Package 'embed'

January 29, 2026

Title Extra Recipes for Encoding Predictors

Version 1.2.2

Description Predictors can be converted to one or more numeric
representations using a variety of methods. Effect encodings using
simple generalized linear models <doi:10.48550/arXiv.1611.09477> or
nonlinear models <doi:10.48550/arXiv.1604.06737> can be used. There
are also functions for dimension reduction and other approaches.

License MIT + file LICENSE

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

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

Depends R (>= 4.1), recipes (>= 1.1.0)

Imports cli, dplyr (>= 1.1.0), generics (>= 0.1.0), glue, lifecycle,
purrr, rlang (>= 1.1.0), rsample, stats, tibble, tidyr, utils,
uwot, vctrs, withr

Suggests covr, dials (>= 1.2.0), ggplot2, hardhat, irlba, keras3,
knitr, lme4, modeldata, rmarkdown, rpart, rstanarm, stringdist,
tensorflow, testthat (>= 3.0.0), VBsparsePCA, xgboost

ByteCompile true

Config/Needs/website tidymodels, ggiraph, tidyverse/tidytemplate,
reticulate

Config/testthat/edition 3

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

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Emil Hvitfeldt [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0679-1945>),
Max Kuhn [aut] (ORCID: <https://orcid.org/0000-0003-2402-136X>),
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

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

Contents

---

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)
```

---

solubility	*Compound solubility data*

Description

Compound solubility data

Details

Tetko et al. (2001) and Huuskonen (2000) investigated a set of compounds with corresponding experimental solubility values using complex sets of descriptors. They used linear regression and neural network models to estimate the relationship between chemical structure and solubility. For our analyses, we will use 1267 compounds and a set of more understandable descriptors that fall into one of three groups: 208 binary "fingerprints" that indicate the presence or absence of a particular chemical sub-structure, 16 count descriptors (such as the number of bonds or the number of Bromine atoms) and 4 continuous descriptors (such as molecular weight or surface area).

Value

solubility   a data frame

Source

Tetko, I., Tanchuk, V., Kasheva, T., and Villa, A. (2001). Estimation of aqueous solubility of chemical compounds using E-state indices. *Journal of Chemical Information and Computer Sciences*, 41(6), 1488-1493.

Huuskonen, J. (2000). Estimation of aqueous solubility for a diverse set of organic compounds based on molecular topology. *Journal of Chemical Information and Computer Sciences*, 40(3), 773-777.

Examples

```
data(solubility)
str(solubility)
```

---

step_collapse_cart   *Supervised Collapsing of Factor Levels*

Description

step_collapse_cart() creates a *specification* of a recipe step that can collapse factor levels into a smaller set using a supervised tree.

Usage

```
step_collapse_cart(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  cost_complexity = 1e-04,
  min_n = 5,
  results = NULL,
  skip = FALSE,
  id = rand_id("step_collapse_cart")
)
```

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 are affected by the step. 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 to train CART models in order to pool factor levels.
cost_complexity	A non-negative value that regulates the complexity of the tree when pruning occurs. Values near 0.1 usually correspond to a tree with a single splits. Values of zero correspond to unpruned tree.
min_n	An integer for how many data points are required to make further splits during the tree growing process. Larger values correspond to less complex trees.
results	A list of results to convert to new factor levels.
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 uses a CART tree (classification or regression) to group the existing factor levels into a
potentially smaller set. It changes the levels in the factor predictor (and the tidy() method can be
used to understand the translation).

There are a few different ways that the step will not be able to collapse levels. If the model fails
or, if the results have each level being in its own split, the original factor levels are retained. There
are also cases where there is "no admissible split" which means that the model could not find any
signal in the data.

Value

An updated recipe step.

Tidying

When you [tidy()](#) this step, a tibble is returned with columns terms, old, new, and id:

terms character, the selectors or variables selected

old character, the old levels

new character, the new levels

id character, id of this step

Case weights

The underlying operation does not allow for case weights.

See Also

[step_collapse_stringdist()](#)

Examples

```
data(ames, package = "modeldata")
ames$Sale_Price <- log10(ames$Sale_Price)

rec <-
  recipe(Sale_Price ~ ., data = ames) |>
  step_collapse_cart(
    Sale_Type, Garage_Type, Neighborhood,
    outcome = vars(Sale_Price)
  ) |>
  prep()
tidy(rec, number = 1)
```

