Package 'textrecipes'

September 6, 2019

Title Extra 'Recipes' for Text Processing

Version 0.0.2

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

License MIT + file LICENSE

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

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

Depends R (¿= 2.10),
recipes (¿= 0.1.4)

Imports generics,
rlang,
tokenizers,
dplyr,
tibble,
purrr,
SnowballC,
stopwords,
magrittr,
stringr,
text2vec,
textfeatures (¿= 0.3.3),
lifecycle

Suggests covr,
testthat (¿= 2.1.0),
knitr,
rmarkdown

VignetteBuilder knitr

RdMacros lifecycle

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

SystemRequirements GNU make, C++11

R topics documented:

`count_functions` *Counting functions from textfeatures*

Description

Counting functions from textfeatures

`okc_text` *OkCupid Text Data*

Description

These are a sample of columns and users of OkCupid dating website. The data are from Kim and Escobedo-Land (2015). Permission to use this data set was explicitly granted by OkCupid. The data set contains 10 text fields filled out by users.

Value

`okc_text` a tibble

Source

Kim, A. Y., and A. Escobedo-Land. 2015. "OkCupid Data for Introductory Statistics and Data Science Courses." *Journal of Statistics Education: An International Journal on the Teaching and Learning of Statistics*.

Examples

```
data(okc_text)
str(okc_text)
```

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

Description

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

Usage

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

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

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose variables. For 'step_sequence_onehot', this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the 'tidy' method, these are not currently used.
`role`	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
`trained`	A logical to indicate if the recipe has been baked.
`columns`	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
`length`	A numeric, number of characters to keep before discarding. Defaults to 100.
`key`	A character vector, characters to be mapped to integers. characters not in the key will be encoded as 0. Defaults to 'letters'.
`prefix`	A prefix for generated column names, default to "seq1hot".
`skip`	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.
`id`	A character string that is unique to this step to identify it
`x`	A 'step_sequence_onehot' object.

Details

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

Value

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

Source

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

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

juice(okc_obj)

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

---

step_stem *Stemming of list-column variables*

Description

‘step_stem‘ creates a *specification* of a recipe step that will convert a list of tokens into a list of stemmed tokens.

Usage

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

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

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables. For ‘step_stem‘, this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the ‘tidy‘ method, these are not currently used.
role	Not used by this step since no new variables are created.

trained	A logical to indicate if the recipe has been baked.
columns	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
options	A list of options passed to the stemmer function.
custom_stemmer	A custom stemming function. If none is provided it will default to "SnowballC".
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.
x	A 'step_stem' object.

Details

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

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

Value

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

See Also

[step_stopwords()] [step_tokenfilter()] [step_tokenize()]

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

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

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

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

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

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

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

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

`step_stopwords` *Filtering of stopwords from a list-column variable*

Description

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

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"))

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

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose variables. For 'step_stopwords', this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the 'tidy' method, these are not currently used.
`role`	Not used by this step since no new variables are created.
`trained`	A logical to indicate if the recipe has been baked.
`columns`	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
`language`	A character to indicate the langauge of stopwords by ISO 639-1 coding scheme.

`keep` A logical. Specifies whether to keep the stopwords or discard them.

`stopword_source`
A character to indicate the stopwords source as listed in 'stopwords::stopwords_getsources'.

`custom_stopword_source`
A character vector to indicate a custom list of words that cater to the users specific problem.

`skip` A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.

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

`x` A 'step_stopwords' object.

Details

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

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

Value

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

See Also

[step_stem()] [step_tokenfilter()] [step_tokenize()]

