Package 'embed'

January 7, 2020

Version 0.0.5

Title Extra Recipes for Encoding Categorical Predictors

Description Predictors can be converted to one or more numeric representations using simple generalized linear models <arXiv:1611.09477> or nonlinear models <arXiv:1604.06737>. All encoding methods are supervised.

Depends R (>= 3.1), recipes (>= 0.1.8)

Imports rstanarm, keras, stats, dplyr, purrr, rlang, utils, generics, tidyr, tibble, lme4, tensorflow, uwot, withr

Suggests testthat, knitr, rmarkdown, covr, ggplot2, modeldata

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2.9000

ByteCompile true

URL `https://tidymodels.github.io/embed`,
`https://github.com/tidymodels/embed`

BugReports `https://github.com/tidymodels/embed/issues`

NeedsCompilation no

Author Max Kuhn [aut, cre],
RStudio [cph]

Maintainer Max Kuhn `<max@rstudio.com>`

Repository CRAN

Date/Publication 2020-01-07 17:20:04 UTC

R topics documented:

`add_woe` *Add WoE in a data frame*

Description

A tidyverse friendly way to plug WoE versions of a set of predictor variables against a given binary outcome.

Usage

```
add_woe(.data, outcome, ..., dictionary = NULL, prefix = "woe")
```

Arguments

`.data`	A tbl. The data.frame to plug the new woe version columns.
`outcome`	The bare name of the outcome variable.
`...`	Bare names of predictor variables, passed as you would pass variables to `dplyr::select()`. This means that you can use all the helpers like `starts_with()` and `matches()`.
`dictionary`	A tbl. If NULL the function will build a dictionary with those variables passed to `...`. You can pass a custom dictionary too, see `dictionary()` for details.
`prefix`	A character string that will be the prefix to the resulting new variables.

Details

You can pass a custom dictionary to `add_woe()`. It must have the exactly the same structure of the output of `dictionary()`. One easy way to do this is to tweak a output returned from it.

Value

A tibble with the original columns of .data plus the woe columns wanted.

Examples

```
mtcars %>% add_woe(am, cyl, gear:carb)
```

dictionary — *Weight of evidence dictionary*

Description

Builds the woe dictionary of a set of predictor variables upon a given binary outcome. Convenient to make a woe version of the given set of predictor variables and also to allow one to tweak some woe values by hand.

Usage

```
dictionary(.data, outcome, ..., Laplace = 1e-06)
```

Arguments

`.data`	A tbl. The data.frame where the variables come from.
`outcome`	The bare name of the outcome variable with exactly 2 distinct values.
`...`	bare names of predictor variables or selectors accepted by `dplyr::select()`.
`Laplace`	Default to 1e-6. The `pseudocount` parameter of the Laplace Smoothing estimator. Value to avoid -Inf/Inf from predictor category with only one outcome class. Set to 0 to allow Inf/-Inf.

Details

You can pass a custom dictionary to `step_woe()`. It must have the exactly the same structure of the output of `dictionary()`. One easy way to do this is by tweaking an output returned from it.

Value

a tibble with summaries and woe for every given predictor variable stacked up.

References

Kullback, S. (1959). *Information Theory and Statistics.* Wiley, New York.

Hastie, T., Tibshirani, R. and Friedman, J. (1986). *Elements of Statistical Learning*, Second Edition, Springer, 2009.

Good, I. J. (1985), "Weight of evidence: A brief survey", *Bayesian Statistics*, 2, pp.249-270.

Examples

```
mtcars %>% dictionary(am, cyl, gear:carb)
```

`step_embed` *Encoding Factors into Multiple Columns*

Description

`step_embed` creates a *specification* of a recipe step that will convert a nominal (i.e. factor) predictor into a set of scores derived from a tensorflow model via a word-embedding model. `embed_control` is a simple wrapper for setting default options.