```
step_collapse_stringdist
```
collapse factor levels using stringdist

Description

step_collapse_stringdist() creates a *specification* of a recipe step that will collapse factor levels that have a low stringdist between them.

Usage

```
step_collapse_stringdist(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  distance = NULL,
  method = "osa",
  options = list(),
  results = NULL,
  columns = NULL,
  skip = FALSE,
  id = rand_id("collapse_stringdist")
)
```

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 are affected by the step. 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.
distance	Integer, value to determine which strings should be collapsed with which. The value is being used inclusive, so 2 will collapse levels that have a string distance between them of 2 or lower.
method	Character, method for distance calculation. The default is "osa", see stringdist::stringdist-metrics.
options	List, other arguments passed to stringdist::stringdistmatrix() such as weight, q, p, and bt, that are used for different values of method.
results	A list denoting the way the labels should be collapses is stored here once this preprocessing step has be trained by recipes::prep.
columns	A character string of variable names that will be populated (eventually) by the terms argument.

| skip | A logical. Should the step be skipped when the recipe is baked by [bake()](#)? While all operations are baked when [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). For the `tidy` method, a tibble with columns `terms` (the columns that will be affected) and `base`.

Tidying

When you [tidy()](#) this step, a tibble is returned with columns `terms`, `from`, `to`, and `id`:

terms character, the selectors or variables selected

from character, the old levels

too character, the new levels

id character, id of this step

Case weights

The underlying operation does not allow for case weights.

See Also

[step_collapse_cart()](#)

Examples

```
library(recipes)
library(tibble)
data0 <- tibble(
  x1 = c("a", "b", "d", "e", "sfgsfgsd", "hjhgfgjgr"),
  x2 = c("ak", "b", "djj", "e", "hjhgfgjgr", "hjhgfgjgr")
)

rec <- recipe(~., data = data0) |>
  step_collapse_stringdist(all_predictors(), distance = 1) |>
  prep()

rec |>
  bake(new_data = NULL)

tidy(rec, 1)

rec <- recipe(~., data = data0) |>
  step_collapse_stringdist(all_predictors(), distance = 2) |>
  prep()
```

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

tidy(rec, 1)
```

---

step_discretize_cart   *Discretize numeric variables with CART*

Description

`step_discretize_cart()` creates a *specification* of a recipe step that will discretize numeric data (e.g. integers or doubles) into bins in a supervised way using a CART model.

Usage

```
step_discretize_cart(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  cost_complexity = 0.01,
  tree_depth = 10,
  min_n = 20,
  rules = NULL,
  skip = FALSE,
  id = rand_id("discretize_cart")
)
```

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 are affected by the step. See [recipes::selections](#) for more details.
role	Defaults to `"predictor"`.
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 to train CART models in order to discretize explanatory variables.
cost_complexity	
	The regularization parameter. Any split that does not decrease the overall lack of fit by a factor of `cost_complexity` is not attempted. Corresponds to `cp` in [rpart::rpart()](#). Defaults to 0.01.

tree_depth	The *maximum* depth in the final tree. Corresponds to maxdepth in [rpart::rpart()](). Defaults to 10.
min_n	The number of data points in a node required to continue splitting. Corresponds to minsplit in [rpart::rpart()](). Defaults to 20.
rules	The splitting rules of the best CART tree to retain for each variable. If length zero, splitting could not be used on that column.
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

step_discretize_cart() creates non-uniform bins from numerical variables by utilizing the information about the outcome variable and applying a CART model.

The best selection of buckets for each variable is selected using the standard cost-complexity pruning of CART, which makes this discretization method resistant to overfitting.

This step requires the **rpart** package. If not installed, the step will stop with a note about installing the package.

Note that the original data will be replaced with the new bins.

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

Tidying

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

terms character, the selectors or variables selected

value numeric, location of the splits

id character, id of this step

Tuning Parameters

This step has 3 tuning parameters:

- cost_complexity: Cost-Complexity Parameter (type: double, default: 0.01)
- tree_depth: Tree Depth (type: integer, default: 10)
- min_n: Minimal Node Size (type: integer, default: 20)

Case weights

This step performs an supervised operation that can utilize case weights. To use them, see the documentation in [recipes::case_weights]() and the examples on tidymodels.org.

