Package: textrecipes (via r-universe)

November 25, 2026

Title Extra 'Recipes' for Text Processing

Version 1.2.0

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

License MIT + file LICENSE

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

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

Depends R (>= 4.1), recipes (>= 1.4.0)

Imports cli, dplyr, generics (>= 0.1.0), glue, lifecycle, magrittr, Matrix, purrr, rlang (>= 1.1.0), SnowballC, sparsevctrs (>= 0.3.0), tibble, tokenizers, utils, vctrs

Suggests covr, dials (>= 1.2.0), hardhat, httr2 (>= 1.0.0), janitor, knitr, modeldata, reticulate, rmarkdown, safetensors, sentencepiece, spacyr, stopwords, stringi, testthat (>= 3.0.0), text2vec, tokenizers.bpe, torch, 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

SystemRequirements ``GNU make''

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Emil Hvitfeldt [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0679-1945>), Michael W. Kearney
[cph] (author of count_functions), Posit Software, PBC [cph,
fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Emil Hvitfeldt <emil.hvitfeldt@posit.co>

Config/pak/sysreqs make libicu-dev

Repository https://cranhaven.r-universe.dev

Date/Publication 2026-11-25 02:02:00 UTC

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/textrecipes

RemoteSha 14bdf5a0d186903286f34b295448e792681813d0

RemoteSubdir textrecipes

Contents

all_tokenized . . . 3
count_functions . . . 3
download_embeddings_model2vec . . . 5
embedding_custom . . . 6
embedding_huggingface . . . 7
embedding_model2vec . . . 9
embedding_ollama . . . 11
emoji_samples . . . 13
read_embeddings_model2vec . . . 13
show_tokens . . . 14
step_clean_levels . . . 15
step_clean_names . . . 17
step_dummy_hash . . . 19
step_lda . . . 22
step_lemma . . . 24
step_ngram . . . 26
step_pos_filter . . . 28
step_sentence_embeddings . . . 30
step_sequence_onehot . . . 33
step_stem . . . 36
step_stopwords . . . 38
step_text_normalization . . . 41
step_textfeature . . . 42
step_texthash . . . 45
step_tf . . . 48
step_tfidf . . . 51
step_token_replace . . . 54
step_tokenfilter . . . 56
step_tokenize . . . 59
step_tokenize_bpe . . . 65

step_tokenize_sentencepiece 67
step_tokenize_wordpiece 69
step_tokenmerge 71
step_untokenize 73
step_word_embeddings 75
tokenlist 78

Index **79**

`all_tokenized` *Role Selection*

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` *List of all feature counting functions*

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_lowers` 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.

download_embeddings_model2vec *Download a Pretrained Model2Vec Model*

Description

`download_embeddings_model2vec()` downloads the files of a Model2Vec model, so they can be read with `read_embeddings_model2vec()`.

Usage

```
download_embeddings_model2vec(url, path = tempfile("model2vec-"))
```

Arguments

`url`	A string giving the URL of the folder that holds the model files. For a model on the Hugging Face Hub this has the form `https://huggingface.co/<owner>/<model>/resolve/<revi` where `<revision>` is a branch name such as `main` or a commit hash. Use a commit hash to always get the same version of the model.
`path`	A string giving the path to the folder to download the files into. The folder is created if it doesn't exist, and files in it with the same names are overwritten. Defaults to a new temporary folder, which is removed when the R session ends.

Details

Two files are downloaded: `model.safetensors`, which holds the vectors, and `vocab.txt`, which holds the tokens. The files of a small model are about 30 MB, so the download can take a while. The downloaded files are not needed once a recipe using the embeddings has been prepped.

The embeddings are tied to the tokenizer the model was trained with. See `vignette("static-embeddings", package = "textrecipes")` for how to use a downloaded model in a recipe.

Value

The path to the folder holding the files, invisibly.

See Also

`read_embeddings_model2vec()` to read the downloaded files.

Examples

```
url <- paste0(
  "https://huggingface.co/minishlab/potion-base-8M/resolve/",
  "bf8b056651a2c21b8d2565580b8569da283cab23"
)

model_dir <- download_embeddings_model2vec(url)
embeddings <- read_embeddings_model2vec(model_dir)
```

`embedding_custom` *Embedding Engines for* `step_sentence_embeddings()`

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 `~ my_embedder(.x)` is converted to a function with `rlang::as_function()`.

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_huggingface()`, `embedding_model2vec()`, `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_huggingface` *Hugging Face Embedding Engine*

Description

`embedding_huggingface()` creates an engine for `step_sentence_embeddings()` that runs a sentence-transformers model from the Hugging Face Hub on your own machine, using Python through reticulate.