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

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

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

tidy(okc_rec, number = 2)
```

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

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

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

`step_textfeature` *Generate the basic set of text features*

Description

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

Usage

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

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

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose variables. For 'step_textfeature', this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the 'tidy' method, these are not currently used.
`role`	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
`trained`	A logical to indicate if the recipe has been baked.
`columns`	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
`extract_functions`	A named list of feature extracting functions. default to [count_functions] from the textfeatures package. See details for more information.
`prefix`	A prefix for generated column names, default to "textfeature".

skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it
x	A 'step_textfeature' object.

Details

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

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

Value

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

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

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

juice(okc_obj) %>%
  pull(textfeature_essay0_n_words)

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

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

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

step_texthash *Term frequency of tokens*

Description

'step_texthash' creates a *specification* of a recipe step that will convert a list of tokens into multiple variables using the hashing trick.

Usage

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

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

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables. For 'step_texthash', this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the 'tidy' method, these are not currently used.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the recipe has been baked.
columns	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
signed	A logical, indicating whether to use a signed hash-function to reduce collisions when hashing. Defaults to TRUE.
num_terms	An integer, the number of variables to output. Defaults to 1024.
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.
x	A 'step_texthash' object.

Details

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

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

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

Value

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

References

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

See Also

[step_tf()] [step_tfidf()] [step_tokenize()]

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

bake(okc_obj, okc_text)

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

---

step_tf *Term frequency of tokens*

Description

‘step_tf‘ creates a *specification* of a recipe step that will convert a list of tokens 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", skip = FALSE,
  id = rand_id("tf"))

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

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables. For 'step_tf', this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the 'tidy' method, these are not currently used.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the recipe has been baked.
columns	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
weight_scheme	A character determining the weighting scheme for the term frequency calculations. Must be one of "binary", "raw count", "term frequency", "log normalization" or "double normalization". Defaults to "raw count".
weight	A numeric weight used if 'weight_scheme' is set to "double normalization". Defaults to 0.5.
vocabulary	A character vector of strings to be considered.
res	The words that will be used to calculate the term frequency will be stored here once this preprocessing step has be trained by [prep.recipe()].
prefix	A character string that will be the prefix to the resulting new variables. See notes below
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.
x	A 'step_tf' object.

Details

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

Term frequency is a weight of how many times each token appear in each observation. There are different ways to calculate the weight and this step can do it in a couple of ways.

Setting the argument ‘weight_scheme‘ to ”binary” will result in a set of binary variables denoting if a token is present in the observation. ”raw count” will count the times a token is present in the observation. ”term frequency” will devide the count with the total number of words in the document to limit the effect of the document length as longer documents tends to have the word present more times but not necessarily at a higher procentage. ”log normalization” takes the log of 1 plus the count, adding 1 is done to avoid taking log of 0. Finally ”double normalization” is the raw frequency divided by the raw frequency of the most occurring term in the document. This is then multiplied by ‘weight‘ and ‘weight‘ is added to the result. This is again done to prevent a bias towards longer documents.

The new components will have names that begin with ‘prefix‘, then the name of the variable, followed by the tokens all seperated by ‘-‘. The new variables will be created alphabetically according to token.

Value

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

See Also

[step_hashing()] [step_tfidf()] [step_tokenize()]

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

bake(okc_obj, okc_text)

tidy(okc_rec, number = 2)
tidy(okc_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 list of tokens 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",
  skip = FALSE, id = rand_id("tfidf"))

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

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables. For 'step_tfidf', this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the 'tidy' method, these are not currently used.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the recipe has been baked.
columns	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
vocabulary	A character vector of strings to be considered.
res	The words that will be used to calculate the term frequency will be stored here once this preprocessing step has be trained by [prep.recipe()].
smooth_idf	TRUE smooth IDF weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. This prevents division by zero.
norm	A character, defines the type of normalization to apply to term vectors. "l1" by default, i.e., scale by the number of words in the document. Must be one of c("l1", "l2", "none").
sublinear_tf	A logical, apply sublinear term-frequency scaling, i.e., replace the term frequency with 1 + log(TF). Defaults to FALSE.
prefix	A character string that will be the prefix to the resulting new variables. See notes below.
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.
x	A 'step_tfidf' object.

Details

It is strongly advised to use [step_tokenfilter] before using [step_tfidf] to limit the number of variables created, otherwise you might run into memmory 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-inverse document frequency is the product of two statistics. The term frequency (TF) and the inverse document frequency (IDF).

Term frequency is a weight of how many times each token appear in each observation.

Inverse document frequency is a measure of how much information a word gives, in other words, how common or rare is the word across all the observations. If a word appears in all the observations it might not give us that much insight, but if it only appear in some it might help us differentiate the observations.

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

The new components will have names that begin with 'prefix', then the name of the variable, followed by the tokens all seperated by '-'. The new variables will be created alphabetically according to token.

Value

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

See Also

[step_hashing()] [step_tf()] [step_tokenize()]

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

bake(okc_obj, okc_text)

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

---

step_tokenfilter *Filter the tokens based on term frequency*

Description

'step_tokenfilter' creates a *specification* of a recipe step that will convert a list of tokens into a list where the tokens are 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, res = NULL, skip = FALSE,
  id = rand_id("tokenfilter"))

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

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables. For 'step_tokenfilter', this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the 'tidy' method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the recipe has been baked.
columns	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
max_times	An integer. Maximal number of times a word can appear before getting removed.
min_times	An integer. Minimum number of times a word can appear before getting removed.
percentage	A logical. Should max_times and min_times be interpreded as a percentage instead of count.
max_tokens	An integer. Will only keep the top max_tokens tokens after filtering done by max_times and min_times. Defaults to 100.
res	The words that will be keep will be stored here once this preprocessing step has be trained by [prep.recipe()].
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.
x	A 'step_tokenfilter' object.

Details

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

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

Value

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

See Also

[step_untokenize()]

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

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

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

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

---

step_tokenize *Tokenization of character variables*

Description

‘step_tokenize‘ creates a *specification* of a recipe step that will convert a character predictor into a list of tokens.

Usage

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

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

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables. For ‘step_tokenize‘, this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the ‘tidy‘ method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the recipe has been baked.
columns	A list of tibble results that define the encoding. This is ‘NULL‘ until the step is trained by [recipes::prep.recipe()].
options	A list of options passed to the tokenizer.
token	Unit for tokenizing. Built-in options from the [tokenizers] package are "words" (default), "characters", "character_shingles", "ngrams", "skip_ngrams", "sentences", "lines", "paragraphs", "regex", "tweets" (tokenization by word that preserves usernames, hashtags, and URLS), "ptb" (Penn Treebank), "skip_ngrams" and "word_stems".
custom_token	User supplied tokenizer. use of this argument will overwrite the token argument. Must take a character vector as input and output a list of character vectors.
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using ‘skip = TRUE‘ as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it
x	A ‘step_tokenize‘ object.

Details

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

Working will ‘textrecipes‘ will always start by calling ‘step_tokenize‘ followed by modifying and filtering steps.

Value

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

See Also

[step_untokenize]

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

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

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

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

okc_obj_chars <- recipe(~ ., data = okc_text) %>%
  step_tokenize(essay0, token = "characters") %>%
  prep(training = okc_text, retain = TRUE)

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

---

step_tokenmerge *Generate the basic set of text features*

Description

'step_tokenmerge' creates a *specification* of a recipe step that will take multiple list-columns of tokens and combine them into one list-column.

Usage

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

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

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables. For 'step_tokenmerge', this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the 'tidy' method, these are not currently used.

role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the recipe has been baked.
columns	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
prefix	A prefix for generated column names, default to "tokenmerge".
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it
x	A 'step_tokenmerge' object.

Value

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

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

juice(okc_obj)

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

---

step_untokenize *Untokenization of list-column variables*

Description

'step_untokenize' creates a *specification* of a recipe step that will convert a list of tokens into a character predictor.

Usage

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

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

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables. For ‘step_untokenize‘, this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the ‘tidy‘ method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the recipe has been baked.
columns	A list of tibble results that define the encoding. This is ‘NULL‘ until the step is trained by [recipes::prep.recipe()].
sep	a character to determine how the tokens should be seperated when pasted together. Defaults to ‘" "‘.
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using ‘skip = TRUE‘ as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.
x	A ‘step_untokenize‘ object.

Details

This steps will turn a tokenized list-column back into a character vector.

Value

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

Examples