Usage

```
step_embed(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  outcome = NULL,
  predictors = NULL,
  num_terms = 2,
  hidden_units = 0,
  options = embed_control(),
  mapping = NULL,
  history = NULL,
  skip = FALSE,
  id = rand_id("lencode_bayes")
)

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

embed_control(
  loss = "mse",
  metrics = NULL,
  optimizer = "sgd",
  epochs = 20,
  validation_split = 0,
  batch_size = 32,
  verbose = 0,
  callbacks = NULL
)
```

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_embed`, this indicates the variables to be encoded into a numeric format. 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 embedding variables created will be used as predictors in a model.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`outcome`	A call to `vars` to specify which variable is used as the outcome in the neural network. Only numeric and two-level factors are currently supported.
`predictors`	An optional call to `vars` to specify any variables to be added as additional predictors in the neural network. These variables should be numeric and perhaps centered and scaled.
`num_terms`	An integer for the number of resulting variables.
`hidden_units`	An integer for the number of hidden units in a dense ReLu layer between the embedding and output later. Use a value of zero for no intermediate layer (see Details below).
`options`	A list of options for the model fitting process.
`mapping`	A list of tibble results that define the encoding. This is `NULL` until the step is trained by `recipes::prep.recipe()`.
`history`	A tibble with the convergence statistics for each term. This is `NULL` until the step is trained by `recipes::prep.recipe()`.
`skip`	A logical. Should the step be skipped when the recipe is baked by `recipes::bake.recipe()`? While all operations are baked when `recipes::prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`id`	A character string that is unique to this step to identify it.
`x`	A `step_embed` object.
`optimizer, loss, metrics`	Arguments to pass to `keras::compile()`
`epochs, validation_split, batch_size, verbose, callbacks`	Arguments to pass to `keras::fit()`

Details

Factor levels are initially assigned at random to the new variables and these variables are used in a neural network to optimize both the allocation of levels to new columns as well as estimating a model to predict the outcome. See Section 6.1.2 of Francois and Allaire (2018) for more details.

The new variables are mapped to the specific levels seen at the time of model training and an extra instance of the variables are used for new levels of the factor.

One model is created for each call to `step_embed`. All terms given to the step are estimated and encoded in the same model which would also contain predictors give in `predictors` (if any).

When the outcome is numeric, a linear activation function is used in the last layer while softmax is used for factor outcomes (with any number of levels).

For example, the `keras` code for a numeric outcome, one categorical predictor, and no hidden units used here would be

```
keras_model_sequential() \\%>\\%
layer_embedding(
  input_dim = num_factor_levels_x + 1,
  output_dim = num_terms,
  input_length = 1
) \\%>\\%
layer_flatten() \\%>\\%
layer_dense(units = 1, activation = 'linear')
```

If a factor outcome is used and hidden units were requested, the code would be

```
keras_model_sequential() \\%>\\%
layer_embedding(
  input_dim = num_factor_levels_x + 1,
  output_dim = num_terms,
  input_length = 1
 ) \\%>\\%
layer_flatten() \\%>\\%
layer_dense(units = hidden_units, activation = "relu") \\%>\\%
layer_dense(units = num_factor_levels_y, activation = 'softmax')
```

Other variables specified by `predictors` are added as an additional dense layer after `layer_flatten` and before the hidden layer.

Also note that it may be difficult to obtain reproducible results using this step due to the nature of Tensorflow (see link in References).

tensorflow models cannot be run in parallel within the same session (via `foreach` or `futures`) or the `parallel` package. If using a recipes with this step with `caret`, avoid parallel processing.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the selectors or variables for encoding), `level` (the factor levels), and several columns containing `embed` in the name.

References

Francois C and Allaire JJ (2018) *Deep Learning with R*, Manning

"How can I obtain reproducible results using Keras during development?" https://tinyurl.com/keras-repro

"Concatenate Embeddings for Categorical Variables with Keras" https://flovv.github.io/Embeddings_with_keras_part2/

Examples

```
library(modeldata)
data(okc)

rec <- recipe(Class ~ age + location, data = okc) %>%
  step_embed(location, outcome = vars(Class),
             options = embed_control(epochs = 10))

# See https://tidymodels.github.io/embed/ for examples
```

---

step_lencode_bayes — *Supervised Factor Conversions into Linear Functions using Bayesian Likelihood Encodings*

Description

`step_lencode_bayes` creates a *specification* of a recipe step that will convert a nominal (i.e. factor) predictor into a single set of scores derived from a generalized linear model estimated using Bayesian analysis.