Usage

```
embedding_huggingface(
  model,
  revision,
  device = NULL,
  text_prefix = NULL,
  normalize = FALSE,
  batch_size = 32
)
```

Arguments

`model`	A single string giving the id of the model on the Hugging Face Hub, such as `"sentence-transformers/all-MiniLM-L6-v2"`. There is no default, so the model is always visible in your code.
`revision`	A single string giving the version of the model to use, preferably the hash of a commit. A branch name such as `"main"` is accepted, but it can point to a different version later on, which defeats the purpose of pinning. See the details for how to find the hash. There is no default, so the version is always visible in your code.
`device`	A single string giving the device to run the model on: `"cpu"`, `"cuda"`, `"mps"`, or a numbered device such as `"cuda:1"`. Defaults to `NULL`, which lets sentence-transformers choose a device. The choice then depends on the machine, so set it if you want the recipe to behave in the same way everywhere.
`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: "` for the e5 models. 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 model returns them. Some models already return vectors of unit length.
`batch_size`	A single positive whole number giving the maximum number of texts the model embeds at once. Larger batches are faster, up to the memory of the device. Defaults to `32`.

Details

The embeddings are what `SentenceTransformer$encode()` returns for the model, so the pooling and normalization steps that are part of the model are used. Texts longer than the maximum sequence length of the model are cut off by the model, as they are in sentence-transformers.

The model is downloaded from the Hub the first time it is used, and is kept in the Hugging Face cache on your machine, which is usually `~/.cache/huggingface`. After that the model runs locally, and your text is not sent anywhere. Loading a model into memory takes a while, so the loaded model is kept for the rest of the R session. Each parallel worker loads its own copy.

The engine stores the model id, the revision, and the other settings, not the model. A prepped recipe therefore needs the Python environment and the model at the same revision to be available where it is baked. They are downloaded on first use if the machine can reach the Hub.

Before any text is embedded, the engine checks that the Python module can be imported, that the device is available, and that the model can be loaded at the revision, and gives an error that says what is wrong if not.

Requires the reticulate package and the Python module `sentence_transformers`. Models that need `trust_remote_code` are not supported.

Value

An object of class `sentence_embedding_engine`.

Pinning the revision

A model repository on the Hub can be updated, so the same model id can give different embeddings over time. The `revision` argument fixes the version. To find the hash of the latest commit, open the model page and then *Files and versions*, or request `https://huggingface.co/api/models/<model>` and read the `sha` field. Models that are pinned to a commit give the same embeddings every time, up to small differences between devices.

Setting up Python

The engine needs the Python module `sentence_transformers`, which brings in torch. If you don't already have a Python environment you want to use, either of these works:

- Install it into the environment reticulate uses, for example with `reticulate::py_install("sentence-transformers`
- Before Python is started in the R session, run `reticulate::py_require("sentence-transformers")`, and reticulate sets up a temporary environment for you.

See the reticulate documentation for choosing an environment.

macOS

On macOS, loading torch in an R session that already has some R packages loaded, such as data.table (which recipes depends on), can abort R with the error `OMP: Error #15: Initializing libomp.dylib, but found libomp.dylib already initialized.` This happens because both load their own OpenMP library. To avoid it, the engine sets the environment variables `KMP_DUPLICATE_LIB_OK` to `"TRUE"` and `OMP_NUM_THREADS` to `"1"` before the model is loaded, unless you have already set them, and tells you when it does. The model then runs on one thread, which is slower. The OpenMP

developers describe KMP_DUPLICATE_LIB_OK as an unsafe workaround that can in principle give wrong results. In testing, the embeddings matched those of Python to within 1e-6. Set both variables yourself, for example in .Renviron, to use other values or to stop the engine from setting them.

The variables only help if they are set before Python is started. If you have already used reticulate to start Python in the R session, for example with reticulate::py_config(), the engine is too late to prevent the crash. Restart R and set the variables in .Renviron in that case.

See Also

step_sentence_embeddings()

Other Embedding Engines: embedding_custom(), embedding_model2vec(), embedding_ollama()

Examples

```
# Needs Python with the sentence-transformers module, and downloads the model
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_huggingface(
      "sentence-transformers/all-MiniLM-L6-v2",
      revision = "1110a243fdf4706b3f48f1d95db1a4f5529b4d41",
      device = "cpu"
    )
  )

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

---

embedding_model2vec *Model2Vec Embedding Engine*

Description

embedding_model2vec() creates an engine for step_sentence_embeddings() that embeds text with a Model2Vec model. The model is a table of token vectors that is read from disk and used in R, so it needs neither Python nor a server, and text never leaves your machine.

Usage