See Also

recipes::step_discretize(), step_discretize_xgb()

Examples

```
library(modeldata)
data(ad_data)
library(rsample)

split <- initial_split(ad_data, strata = "Class")

ad_data_tr <- training(split)
ad_data_te <- testing(split)

cart_rec <-
  recipe(Class ~ ., data = ad_data_tr) |>
  step_discretize_cart(
    tau, age, p_tau, Ab_42,
    outcome = "Class", id = "cart splits"
  )

cart_rec <- prep(cart_rec, training = ad_data_tr)

# The splits:
tidy(cart_rec, id = "cart splits")

bake(cart_rec, ad_data_te, tau)
```

---

step_discretize_xgb   *Discretize numeric variables with XgBoost*

Description

step_discretize_xgb() creates a *specification* of a recipe step that will discretize numeric data (e.g. integers or doubles) into bins in a supervised way using an XgBoost model.

Usage

```
step_discretize_xgb(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  sample_val = 0.2,
  learn_rate = 0.3,
  num_breaks = 10,
  tree_depth = 1,
```

```
  min_n = 5,
  rules = NULL,
  skip = FALSE,
  id = rand_id("discretize_xgb")
)
```

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 are affected by the step. See recipes::selections for more details.
role	Defaults to "predictor".
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 to train XgBoost models in order to discretize explanatory variables.
sample_val	Share of data used for validation (with early stopping) of the learned splits (the rest is used for training). Defaults to 0.20.
learn_rate	The rate at which the boosting algorithm adapts from iteration-to-iteration. Corresponds to eta in the **xgboost** package. Defaults to 0.3.
num_breaks	The *maximum* number of discrete bins to bucket continuous features. Corresponds to max_bin in the **xgboost** package. Defaults to 10.
tree_depth	The maximum depth of the tree (i.e. number of splits). Corresponds to max_depth in the **xgboost** package. Defaults to 1.
min_n	The minimum number of instances needed to be in each node. Corresponds to min_child_weight in the **xgboost** package. Defaults to 5.
rules	The splitting rules of the best XgBoost tree to retain for each variable.
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

step_discretize_xgb() creates non-uniform bins from numerical variables by utilizing the information about the outcome variable and applying the xgboost model. It is advised to impute missing values before this step. This step is intended to be used particularly with linear models because thanks to creating non-uniform bins it becomes easier to learn non-linear patterns from the data.

The best selection of buckets for each variable is selected using an internal early stopping scheme implemented in the **xgboost** package, which makes this discretization method prone to overfitting.

The pre-defined values of the underlying xgboost learns good and reasonably complex results. However, if one wishes to tune them the recommended path would be to first start with changing the value of num_breaks to e.g.: 20 or 30. If that doesn't give satisfactory results one could experiment

with modifying the `tree_depth` or `min_n` parameters. Note that it is not recommended to tune `learn_rate` simultaneously with other parameters.

This step requires the **xgboost** package. If not installed, the step will stop with a note about installing the package.

Note that the original data will be replaced with the new bins.

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Tidying

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

terms character, the selectors or variables selected

value numeric, location of the splits

id character, id of this step

Tuning Parameters

This step has 5 tuning parameters:

- `sample_val`: Proportion of data for validation (type: double, default: 0.2)
- `learn_rate`: Learning Rate (type: double, default: 0.3)
- `num_breaks`: Number of Cut Points (type: integer, default: 10)
- `tree_depth`: Tree Depth (type: integer, default: 1)
- `min_n`: Minimal Node Size (type: integer, default: 5)

Case weights

This step performs an supervised operation that can utilize case weights. To use them, see the documentation in [recipes::case_weights](#) and the examples on tidymodels.org.

