::wordpiece_vocab() .
unk_token	A single string used to represent unknown words. Defaults to "[UNK]".
max_chars	An integer, the maximum length of a word recognized. Defaults to 100.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake() ? While all operations are baked when recipes::prep() 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 = TRUE</code> , as it may affect the computations for subsequent operations.
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 is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Case weights

The underlying operation does not allow for case weights.

See Also

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

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

Examples

```
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",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("tokenmerge")
)
```

Arguments

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> for more details.
<code>role</code>	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character vector of the selected variable names. This is <code>NULL</code> until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>prefix</code>	A character string that will be the prefix to the resulting new variables. See notes below.
<code>keep_original_cols</code>	A logical to keep the original variables in the output. Defaults to <code>FALSE</code> .
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	A character string that is unique to this step to identify it.

Details

All the selected variables are merged into a single new variable named `prefix`, which is "tokenmerge" by default. The name of the selected variables is not part of the new name.

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 is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

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 <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> 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 vector of the selected variable names. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
sep	A single string determining 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()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
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 is returned with columns `terms`, `value`, and `id`:

terms character, the selectors or variables selected

value character, separator used for collapsing

id character, id of this step

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(medium) |>
  step_untokenize(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_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

<code>recipe</code>	A <code>recipes::recipe()</code> 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 <code>recipes::selections()</code> for more details.
<code>role</code>	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as <i>predictors</i> in a model.
<code>trained</code>	A logical to indicate if the quantities for preprocessing have been estimated.
<code>columns</code>	A character vector of the selected variable names. This is NULL until the step is trained by <code>recipes::prep.recipe()</code> , and should not be set by hand.
<code>embeddings</code>	A tibble of pre-trained word embeddings, such as those returned by <code>textdata::embedding_glove6b()</code> . The first column should contain tokens, and additional columns should contain embedding vectors.
<code>aggregation</code>	A single string giving the name of the aggregation function used to combine the embeddings of the tokens in each document. Must be one of "sum", "mean", "min", or "max". Defaults to "sum".
<code>aggregation_default</code>	A single numeric value used for documents where none of the tokens are found in the embeddings. Defaults to 0.
<code>prefix</code>	A character string that will be the prefix to the resulting new variables. See notes below.
<code>keep_original_cols</code>	A logical to keep the original variables in the output. Defaults to FALSE.
<code>skip</code>	A logical. Should the step be skipped when the recipe is baked by <code>recipes::bake()</code> ? While all operations are baked when <code>recipes::prep()</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 = TRUE</code> , as it may affect the computations for subsequent operations.
<code>id</code>	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 variable being embedded, then the name of the variable from the embeddings tibble (usually something like "d7"). For example, using the default "wordembed" prefix, a variable named `text`, and the GloVe embeddings from the `textdata` package (where the column names are `d1`, `d2`, etc), new columns would be `wordembed_text_d1`, `wordembed_text_d2`, 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 is returned with columns `terms`, `embedding_rows`, `aggregation`, and `id`:

terms character, the selectors or variables selected

embedding_rows integer, number of rows in embedding

aggregation character, aggregation

id character, id of this step

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_tfidf()`

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)
```

tokenlist*Create Token Object*

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	A list of character vectors.
lemma	A list of character vectors, must be the same size and shape as tokens .
pos	A list of character vectors, must be the same size and shape as tokens .

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

- * **Embedding Engines**
 - embedding_custom, 5
 - embedding_ollama, 6
- * **Steps for Numeric Variables From Characters**
 - step_dummy_hash, 13
 - step_sentence_embeddings, 25
 - step_sequence_onehot, 28
 - step_textfeature, 38
- * **Steps for Numeric Variables From Tokens**
 - step_lda, 16
 - step_texthash, 40
 - step_tf, 43
 - step_tfidf, 46
 - step_word_embeddings, 68
- * **Steps for Text Cleaning**
 - step_clean_levels, 9
 - step_clean_names, 11
- * **Steps for Text Normalization**
 - step_text_normalization, 36
- * **Steps for Token Modification**
 - step_lemma, 19
 - step_ngram, 21
 - step_pos_filter, 23
 - step_stem, 31
 - step_stopwords, 33
 - step_tokenfilter, 49
 - step_tokenmerge, 64
- * **Steps for Tokenization**
 - step_tokenize, 52
 - step_tokenize_bpe, 57
 - step_tokenize_sentencepiece, 60
 - step_tokenize_wordpiece, 62
- * **Steps for Un-Tokenization**
 - step_untokenize, 66
- * **datasets**
 - emoji_samples, 8
- all_tokenized, 3
- all_tokenized_predictors
 - (all_tokenized), 3
- count_functions, 3
- dplyr::recode_factor(), 10
- embedding_custom, 5
- embedding_custom(), 7, 26, 27
- embedding_ollama, 6
- embedding_ollama(), 5
- emoji_samples, 8
- janitor::make_clean_names(), 10, 12
- recipes::bake(), 10, 12, 14, 17, 19, 22, 24, 26, 29, 31, 34, 36, 39, 41, 44, 48, 51, 53, 58, 61, 63, 65, 67, 69
- recipes::has_role(), 3
- recipes::prep(), 10, 12, 14, 15, 17, 19, 22, 24, 26, 29, 31, 34, 36, 39, 41, 42, 44, 45, 48, 49, 51, 53, 58, 61, 63, 65, 67, 69
- recipes::prep.recipe(), 10, 12, 14, 17, 19, 21, 24, 26, 29, 31, 34, 36, 38, 41, 44, 47, 50, 51, 53, 58, 60, 61, 63, 65, 67, 69
- recipes::recipe(), 8, 9, 11, 13, 16, 19, 21, 23, 26, 28, 31, 34, 36, 38, 41, 44, 47, 50, 53, 58, 60, 63, 65, 67, 69
- recipes::selections(), 9, 11, 13, 17, 19, 21, 23, 26, 28, 31, 34, 36, 38, 41, 44, 47, 50, 53, 58, 60, 63, 65, 67, 69
- recipes::sparse_data, 15, 42, 45, 49
- recipes::step_dummy(), 15
- recipes::step_factor2string(), 10, 12
- recipes::step_mutate(), 54
- recipes::step_novel(), 10, 12
- recipes::step_other(), 10, 12

recipes::step_regex(), 10, 12
 recipes::step_string2factor(), 10, 12
 recipes::step_unknown(), 10, 12
 rlang::as_function(), 5

 saveRDS(), 5
 sentencepiece::sentencepiece(), 60, 61
 sentencepiece::sentencepiece_download_model(), 60
 show_tokens, 8
 SnowballC::wordStem(), 31
 step_clean_levels, 9
 step_clean_levels(), 12
 step_clean_names, 11
 step_clean_names(), 10
 step_dummy_hash, 13
 step_dummy_hash(), 27, 30, 39
 step_lda, 16
 step_lda(), 42, 46, 49, 70
 step_lemma, 19
 step_lemma(), 22, 24, 32, 35, 52, 66
 step_ngram, 21
 step_ngram(), 20, 24, 32, 35, 52, 66
 step_pos_filter, 23
 step_pos_filter(), 20, 22, 32, 35, 52, 66
 step_sentence_embeddings, 25
 step_sentence_embeddings(), 5–7, 15, 30, 39
 step_sequence_onehot, 28
 step_sequence_onehot(), 15, 27, 39
 step_stem, 31
 step_stem(), 20, 22, 24, 35, 52, 66
 step_stopwords, 33
 step_stopwords(), 20, 22, 24, 32, 52, 66
 step_text_normalization, 36
 step_text_normalization(), 42
 step_textfeature, 38
 step_textfeature(), 15, 27, 30
 step_texthash, 40
 step_texthash(), 18, 37, 46, 49, 70
 step_tf, 43, 44, 51
 step_tf(), 18, 42, 49, 70
 step_tfidf, 46, 48, 51
 step_tfidf(), 18, 42, 46, 70
 step_tokenfilter, 44, 48, 49
 step_tokenfilter(), 20, 22, 24, 32, 35, 66
 step_tokenize, 52
 step_tokenize(), 20, 22, 24, 26, 32, 35, 42, 46, 49, 52, 59, 61, 64, 66, 67, 70
 step_tokenize_bpe, 57
 step_tokenize_bpe(), 57, 61, 64
 step_tokenize_sentencepiece, 60
 step_tokenize_sentencepiece(), 57, 59, 64
 step_tokenize_wordpiece, 62
 step_tokenize_wordpiece(), 57, 59, 61
 step_tokenmerge, 64
 step_tokenmerge(), 20, 22, 24, 32, 35, 52
 step_untokenize, 66
 step_untokenize(), 57, 59, 61, 64
 step_word_embeddings, 68
 step_word_embeddings(), 18, 26, 27, 42, 46, 49
 stopwords::stopwords_getsources(), 34
 stringi::stri_trans_nfc(), 36

 text2vec::hash_vectorizer(), 14, 41
 text2vec::LDA, 17
 textdata::embedding_glove6b(), 69
 tidy(), 10, 12, 15, 17, 20, 22, 24, 27, 30, 32, 35, 37, 39, 42, 45, 48, 51, 56, 59, 61, 63, 65, 67, 70
 tidy.step_clean_levels
 (step_clean_levels), 9
 tidy.step_clean_names
 (step_clean_names), 11
 tidy.step_dummy_hash
 (step_dummy_hash), 13
 tidy.step_lda (step_lda), 16
 tidy.step_lemma (step_lemma), 19
 tidy.step_ngram (step_ngram), 21
 tidy.step_pos_filter
 (step_pos_filter), 23
 tidy.step_sentence_embeddings
 (step_sentence_embeddings), 25
 tidy.step_sequence_onehot
 (step_sequence_onehot), 28
 tidy.step_stem (step_stem), 31
 tidy.step_stopwords (step_stopwords), 33
 tidy.step_text_normalization
 (step_text_normalization), 36

`tidy.step_textfeature`
 (*step_textfeature*), 38
`tidy.step_texthash` (*step_texthash*),
 40
`tidy.step_tf` (*step_tf*), 43
`tidy.step_tfidf` (*step_tfidf*), 46
`tidy.step_tokenfilter`
 (*step_tokenfilter*), 49
`tidy.step_tokenize` (*step_tokenize*),
 52
`tidy.step_tokenize_bpe`
 (*step_tokenize_bpe*), 57
`tidy.step_tokenize_sentencepiece`
 (*step_tokenize_sentencepiece*),
 60
`tidy.step_tokenize_wordpiece`
 (*step_tokenize_wordpiece*), 62
`tidy.step_tokenmerge`
 (*step_tokenmerge*), 64
`tidy.step_untokenize`
 (*step_untokenize*), 66
`tidy.step_word_embeddings`
 (*step_word_embeddings*), 68
`token`, 3, 19–21, 23, 31, 33, 40, 43, 46, 49,
 52, 54, 57, 60, 62, 64, 66–68
`tokenizers.bpe::bpe()`, 58
`tokenizers::tokenize_words`, 55
`tokenlist`, 71, 71
`tokens`, 20, 22, 24, 32, 35, 42, 46, 49, 52,
 66, 67, 70

`udpipe::udpipe_download_model()`, 56
`udpipe::udpipe_load_model()`, 53, 56

`wordpiece::wordpiece_vocab()`, 63