```
embedding_model2vec(path, normalize = TRUE, max_tokens = 512)
```

Arguments

path	A single string giving the path to a folder holding a Model2Vec model, which must contain the files `model.safetensors` and `vocab.txt`. `download_embeddings_model2vec()` downloads such a folder. The path is expanded and stored in the engine.
normalize	A single logical. If `TRUE`, each embedding is scaled to have unit length, as the Model2Vec Python package does by default. Defaults to `TRUE`.
max_tokens	A single positive whole number giving the maximum number of tokens of a text that are used. Later tokens are ignored. Defaults to `512`, as in the Model2Vec Python package.

Details

The embedding of a text is the mean of the vectors of its tokens, scaled to unit length if `normalize = TRUE`. This is what Model2Vec does. The text is tokenized with the WordPiece vocabulary of the model using `wordpiece::wordpiece_tokenize()`, tokens that are not in the vocabulary (`"[UNK]"`) are ignored, and no special tokens are added. A text without any known tokens gets a vector of zeros.

This follows the way Model2Vec builds an embedding, but it is not guaranteed to match the Python `model2vec` package exactly. Tokenizer details such as the handling of accents or unusual characters may differ, so compare a few texts with the Python package if you need identical values. Only models that use a WordPiece vocabulary, such as the English potion models, are supported. Models with a different tokenizer, with quantized weights, or with extra weights for the tokens can't be used.

The engine stores the path to the model folder, not the model. A prepped recipe therefore needs the model files to be at the same path where it is baked. The model is read the first time it is used and is then kept for the rest of the R session. Each parallel worker reads its own copy.

Requires the wordpiece, safetensors and torch packages. The torch package is only used to read the file, and the model is not run with it.

Value

An object of class `sentence_embedding_engine`.

See Also

`step_sentence_embeddings()`, `download_embeddings_model2vec()`, and `vignette("static-embeddings", package = "textrecipes")` for using the same model with `step_word_embeddings()`.

Other Embedding Engines: `embedding_custom()`, `embedding_huggingface()`, `embedding_ollama()`

Examples

```
# Downloads the model files
library(recipes)

url <- paste0(
  "https://huggingface.co/minishlab/potion-base-8M/resolve/",
  "bf8b056651a2c21b8d2565580b8569da283cab23"
)
model_dir <- download_embeddings_model2vec(url)
```

```
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_model2vec(model_dir))

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

`embedding_ollama` *Ollama Embedding Engine*

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 `ollama pull <model>`. 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()`, `embedding_huggingface()`, `embedding_model2vec()`

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` *Sample sentences with emojis*

Description

This data set is primarily used for examples.

Usage

```
emoji_samples
```

Format

tibble with 1 column

`read_embeddings_model2vec`
Read Pretrained Static Embeddings From Files

Description

These functions read downloaded static embeddings into the tibble that the `embeddings` argument of `step_word_embeddings()` expects: the first column holds the tokens and the other columns hold the vectors. They read files that are already on disk and do not download anything. Use `download_embeddings_model2vec()` to download a Model2Vec model first.

Usage

```
read_embeddings_model2vec(path)

read_embeddings_bpemb(path)
```

Arguments

`path`	For `read_embeddings_model2vec()`, a string giving the path to a folder holding a Model2Vec model, which must contain the files `model.safetensors` and `vocab.txt`. For `read_embeddings_bpemb()`, a string giving the path to an unpacked BPEmb vectors file, which ends in `.w2v.txt`.

Details

The vectors belong to the tokens of one specific tokenizer, and tokens that are not found in the table are dropped without a warning. The text therefore has to be tokenized in the same way the model was trained. See `vignette("static-embeddings", package = "textrecipes")` for the recipes to use with each model, and for where to download the files.

Model2Vec:

`read_embeddings_model2vec()` reads the `embeddings` tensor of `model.safetensors` and uses the lines of `vocab.txt` as the tokens. This needs the safetensors and torch packages. Models that store quantized weights are not supported. The columns of the vectors are named `d1`, `d2`, and so on.

BPEmb:

`read_embeddings_bpemb()` reads the word2vec text format, where the first line gives the number of rows and columns, and each other line is a token followed by its numbers. The columns of the vectors are named `d1`, `d2`, and so on.

Value

A tibble with a character column `token` followed by one numeric column for each dimension of the embeddings.

See Also

`step_word_embeddings()` to use the embeddings in a recipe.

Examples

```
# A small file in the BPEmb format
file <- tempfile(fileext = ".w2v.txt")
writeLines(
  c("3 2", "<unk> 0.1 0.2", "\u2581the 0.3 0.4", "\u2581cat 0.5 0.6"),
  file
)

read_embeddings_bpemb(file)
```

`show_tokens` *Show token output of recipe*

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 recipes::recipe() 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 *Clean Categorical Levels*

Description

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

Usage

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

Arguments

recipe	A 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 recipes::prep.recipe(). Note that if the original variable is a character vector, it will be converted to a factor.
skip	A logical. Should the step be skipped when the recipe is baked by 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 skip = TRUE, 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, orginal, 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 *Clean Variable Names*

Description

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

Usage

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

Arguments

`recipe`	A `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 variable names. This is NULL until computed by recipes::prep.recipe().
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? 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 skip = TRUE, 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 *Indicator Variables via Feature Hashing*

Description

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

Usage

```
step_dummy_hash(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  signed = TRUE,
  num_terms = 32L,
  collapse = FALSE,
  prefix = "dummyhash",
  sparse = "auto",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("dummy_hash")
)
```

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	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 *predictors* 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 NULL until the step is trained by recipes::prep.recipe(), and should not be set by hand.