See Also

[`recipes::step_discretize()`](#), [`step_discretize_cart()`](#)

Examples

```
library(rsample)
library(recipes)
data(credit_data, package = "modeldata")

set.seed(1234)
split <- initial_split(credit_data[1:1000, ], strata = "Status")

credit_data_tr <- training(split)
credit_data_te <- testing(split)
```

```
xgb_rec <-
  recipe(Status ~ Income + Assets, data = credit_data_tr) |>
  step_impute_median(Income, Assets) |>
  step_discretize_xgb(Income, Assets, outcome = "Status")

xgb_rec <- prep(xgb_rec, training = credit_data_tr)

bake(xgb_rec, credit_data_te, Assets)
```

---

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,
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("embed")
)

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.
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().
history	A tibble with the convergence statistics for each term. This is NULL until the step is trained by recipes::prep().
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.
optimizer, loss, metrics	Arguments to pass to keras3::compile()
epochs, validation_split, batch_size, verbose, callbacks	Arguments to pass to keras3::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 keras3 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
) |>
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
) |>
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.

Tidying

When you [tidy()](#) this step, a tibble is returned with a number of columns with embedding information, and columns terms, levels, and id:

terms character, the selectors or variables selected

levels character, levels in variable

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `num_terms`: # Model Terms (type: integer, default: 2)

- `hidden_units`: # Hidden Units (type: integer, default: 0)

Case weights

The underlying operation does not allow for case weights.

References

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

"Concatenate Embeddings for Categorical Variables with keras3" https://flovv.github.io/Embeddings_with_keras_part2/

Examples

```
data(grants, package = "modeldata")

set.seed(1)
grants_other <- sample_n(grants_other, 500)

rec <- recipe(class ~ num_ci + sponsor_code, data = grants_other) |>
  step_embed(sponsor_code,
    outcome = vars(class),
    options = embed_control(epochs = 10)
  )
```

---

step_lencode   *Likelihood encoding using analytical formula*

Description

`step_lencode()` creates a *specification* of a recipe step that will convert a nominal (i.e. factor) predictor into a single set of scores derived analytically.

Usage

```
step_lencode(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  smooth = TRUE,
  mapping = NULL,
```

```
    skip = FALSE,
    id = rand_id("lencode")
)
```

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(), 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. Only numeric and two-level factors are currently supported.
smooth	A logical, default to TRUE, should the estimates of groups with low counts be pulled towards the gobal estimate? Defaults to TRUE. See Details for how this is done. This is also known as partial pooling or shrinkage. Only works for numeric outcomes.
mapping	A list of tibble results that define the encoding. This is NULL until the step is trained by recipes::prep().
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

Each selected nominal predictor will be replaced by a numeric predictor. Each unique value of the nominal predictor is replaced by a numeric value. Thse values are calculated differently depending on the type of the outcome.

For **numeric** outcomes each value is the average value of the outcome inside each of the levels of the predictor. Unseen levels of the predictor will be using the global mean of the predictor. If case weights are used then a weighted mean is calculated instead.

For **nominal** outcomes each value is the log odds of the of the first level of the outcome variable being present, within each level of the levels of the predictor. Unseen levels will be replaced by the global log odds without stratification. If case weights are used then a weighted log odds is calculated.

If no or all occurances happens then the log odds is calculated using p = (2 * nrow(data) - 1) / (2 * nrow(data)) to avoid infinity that would happen by taking the log of 0.

For numeric outcomes where smooth = TRUE, the following adjustment is done.

$$estimate = (n/global_{var})/(n/global_{var}+1/outcome_{var})*estimate+(1/outcome_{var})/(n/global_{var}+1/outcome_{var})*g$$

Where n is the number of observations in the group.

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).

Tidying

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

level character, the factor levels

value numeric, the encoding

terms character, the selectors or variables selected

id character, id of this step

Case weights

This step performs an supervised operation that can utilize case weights. To use them, see the documentation in recipes::case_weights and the examples on tidymodels.org.

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

See Also

step_lencode_bayes(), step_lencode_glm(), step_lencode_mixed()

Examples

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

data(grants)

set.seed(1)
grants_other <- sample_n(grants_other, 500)
reencoded <- recipe(class ~ sponsor_code, data = grants_other) |>
  step_lencode(sponsor_code, outcome = vars(class), smooth = FALSE) |>
  prep()

bake(reencoded, grants_other)

tidy(reencoded, 1)
```

---

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")
)
```

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().
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

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, unless passed in by `options`) 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).

Tidying

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

level character, the factor levels

value numeric, the encoding

terms character, the selectors or variables selected

id character, id of this step

Case weights

This step performs an supervised operation that can utilize case weights. To use them, see the documentation in recipes::case_weights and the examples on tidymodels.org.