Usage

```
step_lencode_bayes(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  options = list(seed = sample.int(10^5, 1)),
  verbose = FALSE,
  mapping = NULL,
  skip = FALSE,
  id = rand_id("lencode_bayes")
)

## S3 method for class 'step_lencode_bayes'
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_lencode_bayes`, this indicates the variables to be encoded into a numeric format. 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 quantities for preprocessing have been estimated.
`outcome`	A call to `vars` to specify which variable is used as the outcome in the generalized linear model. Only numeric and two-level factors are currently supported.
`options`	A list of options to pass to `rstanarm::stan_glmer()`.
`verbose`	A logical to control the default printing by `rstanarm::stan_glmer()`.
`mapping`	A list of tibble results that define the encoding. This is `NULL` until the step is trained by `recipes::prep.recipe()`.
`skip`	A logical. Should the step be skipped when the recipe is baked by `recipes::bake.recipe()`? While all operations are baked when `recipes::prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations
`id`	A character string that is unique to this step to identify it.
`x`	A `step_lencode_bayes` object.

Details

For each factor predictor, a generalized linear model is fit to the outcome and the coefficients are returned as the encoding. These coefficients are on the linear predictor scale so, for factor outcomes, they are in log-odds units. The coefficients are created using a no intercept model and, when two factor outcomes are used, the log-odds reflect the event of interest being the *first* level of the factor.

For novel levels, a slightly timmed average of the coefficients is returned.

A hierarchical generalized linear model is fit using `rstanarm::stan_glmer()` and no intercept via

```
  stan_glmer(outcome ~ (1 | predictor), data = data, ...)
```

where the `...` include the `family` argument (automatically set by the step) as well as any arguments given to the `options` argument to the step. Relevant options include `chains`, `iter`, `cores`, and arguments for the priors (see the links in the References below). `prior_intercept` is the argument that has the most effect on the amount of shrinkage.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the selectors or variables for encoding), `level` (the factor levels), and `value` (the encodings).

References

Micci-Barreca D (2001) "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," ACM SIGKDD Explorations Newsletter, 3(1), 27-32.

Zumel N and Mount J (2017) "vtreat: a data.frame Processor for Predictive Modeling," arXiv:1611.09477

"Hierarchical Partial Pooling for Repeated Binary Trials" https://tinyurl.com/stan-pooling

"Prior Distributions for ‘rstanarm“ Models" https://tinyurl.com/stan-priors

"Estimating Generalized (Non-)Linear Models with Group-Specific Terms with `rstanarm`" https://tinyurl.com/stan-glm-grouped

Examples

```
library(recipes)
library(dplyr)
library(modeldata)

data(okc)

reencoded <- recipe(Class ~ age + location, data = okc) %>%
  step_lencode_bayes(location, outcome = vars(Class))

# See https://tidymodels.github.io/embed/ for examples
```

---

step_lencode_glm — *Supervised Factor Conversions into Linear Functions using Likelihood Encodings*

Description

step_lencode_glm creates a *specification* of a recipe step that will convert a nominal (i.e. factor) predictor into a single set of scores derived from a generalized linear model.

Usage

```
step_lencode_glm(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  mapping = NULL,
  skip = FALSE,
  id = rand_id("lencode_bayes")
)

## S3 method for class 'step_lencode_glm'
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_lencode_glm, this indicates the variables to be encoded into a numeric format. 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 quantities for preprocessing have been estimated.

outcome	A call to vars to specify which variable is used as the outcome in the generalized linear model. Only numeric and two-level factors are currently supported.
mapping	A list of tibble results that define the encoding. This is NULL until the step is trained by recipes::prep.recipe().
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake.recipe()? While all operations are baked when recipes::prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations
id	A character string that is unique to this step to identify it.
x	A step_lencode_glm object.