signed	A logical, indicating whether to use a signed hash-function (generating values of -1, 0, or 1), to reduce collisions when hashing. Defaults to TRUE.
num_terms	An integer, the number of variables to output. Defaults to 32.
collapse	A logical. Should all of the selected columns be collapsed into a single column to create a single set of hashed features? Defaults to FALSE.
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 sparse = "auto" 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 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 skip = TRUE, as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

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

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.
`...`	One or more selector functions to choose which variables are affected by the step. See `recipes::selections()` for more details.
`role`	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`columns`	A character 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.
`lda_models`	A WarpLDA model object from text2vec::LDA. If `NULL`, the default, it will train its model based on the training data. Look at the examples for how to fit a WarpLDA model.
`num_topics`	A whole number, the desired number of latent topics. Defaults to `10`.

prefix — A character string that will be the prefix to the resulting new variables. See notes below.

keep_original_cols — A logical to keep the original variables in the output. Defaults to FALSE.

skip — A logical. Should the step be skipped when the recipe is baked by recipes::bake()? 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 skip = TRUE, as it may affect the computations for subsequent operations.

id — 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` *Lemmatization of Token Variables*

Description

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

Usage

```
step_lemma(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
```

```
  columns = NULL,
  skip = FALSE,
  id = rand_id("lemma")
)
```

Arguments

recipe	A 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.
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 skip = TRUE, as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

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

Value

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

Tidying

When you tidy() this step, a tibble 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_token_replace(), step_tokenfilter(), step_tokenmerge()

Examples

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

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

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

rec_prepped <- prep(rec_spec)

bake(rec_prepped, new_data = NULL)

## End(Not run)
```

`step_ngram` *Generate n-grams From Token Variables*

Description

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

Usage

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

Arguments

`recipe`	A `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.
num_tokens	The number of tokens in the n-gram. This must be an integer greater than or equal to 1. Defaults to 3.
min_num_tokens	The minimum number of tokens in the n-gram. This must be an integer greater than or equal to 1 and smaller than or equal to num_tokens. Defaults to 3.
delim	A single string, the separator between words in an n-gram. Defaults to "_".
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? 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 skip = TRUE, 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_token_replace(), 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 *Part of Speech Filtering of Token Variables*

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

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.
keep_tags	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 remove_tags.
remove_tags	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 keep_tags.
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 skip = TRUE, as it may affect the computations for subsequent operations.
id	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_token_replace(), 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

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	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 *predictors* 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 NULL until the step is trained by recipes::prep.recipe(), and should not be set by hand.
engine	An engine object created by one of the embedding engine functions, such as embedding_custom(). The engine defines how strings are turned into vectors and holds the settings needed to do so.
prefix	A single string used as the prefix of the new columns. See Details for how the column names are built. Defaults to "sentembed".
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? 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 skip = TRUE, as it may affect the computations for subsequent operations.
id	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`, `revision`, `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

revision character, the version of the model the engine is pinned to, `NA` if the engine can't pin 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

`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	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 *predictors* 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 NULL until the step is trained by recipes::prep.recipe(), 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 sequence_length. 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 NULL, 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 FALSE.
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 skip = TRUE, 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 then 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

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.
options	A list of options passed to the stemmer function. Note that the arguments X and FUN should not be passed here. Defaults to list().
custom_stemmer	A custom stemming function. If NULL, the default, SnowballC::wordStem() is used.
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 skip = TRUE, as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

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

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

Value

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

Tidying

When you `tidy()` this step, a tibble 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_token_replace()`, `step_tokenfilter()`, `step_tokenmerge()`

Examples

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

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

tate_obj <- tate_rec |>
  prep()

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

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

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

# Using custom stemmer. Here a custom stemmer that removes the last letter
# if it is a "s".
remove_s <- function(x) gsub("s$", "", x)

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

tate_obj <- tate_rec |>
  prep()

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

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

`step_stopwords` *Filtering of Stop Words for Tokens Variables*

Description

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

Usage

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

Arguments

recipe	A 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.
language	A single string indicating the language of the stop words by ISO 639-1 coding scheme. Ignored if custom_stopword_source is used. Defaults to "en".
keep	A logical. Should the stop words be kept instead of discarded? Defaults to FALSE.
stopword_source	A single string indicating the stop words source, as listed in stopwords::stopwords_getsources(). Ignored if custom_stopword_source is used. Defaults to "snowball".
custom_stopword_source	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 stopword_source and language. If used, language and stopword_source are ignored.
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 skip = TRUE, as it may affect the computations for subsequent operations.
id	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_token_replace()`, `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

`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.
`normalization_form`	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.
`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 `skip = TRUE`, 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\U0308n"))

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

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	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 *predictors* 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 NULL until the step is trained by recipes::prep.recipe(), and should not be set by hand.
extract_functions	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 count_functions. See details for more information.
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? 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 skip = TRUE, 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

`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`	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 *predictors* 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 `NULL` until the step is trained by `recipes::prep.recipe()`, and should not be set by hand.

signed	A logical, indicating whether to use a signed hash-function to reduce collisions when hashing. Defaults to TRUE.
num_terms	An integer, the number of variables to output. Defaults to 1024.
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
sparse	A single string. Should the columns produced be sparse vectors? Can take the values "yes", "no", and "auto". If sparse = "auto" 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 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 skip = TRUE, as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

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

`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`	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 *predictors* 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 `NULL` until the step is trained by `recipes::prep.recipe()`, and should not be set by hand.
`weight_scheme`	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 `sparse = "yes"` has no effect with `"double normalization"`, as it doesn't produce sparse data. Defaults to `"raw count"`.

`weight` A single numeric weight used if `weight_scheme` is set to `"double normalization"`. Defaults to `0.5`.

`vocabulary` A character vector of the tokens to create variables for. If `NULL`, the default, all tokens found in the training data are used.

`res` A named list of the tokens used for each selected variable. This is `NULL` until the step is trained by `recipes::prep.recipe()` and should not be set by hand.

`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 `sparse = "auto"` 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 `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 `skip = TRUE`, 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_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 `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`	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 *predictors* 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 `NULL` until the step is trained by `recipes::prep.recipe()`, and should not be set by hand.

vocabulary
: A character vector of the tokens to create variables for. If NULL, 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 NULL until the step is trained by recipes::prep.recipe() 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 TRUE.

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(TF)? Defaults to FALSE.

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 sparse = "auto" 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 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 skip = TRUE, 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: idf = log(1 + (# documents in the corpus) / (# 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_token_replace` *Replace Tokens in a Token Variable*

Description

`step_token_replace()` creates a *specification* of a recipe step that will replace tokens in a `token` variable according to a table of patterns and replacements.

Usage