```
library(recipes)

data(okc_text)

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

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)
```

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

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

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

---

step_word2vec *Calculates word2vec dimension estimates*

Description

Experimental 'step_word2vec' creates a *specification* of a recipe step that will return the word2vec dimension estimates of a text variable.

Usage

```
step_word2vec(recipe, ..., role = "predictor", trained = FALSE,
  columns = NULL, lda_models = NULL, num_topics = 10,
  prefix = "word2vec", skip = FALSE, id = rand_id("word2vec"))

## S3 method for class 'step_word2vec'
tidy(x, ...)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables. For 'step_word2vec', this indicates the variables to be encoded into a list column. See [recipes::selections()] for more details. For the 'tidy' method, these are not currently used.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the recipe has been baked.
columns	A list of tibble results that define the encoding. This is 'NULL' until the step is trained by [recipes::prep.recipe()].
lda_models	A WarpLDA model object from the text2vec package. If left to NULL, the default, will it train its model based on the training data. Look at the examples for how to fit a WarpLDA model.
num_topics	integer desired number of latent topics.
prefix	A prefix for generated column names, default to "word2vec".
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake.recipe()]? While all operations are baked when [recipes::prep.recipe()] is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using 'skip = TRUE' as it may affect the computations for subsequent operations.

id	A character string that is unique to this step to identify it
x	A 'step_word2vec' object.

Value

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

Source

https://arxiv.org/abs/1301.3781

Examples

```
library(recipes)

data(okc_text)

okc_rec <- recipe(~ ., data = okc_text) %>%
  step_word2vec(essay0)

okc_obj <- okc_rec %>%
  prep(training = okc_text, retain = TRUE)

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

# Changing the number of topics.
recipe(~ ., data = okc_text) %>%
  step_word2vec(essay0, essay1, num_topics = 20) %>%
  prep() %>%
  juice() %>%
  slice(1:2)

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

recipe(~ ., data = okc_text) %>%
  step_word2vec(essay0, essay1, lda_models = lda_model) %>%
  prep() %>%
  juice() %>%
  slice(1:2)
```

Index