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://CRAN.R-project.org/package=rstanarm/vignettes/pooling.html

"Prior Distributions for rstanarm Models" http://mc-stan.org/rstanarm/reference/priors.html

"Estimating Generalized (Non-)Linear Models with Group-Specific Terms with rstanarm" http://mc-stan.org/rstanarm/articles/glmer.html

See Also

    step_lencode(), step_lencode_glm(), step_lencode_mixed()

Examples

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

data(grants)

set.seed(1)
grants_other <- sample_n(grants_other, 500)

reencoded <- recipe(class ~ sponsor_code, data = grants_other) |>
  step_lencode_bayes(sponsor_code, outcome = vars(class))
```

---

step_lencode_glm   *Supervised Factor Conversions into Linear Functions using Likeli-*
                                 hood 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_glm")
)
```

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().
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

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).

Tidying

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

level character, the factor levels

value numeric, the encoding

terms character, the selectors or variables selected

id character, id of this step

Case weights

This step performs an supervised operation that can utilize case weights. To use them, see the documentation in recipes::case_weights and the examples on tidymodels.org.

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

See Also

step_lencode(), step_lencode_bayes(), step_lencode_mixed()

Examples

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

data(grants)

set.seed(1)
grants_other <- sample_n(grants_other, 500)

reencoded <- recipe(class ~ sponsor_code, data = grants_other) |>
  step_lencode_glm(sponsor_code, outcome = vars(class))
```

---

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_mixed")
)
```

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().
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

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).

Tidying

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

level character, the factor levels

value numeric, the encoding

terms character, the selectors or variables selected

id character, id of this step

Case weights

This step performs an supervised operation that can utilize case weights. To use them, see the documentation in recipes::case_weights and the examples on tidymodels.org.

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

See Also

step_lencode(), step_lencode_bayes(), step_lencode_glm()

Examples

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

data(grants)

set.seed(1)
grants_other <- sample_n(grants_other, 500)

reencoded <- recipe(class ~ sponsor_code, data = grants_other) |>
  step_lencode_mixed(sponsor_code, outcome = vars(class))
```

---

step_pca_sparse   *Sparse PCA Signal Extraction*

Description

`step_pca_sparse()` creates a *specification* of a recipe step that will convert numeric data into one or more principal components that can have some zero coefficients.

Usage

```
step_pca_sparse(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 5,
  predictor_prop = 1,
  options = list(),
```

```
    res = NULL,
    prefix = "PC",
    keep_original_cols = FALSE,
    skip = FALSE,
    id = rand_id("pca_sparse")
)
```

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 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 principal component 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.
num_comp	The number of components to retain as new predictors. If num_comp is greater than the number of columns or the number of possible components, a smaller value will be used. If num_comp = 0 is set then no transformation is done and selected variables will stay unchanged, regardless of the value of keep_original_cols.
predictor_prop	The maximum number of original predictors that can have non-zero coefficients for each PCA component (via regularization).
options	A list of options to the default method for irlba::ssvd().
res	The rotation matrix once this preprocessing step has be trained by recipes::prep.
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 irlba package is required for this step. If it is not installed, the user will be prompted to do so when the step is defined. The irlba::ssvd() function is used to encourage sparsity; that documentation has details about this method.

The argument num_comp controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with prefix and a sequence of numbers. The variable names are padded with zeros. For example, if num_comp < 10, their names will be PC1 - PC9. If num_comp = 101, the names would be PC1 - PC101.

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 selected), `value` (the loading), and `component`.

Tidying

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

terms character, the selectors or variables selected

value numeric, variable loading

component character, principle component

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `num_comp`: # Components (type: integer, default: 5)
- `predictor_prop`: Proportion of Predictors (type: double, default: 1)

Case weights

The underlying operation does not allow for case weights.

See Also

[recipes::step_pca()](#), [recipes::step_kpca()](#), [recipes::step_kpca_poly()](#), [recipes::step_kpca_rbf()](#), [step_pca_sparse_bayes()](#), [step_pca_truncated()](#)

Examples