```
step_token_replace(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  pattern,
  replacement,
  fixed = TRUE,
  skip = FALSE,
  id = rand_id("token_replace")
)
```

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.
`pattern`	A character vector of tokens to look for. Must be the same length as `replacement`, or `replacement` must be length 1.
`replacement`	A character vector of the tokens to replace the matches with.

fixed	Logical. If TRUE, the default, a token is replaced when it is exactly equal to an element of pattern. If FALSE, pattern is treated as a vector of regular expressions, and a token is replaced when it matches one. Regular expressions are case-sensitive, and backreferences in replacement are not supported: replacement is always used literally.
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 skip = TRUE, as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

The whole token is replaced, not just the part that matched. If a token matches more than one element of pattern, the first match is used. Since the number of tokens is unchanged, the lemma and part-of-speech attributes of the tokens are kept as is.

This is useful for normalizing spelling variants, mapping tokens to a placeholder such as "<number>", or applying a small dictionary of corrections.

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, pattern, replacement, and id:

terms character, the selectors or variables selected

pattern character, the patterns to look for

replacement character, the replacements

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(), step_tokenmerge()

Examples

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

tate_rec <- recipe(~., data = tate_text) |>
  step_tokenize(medium) |>
  step_token_replace(
    medium,
    pattern = c("paper", "canvas"),
    replacement = c("PAPER", "CANVAS")
  )

tate_obj <- tate_rec |>
  prep()

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

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

# Using regular expressions
tate_rec <- recipe(~., data = tate_text) |>
  step_tokenize(medium) |>
  step_token_replace(
    medium,
    pattern = "^[0-9]+$",
    replacement = "<number>",
    fixed = FALSE
  )
```

---

step_tokenfilter *Filter Tokens Based on Term Frequency*

Description

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

Usage

```
step_tokenfilter(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
```

```
  columns = NULL,
  max_times = Inf,
  min_times = 0,
  percentage = FALSE,
  max_tokens = 100,
  filter_fun = NULL,
  res = NULL,
  skip = FALSE,
  id = rand_id("tokenfilter")
)
```

Arguments

recipe	A 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.
max_times	A number. Maximal number of times a token can appear before getting removed. An integer count, or a proportion between 0 and 1 if percentage = TRUE. Defaults to Inf.
min_times	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 percentage = TRUE. Defaults to 0.
percentage	A logical. Should max_times and min_times be interpreted as a proportion of all token occurrences in the training data instead of a count? For example, with percentage = TRUE and min_times = 0.01, tokens that make up less than 1 percent of all tokens are removed. Defaults to FALSE.
max_tokens	An integer. Will only keep the top max_tokens tokens after filtering done by max_times and min_times. Defaults to 100.
filter_fun	A function. This function should take a vector of characters and return a logical vector of the same length. It will be applied to each observation of the data set. If NULL, the default, tokens are filtered using max_times, min_times, percentage, and max_tokens, 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 recipes::prep.recipe() and should not be set by hand.
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 skip = TRUE, 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_token_replace()`, `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` *Tokenization of Character Variables*

Description

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

Usage

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

Arguments

`recipe`	A `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.