Details

For each factor predictor, a generalized linear model is fit to the outcome and the coefficients are returned as the encoding. These coefficients are on the linear predictor scale so, for factor outcomes, they are in log-odds units. The coefficients are created using a no intercept model and, when two factor outcomes are used, the log-odds reflect the event of interest being the *first* level of the factor.

For novel levels, a slightly timmed average of the coefficients is returned.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables for encoding), level (the factor levels), and value (the encodings).

References

Micci-Barreca D (2001) "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," ACM SIGKDD Explorations Newsletter, 3(1), 27-32.

Zumel N and Mount J (2017) "vtreat: a data.frame Processor for Predictive Modeling," arXiv:1611.09477

Examples

```
library(recipes)
library(dplyr)
library(modeldata)

data(okc)

glm_est <- recipe(Class ~ age + location, data = okc) %>%
  step_lencode_glm(location, outcome = vars(Class))

# See https://tidymodels.github.io/embed/ for examples
```

`step_lencode_mixed` *Supervised Factor Conversions into Linear Functions using Bayesian Likelihood Encodings*

Description

`step_lencode_mixed` creates a *specification* of a recipe step that will convert a nominal (i.e. factor) predictor into a single set of scores derived from a generalized linear mixed model.

Usage

```
step_lencode_mixed(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  options = list(verbose = 0),
  mapping = NULL,
  skip = FALSE,
  id = rand_id("lencode_bayes")
)

## S3 method for class 'step_lencode_mixed'
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_lencode_mixed`, this indicates the variables to be encoded into a numeric format. 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 quantities for preprocessing have been estimated.
`outcome`	A call to `vars` to specify which variable is used as the outcome in the generalized linear model. Only numeric and two-level factors are currently supported.
`options`	A list of options to pass to `lme4::lmer()` or `lme4::glmer()`.
`mapping`	A list of tibble results that define the encoding. This is `NULL` until the step is trained by `recipes::prep.recipe()`.
`skip`	A logical. Should the step be skipped when the recipe is baked by `recipes::bake.recipe()`? While all operations are baked when `recipes::prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations

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

Details

For each factor predictor, a generalized linear model is fit to the outcome and the coefficients are returned as the encoding. These coefficients are on the linear predictor scale so, for factor outcomes, they are in log-odds units. The coefficients are created using a no intercept model and, when two factor outcomes are used, the log-odds reflect the event of interest being the *first* level of the factor.

For novel levels, a slightly timmed average of the coefficients is returned.

A hierarchical generalized linear model is fit using `lme4::lmer()` or `lme4::glmer()`, depending on the nature of the outcome, and no intercept via

```
  lmer(outcome ~ 1 + (1 | predictor), data = data, ...)
```

where the `...` include the `family` argument (automatically set by the step) as well as any arguments given to the `options` argument to the step. Relevant options include `control` and others.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the selectors or variables for encoding), `level` (the factor levels), and `value` (the encodings).

References

Micci-Barreca D (2001) "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," ACM SIGKDD Explorations Newsletter, 3(1), 27-32.

Zumel N and Mount J (2017) "vtreat: a data.frame Processor for Predictive Modeling," arXiv:1611.09477

Examples

```
library(recipes)
library(dplyr)
library(modeldata)

data(okc)

reencoded <- recipe(Class ~ age + location, data = okc) %>%
  step_lencode_mixed(location, outcome = vars(Class))

# See https://tidymodels.github.io/embed/ for examples
```

step_umap — *Supervised and unsupervised uniform manifold approximation and projection (UMAP)*

Description

`step_umap` creates a *specification* of a recipe step that will project a set of features into a smaller space.

Usage

```
step_umap(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  outcome = NULL,
  neighbors = 15,
  num_comp = 2,
  min_dist = 0.01,
  learn_rate = 1,
  epochs = NULL,
  options = list(verbose = FALSE, n_threads = 1),
  seed = sample(10^5, 2),
  retain = FALSE,
  object = NULL,
  skip = FALSE,
  id = rand_id("umap")
)

## S3 method for class 'step_umap'
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_umap`, this indicates the variables to be encoded into a numeric format. Numeric and factor variables can be used. 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 embedding columns created by the original variables will be used as predictors in a model.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.