training_options — A list of options passed to the tokenizer when it is being trained. Only applicable when engine = "tokenizers.bpe" or engine = "udpipe". For engine = "udpipe" the list must contain model, a loaded udpipe model as returned by udpipe::udpipe_load_model(). Note that the argument x should not be passed here. Defaults to list().

options — A list of options passed to the tokenizer. Note that the argument x should not be passed here. For engine = "tokenizers.bpe" these options only apply when the tokenizer is trained (at prep() time); they have no effect on already-trained models at bake() time. For engine = "spacyr" the options are applied each time the data is tokenized, including at bake() time. Defaults to list().

token — A single string giving the unit for tokenizing. See details for options. Defaults to "words".

engine — A single string giving the package that will be used for tokenization. See details for options. Defaults to "tokenizers".

custom_token — A user supplied tokenizer function. Must take a character vector as input and output a list of character vectors. If NULL, the default, token and engine determine the tokenizer; otherwise they are overwritten.

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

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

Details

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

This tokenizer has arguments that change how the tokenization occurs and can 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)
#> [[1]]
#> [1] "This"  "is"    "words"
#>
#> [[2]]
#> [1] "They" "are"  "nice"
```

We can also stop removing punctuation.

```
recipe(~ text, data = text_tibble) |>
  step_tokenize(text,
                options = list(strip_punct = FALSE,
                               lowercase = FALSE)) |>
  show_tokens(text, n = 2)
#> [[1]]
#> [1] "This"  "is"    "words"
#>
#> [[2]]
#> [1] "They" "are"  "nice" "!"
```

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

```
recipe(~ text, data = text_tibble) |>
  step_tokenize(text, token = "characters") |>
  show_tokens(text, n = 2)
#> [[1]]
#>  [1] "t" "h" "i" "s" "i" "s" "w" "o" "r" "d" "s"
#>
#> [[2]]
#>  [1] "t" "h" "e" "y" "a" "r" "e" "n" "i" "c" "e"
```

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

spacyr:

- "words"

tokenizers.bpe:

The tokeenizers.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`. The 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)

#> [[1]]
#>  [1] "_Th" "is"  "_"   "is"  "_"   "w"   "o"   "r"   "d"   "s"
#>
```

```
#> [[2]]
#>  [1] "_Th" "e"   "y"   "_"   "a"   "r"   "e"   "_"   "n"   "i"   "c"   "e"
#> [13] "!"
```

udpipe:

- "words"

The udpipe engine requires a pre-trained udpipe 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 <- udpipe::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)
#> [[1]]
#> [1] "This"  "is"    "words"
#>
#> [[2]]
#> [1] "They"  "are"   "nice!"
```

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

Tuning Parameters

This step has 1 tuning parameters:

- `token`: Token Unit (type: character, default: words)

Case weights

The underlying operation does not allow for case weights.

See Also

`step_untokenize()` to untokenize.

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

Examples

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

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

tate_obj <- tate_rec |>
  prep()

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

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

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

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

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

step_tokenize_bpe *BPE Tokenization of Character Variables*

Description

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

Usage

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

Arguments

`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.
`options`	A list of options passed to `tokenizers.bpe::bpe()` when the tokenizer is trained at `prep()` time. These options have no effect on already-trained models at `bake()` time. Note that the arguments `x`, `model_path`, and `vocab_size` should not be passed here, use `vocabulary_size` instead. Defaults to `list()`.
`res`	The fitted `tokenizers.bpe::bpe()` model tokenizer. This is `NULL` until the step is trained by `recipes::prep.recipe()` and should not be set by hand.
`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 `skip = TRUE`, 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

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 sentencepiece::sentencepiece() when the tokenizer is trained at prep() time. These options have no effect on already-trained models at bake() time. Note that the arguments x, model_dir, and vocab_size should not be passed here, use vocabulary_size instead. Ignored when model is supplied. Defaults to list().
res	The fitted sentencepiece::sentencepiece() model tokenizer. This is NULL until the step is trained by recipes::prep.recipe() and should not be set by hand.
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 skip = TRUE, 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 NULL 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 skip = TRUE, 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

`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	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 *predictors* 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 NULL until the step is trained by recipes::prep.recipe(), and should not be set by hand.
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? 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 skip = TRUE, as it may affect the computations for subsequent operations.
id	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_token_replace(), 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 *Untokenization of Token Variables*

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 `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.
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 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 skip = TRUE, 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, seperator 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

`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`	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 *predictors* 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 NULL until the step is trained by recipes::prep.recipe(), and should not be set by hand.

embeddings
: A tibble of pre-trained word embeddings, such as those returned by textdata::embedding_glove6b(). The first column should contain tokens, and additional columns should contain embedding vectors.

aggregation
: 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".

aggregation_default
: A single numeric value used for documents where none of the tokens are found in the embeddings. Defaults to 0.

prefix
: A character string that will be the prefix to the resulting new variables. See notes below.

keep_original_cols
: A logical to keep the original variables in the output. Defaults to FALSE.

skip
: A logical. Should the step be skipped when the recipe is baked by 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 skip = TRUE, as it may affect the computations for subsequent operations.

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

Details

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

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

The new components will have names that begin with prefix, then the name of the 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.

Any table of token vectors can be used, not only GloVe. See vignette("static-embeddings", package = "textrecipes") for how to use the Model2Vec and BPEmb embeddings, which need the text to be tokenized in a specific way. read_embeddings_model2vec() and read_embeddings_bpemb() read the downloaded files of these models into the embeddings tibble.

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, 6
embedding_huggingface, 7
embedding_model2vec, 9
embedding_ollama, 11
∗ **Steps for Numeric Variables From Characters**
step_dummy_hash, 19
step_sentence_embeddings, 30
step_sequence_onehot, 33
step_textfeature, 42
∗ **Steps for Numeric Variables From Tokens**
step_lda, 22
step_texthash, 45
step_tf, 48
step_tfidf, 51
step_word_embeddings, 75
∗ **Steps for Text Cleaning**
step_clean_levels, 15
step_clean_names, 17
∗ **Steps for Text Normalization**
step_text_normalization, 41
∗ **Steps for Token Modification**
step_lemma, 24
step_ngram, 26
step_pos_filter, 28
step_stem, 36
step_stopwords, 38