`outcome`	A call to `vars` to specify which variable is used as the outcome in the encoding process (if any).
`neighbors`	An integer for the number of nearest neighbors used to construct the target simplicial set.
`num_comp`	An integer for the number of UMAP components.
`min_dist`	The effective minimum distance between embedded points.
`learn_rate`	Positive number of the learning rate for the optimization process.
`epochs`	Number of iterations for the neighbor optimization. See `uwot::umap()` for mroe details.
`options`	A list of options to pass to `uwot::umap()`. The arguments `X`, `n_neighbors`, `n_components`, `min_dist`, `n_epochs`, `ret_model`, and `learning_rate` should not be passed here. By default, `verbose` and `n_threads` are set.
`seed`	Two integers to control the random numbers used by the numerical methods. The default pulls from the main session's stream of numbers and will give reproducible results if the seed is set prior to calling `prep.recipe()` or `bake.recipe()`.
`retain`	A single logical for whether the original predictors should be kept (in addition to the new embedding variables).
`object`	An object that defines the encoding. This is `NULL` until the step is trained by `recipes::prep.recipe()`.
`skip`	A logical. Should the step be skipped when the recipe is baked by `recipes::bake.recipe()`? While all operations are baked when `recipes::prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations
`id`	A character string that is unique to this step to identify it.
`x`	A `step_umap` object.

Details

UMAP, short for Uniform Manifold Approximation and Projection, is a nonlinear dimension reduction technique that finds local, low-dimensional representations of the data. It can be run unsupervised or supervised with different types of outcome data (e.g. numeric, factor, etc).

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with a column called `terms` (the selectors or variables for embedding) is returned.

References

McInnes, L., & Healy, J. (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. `https://arxiv.org/abs/1802.03426`.

"How UMAP Works" `https://umap-learn.readthedocs.io/en/latest/how_umap_works.html`

Examples

```
library(recipes)
library(dplyr)
library(ggplot2)

split <- seq.int(1, 150, by = 9)
tr <- iris[-split, ]
te <- iris[ split, ]

set.seed(11)
supervised <-
  recipe(Species ~ ., data = tr) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_umap(all_predictors(), outcome = vars(Species), num_comp = 2) %>%
  prep(training = tr)

theme_set(theme_bw())

bake(supervised, new_data = te, Species, starts_with("umap")) %>%
  ggplot(aes(x = umap_1, y = umap_2, col = Species)) +
  geom_point(alpha = .5)
```

---

step_woe *Weight of evidence transformation*

Description

`step_woe` creates a *specification* of a recipe step that will transform nominal data into its numerical transformation based on weights of evidence against a binary outcome.

Usage

```
step_woe(
  recipe,
  ...,
  role = "predictor",
  outcome,
  trained = FALSE,
  dictionary = NULL,
  Laplace = 1e-06,
  prefix = "woe",
  skip = FALSE,
  id = rand_id("woe")
)

## S3 method for class 'step_woe'
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 which variables will be used to compute the components. See `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 woe components columns created by the original variables will be used as predictors in a model.
`outcome`	The bare name of the binary outcome.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`dictionary`	A tbl. A map of levels and woe values. It must have the same layout than the output returned from `dictionary()`. If 'NULL" the function will build a dictionary with those variables passed to `...`. See `dictionary()` for details.
`Laplace`	The Laplace smoothing parameter. A value usually applied to avoid -Inf/Inf from predictor category with only one outcome class. Set to 0 to allow Inf/-Inf. The default is 1e-6. Also kwon as 'pseudocount' parameter of the Laplace smoothing technique.
`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_woe` object.

Details

WoE is a transformation of a group of variables that produces a new set of features. The formula is

$$woe_c = log((P(X = c|Y = 1))/(P(X = c|Y = 0)))$$

where c goes from 1 to C levels of a given nominal predictor variable X.

These components are designed to transform nominal variables into numerical ones with the property that the order and magnitude reflects the association with a binary outcome. To apply it on numerical predictors, it is advisable to discretize the variables prior to running WoE. Here, each variable will be binarized to have woe associated later. This can achieved by using `step_discretize()`.

The argument `Laplace` is an small quantity added to the proportions of 1's and 0's with the goal to avoid log(p/0) or log(0/p) results. The numerical woe versions will have names that begin with `woe_` followed by the respecttive original name of the variables. See Good (1985).

One can pass a custom `dictionary` tibble to `step_woe()`. It must have the same structure of the output from `dictionary()` (see examples). If not provided it will be created automatically. The role of this tibble is to store the map between the levels of nominal predictor to its woe values. You may want to tweak this object with the goal to fix the orders between the levels of one given predictor. One easy way to do this is by tweaking an output returned from `dictionary()`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with the woe dictionary used to map categories with woe values.

References

Kullback, S. (1959). *Information Theory and Statistics.* Wiley, New York.

Hastie, T., Tibshirani, R. and Friedman, J. (1986). *Elements of Statistical Learning*, Second Edition, Springer, 2009.

Good, I. J. (1985), "Weight of evidence: A brief survey", *Bayesian Statistics*, 2, pp.249-270.

Examples

```
library(modeldata)
data("credit_data")

set.seed(111)
in_training <- sample(1:nrow(credit_data), 2000)

credit_tr <- credit_data[ in_training, ]
credit_te <- credit_data[-in_training, ]

rec <- recipe(Status ~ ., data = credit_tr) %>%
  step_woe(Job, Home, outcome = Status)

woe_models <- prep(rec, training = credit_tr)

# the encoding:
bake(woe_models, new_data = credit_te %>% slice(1:5), starts_with("woe"))
# the original data
credit_te %>% slice(1:5) %>% dplyr::select(Job, Home)
# the details:
tidy(woe_models, number = 1)

# Example of custom dictionary + tweaking
# custom dictionary
woe_dict_custom <- credit_tr %>% dictionary(Job, Home, outcome = Status)
woe_dict_custom[4, "woe"] <- 1.23 #tweak

#passing custom dict to step_woe()
rec_custom <- recipe(Status ~ ., data = credit_tr) %>%
  step_woe(Job, Home, outcome = Status, dictionary = woe_dict_custom) %>%
  prep

rec_custom_baked <- bake(rec_custom, new_data = credit_te)
rec_custom_baked %>% dplyr::filter(woe_Job == 1.23) %>% head
```

woe_table — *Crosstable with woe between a binary outcome and a predictor variable.*

Description

Calculates some summaries and the WoE (Weight of Evidence) between a binary outcome and a given predictor variable. Used to biuld the dictionary.

Usage

```
woe_table(predictor, outcome, Laplace = 1e-06)
```

Arguments

`predictor`	A atomic vector, usualy with few distinct values.
`outcome`	The dependent variable. A atomic vector with exactly 2 distinct values.
`Laplace`	The `pseudocount` parameter of the Laplace Smoothing estimator. Default to 1e-6. Value to avoid -Inf/Inf from predictor category with only one outcome class. Set to 0 to allow Inf/-Inf.

Value

a tibble with counts, proportions and woe. Warning: woe can possibly be -Inf. Use 'Laplace' arg to avoid that.

References

Kullback, S. (1959). *Information Theory and Statistics.* Wiley, New York.

Hastie, T., Tibshirani, R. and Friedman, J. (1986). *Elements of Statistical Learning*, Second Edition, Springer, 2009.

Good, I. J. (1985), "Weight of evidence: A brief survey", *Bayesian Statistics*, 2, pp.249-270.

Index