step_token_replace, 54
step_tokenfilter, 56
step_tokenmerge, 71
∗ **Steps for Tokenization**
step_tokenize, 59
step_tokenize_bpe, 65
step_tokenize_sentencepiece, 67
step_tokenize_wordpiece, 69
∗ **Steps for Un-Tokenization**
step_untokenize, 73
∗ **datasets**
emoji_samples, 13

all_tokenized, 3
all_tokenized_predictors *(*all_tokenized*)*, 3

count_functions, 3

download_embeddings_model2vec, 5
download_embeddings_model2vec(), *10*, *13*
dplyr::recode_factor(), *16*

embedding_custom, 6
embedding_custom(), *9*, *10*, *12*, *31*, *32*
embedding_huggingface, 7
embedding_huggingface(), *6*, *10*, *12*
embedding_model2vec, 9
embedding_model2vec(), *6*, *9*, *12*
embedding_ollama, 11
embedding_ollama(), *6*, *9*, *10*
emoji_samples, 13

janitor::make_clean_names(), *16*, *18*

read_embeddings_bpemb *(*read_embeddings_model2vec*)*, 13
read_embeddings_bpemb(), *76*
read_embeddings_model2vec, 13
read_embeddings_model2vec(), *5*, *76*
recipes::bake(), *16*, *18*, *20*, *23*, *25*, *27*, *29*, *31*, *34*, *36*, *39*, *41*, *43*, *46*, *49*, *52*, *55*, *57*, *60*, *65*, *68*, *70*, *72*, *74*, *76*
recipes::has_role(), *3*
recipes::prep(), *16*, *18*, *20*, *21*, *23*, *25*, *27*, *29*, *31*, *34*, *36*, *39*, *41*, *43*, *46*, *47*, *49*, *50*, *52*, *53*, *55*, *57*, *60*, *65*, *68*, *70*, *72*, *74*, *76*
recipes::prep.recipe(), *16*, *18*, *19*, *22*, *25*, *27*, *29*, *31*, *34*, *36*, *39*, *41*, *43*, *45*, *48*, *49*, *51*, *52*, *54*, *57*, *60*, *65*, *67*, *68*, *70*, *72*, *74*, *76*

recipes::recipe(), *15–17*, *19*, *22*, *25, 26*, *29*, *31*, *33*, *36*, *39*, *41*, *43*, *45*, *48*, *51*, *54*, *57*, *59*, *65*, *67*, *70, 71*, *73*, *75*
recipes::selections(), *16, 17*, *19*, *22*, *25*, *27*, *29*, *31*, *34*, *36*, *39*, *41*, *43*, *45*, *48*, *51*, *54*, *57*, *59*, *65*, *67*, *70*, *72, 73*, *75*
recipes::sparse_data, *21*, *47*, *50*, *53*
recipes::step_dummy(), *21*
recipes::step_factor2string(), *16*, *18*
recipes::step_mutate(), *60*
recipes::step_novel(), *16*, *18*
recipes::step_other(), *16*, *18*
recipes::step_regex(), *16*, *18*
recipes::step_string2factor(), *16*, *18*
recipes::step_unknown(), *16*, *18*
rlang::as_function(), *6*

saveRDS(), *6*
sentencepiece::sentencepiece(), *67, 68*
sentencepiece::sentencepiece_download_model(), *67*
show_tokens, 14
SnowballC::wordStem(), *36*
step_clean_levels, 15
step_clean_levels(), *18*
step_clean_names, 17
step_clean_names(), *16*
step_dummy_hash, 19
step_dummy_hash(), *32*, *35*, *44*
step_lda, 22
step_lda(), *47*, *50*, *53*, *77*
step_lemma, 24
step_lemma(), *28*, *30*, *37*, *40*, *55*, *58*, *72*
step_ngram, 26
step_ngram(), *25*, *30*, *37*, *40*, *55*, *58*, *72*
step_pos_filter, 28
step_pos_filter(), *25*, *28*, *37*, *40*, *55*, *58*, *72*
step_sentence_embeddings, 30
step_sentence_embeddings(), *6, 7*, *9–12*, *21*, *35*, *44*
step_sequence_onehot, 33
step_sequence_onehot(), *21*, *32*, *44*
step_stem, 36
step_stem(), *25*, *28*, *30*, *40*, *55*, *58*, *72*
step_stopwords, 38
step_stopwords(), *25*, *28*, *30*, *37*, *55*, *58*, *72*
step_text_normalization, 41
step_text_normalization(), *47*
step_textfeature, 42
step_textfeature(), *21*, *32*, *35*
step_texthash, 45
step_texthash(), *23*, *42*, *50*, *53*, *77*
step_tf, 48, *49*, *58*
step_tf(), *23*, *47*, *53*, *77*
step_tfidf, 51, *52*, *58*
step_tfidf(), *23*, *47*, *50*, *77*
step_token_replace, 54
step_token_replace(), *25*, *28*, *30*, *37*, *40*, *58*, *72*
step_tokenfilter, *49*, *52*, 56
step_tokenfilter(), *25*, *28*, *30*, *37*, *40*, *55*, *72*
step_tokenize, 59
step_tokenize(), *25*, *28*, *30, 31*, *37*, *40*, *47*, *50*, *53*, *55*, *58*, *66*, *68*, *70*, *72*, *74*, *77*
step_tokenize_bpe, 65
step_tokenize_bpe(), *64*, *68*, *70*
step_tokenize_sentencepiece, 67
step_tokenize_sentencepiece(), *64*, *66*, *70*
step_tokenize_wordpiece, 69
step_tokenize_wordpiece(), *64*, *66*, *68*
step_tokenmerge, 71
step_tokenmerge(), *25*, *28*, *30*, *37*, *40*, *55*, *58*
step_untokenize, 73
step_untokenize(), *64*, *66*, *68*, *70*
step_word_embeddings, 75
step_word_embeddings(), *10*, *13, 14*, *23*, *31, 32*, *47*, *50*, *53*
stopwords::stopwords_getsources(), *39*
stringi::stri_trans_nfc(), *41*

text2vec::hash_vectorizer(), *20*, *46*
text2vec::LDA, *22*
textdata::embedding_glove6b(), *76*
tibble, *14*
tidy(), *16*, *18*, *20*, *23*, *25*, *27*, *29*, *32*, *35*, *37*, *39*, *42*, *44*, *46*, *50*, *53*, *55*, *58*, *63*, *66*, *68*, *70*, *72*, *74*, *76*
tidy.step_clean_levels *(*step_clean_levels*)*, 15
tidy.step_clean_names *(*step_clean_names*)*, 17
tidy.step_dummy_hash *(*step_dummy_hash*)*, 19
tidy.step_lda *(*step_lda*)*, 22
tidy.step_lemma *(*step_lemma*)*, 24
tidy.step_ngram *(*step_ngram*)*, 26

tidy.step_pos_filter (step_pos_filter), 28
tidy.step_sentence_embeddings (step_sentence_embeddings), 30
tidy.step_sequence_onehot (step_sequence_onehot), 33
tidy.step_stem (step_stem), 36
tidy.step_stopwords (step_stopwords), 38
tidy.step_text_normalization (step_text_normalization), 41
tidy.step_textfeature (step_textfeature), 42
tidy.step_texthash (step_texthash), 45
tidy.step_tf (step_tf), 48
tidy.step_tfidf (step_tfidf), 51
tidy.step_token_replace (step_token_replace), 54
tidy.step_tokenfilter (step_tokenfilter), 56
tidy.step_tokenize (step_tokenize), 59
tidy.step_tokenize_bpe (step_tokenize_bpe), 65
tidy.step_tokenize_sentencepiece (step_tokenize_sentencepiece), 67
tidy.step_tokenize_wordpiece (step_tokenize_wordpiece), 69
tidy.step_tokenmerge (step_tokenmerge), 71
tidy.step_untokenize (step_untokenize), 73
tidy.step_word_embeddings (step_word_embeddings), 75
token, *3*, *24–26*, *28*, *36*, *38*, *45*, *48*, *51*, *54*, *56*, *59*, *60*, *65*, *67*, *69*, *71*, *73–75*
tokenizers.bpe::bpe(), *65*
tokenizers::tokenize_words, *61*
tokenlist, 78, *78*
tokens, *25*, *28*, *30*, *37*, *40*, *47*, *50*, *53*, *55*, *58*, *72*, *74*, *77*

udpipe::udpipe_download_model(), *63*
udpipe::udpipe_load_model(), *60*, *63*

wordpiece::wordpiece_tokenize(), *10*
wordpiece::wordpiece_vocab(), *70*