```
library(recipes)
library(ggplot2)

data(ad_data, package = "modeldata")

ad_rec <-
  recipe(Class ~ ., data = ad_data) |>
  step_zv(all_predictors()) |>
  step_YeoJohnson(all_numeric_predictors()) |>
  step_normalize(all_numeric_predictors()) |>
  step_pca_sparse(
    all_numeric_predictors(),
    predictor_prop = 0.75,
    num_comp = 3,
    id = "sparse pca"
  ) |>
  prep()
```

```
tidy(ad_rec, id = "sparse pca") |>
  mutate(value = ifelse(value == 0, NA, value)) |>
  ggplot(aes(x = component, y = terms, fill = value)) +
  geom_tile() +
  scale_fill_gradient2() +
  theme(axis.text.y = element_blank())
```

step_pca_sparse_bayes   *Sparse Bayesian PCA Signal Extraction*

Description

step_pca_sparse_bayes() creates a *specification* of a recipe step that will convert numeric data into one or more principal components that can have some zero coefficients.

Usage

```
step_pca_sparse_bayes(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 5,
  prior_slab_dispersion = 1,
  prior_mixture_threshold = 0.1,
  options = list(),
  res = NULL,
  prefix = "PC",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("pca_sparse_bayes")
)
```

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 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 principal component 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.

num_comp   The number of components to retain as new predictors. If num_comp is greater
                    than the number of columns or the number of possible components, a smaller
                    value will be used. If num_comp = 0 is set then no transformation is done and se-
                    lected variables will stay unchanged, regardless of the value of keep_original_cols.

prior_slab_dispersion
                    This value is proportional to the dispersion (or scale) parameter for the slab
                    portion of the prior. Smaller values result in an increase in zero coefficients.

prior_mixture_threshold
                    The parameter that defines the trade-off between the spike and slab components
                    of the prior. Increasing this parameter increases the number of zero coefficients.

options   A list of options to the default method for [VBsparsePCA::VBsparsePCA()](#).

res   The rotation matrix once this preprocessing step has been trained by [recipes::prep](#).

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 VBsparsePCA package is required for this step. If it is not installed, the user will be prompted
to do so when the step is defined.

A spike-and-slab prior is a mixture of two priors. One (the "spike") has all of its mass at zero and
represents a variable that has no contribution to the PCA coefficients. The other prior is a broader
distribution that reflects the coefficient distribution of variables that do affect the PCA analysis. This
is the "slab". The narrower the slab, the more likely that a coefficient will be zero (or are regularized
to be closer to zero). The mixture of these two priors is governed by a mixing parameter, which
itself has a prior distribution and a hyper-parameter prior.

PCA coefficients and their resulting scores are unique only up to the sign. This step will attempt to
make the sign of the components more consistent from run-to-run. However, the sparsity constraint
may interfere with this goal.

The argument num_comp controls the number of components that will be retained (the original
variables that are used to derive the components are removed from the data). The new components
will have names that begin with prefix and a sequence of numbers. The variable names are padded
with zeros. For example, if num_comp < 10, their names will be PC1 - PC9. If num_comp = 101, the
names would be PC1 - PC101.

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 selected), value (the
loading), and component.

Tidying

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

terms character, the selectors or variables selected

value numeric, variable loading

component character, principle component

id character, id of this step

Tuning Parameters

This step has 3 tuning parameters:

- num_comp: # Components (type: integer, default: 5)
- prior_slab_dispersion: Dispersion of Slab Prior (type: double, default: 1)
- prior_mixture_threshold: Threshold for Mixture Prior (type: double, default: 0.1)

Case weights

The underlying operation does not allow for case weights.

References

Ning, B. (2021). Spike and slab Bayesian sparse principal component analysis. arXiv:2102.00305.

See Also

[`recipes::step_pca()`](#), [`recipes::step_kpca()`](#), [`recipes::step_kpca_poly()`](#), [`recipes::step_kpca_rbf()`](#), [`step_pca_sparse()`](#), [`step_pca_truncated()`](#)

Examples

```
library(recipes)
library(ggplot2)

data(ad_data, package = "modeldata")

ad_rec <-
  recipe(Class ~ ., data = ad_data) |>
  step_zv(all_predictors()) |>
  step_YeoJohnson(all_numeric_predictors()) |>
  step_normalize(all_numeric_predictors()) |>
  step_pca_sparse_bayes(
    all_numeric_predictors(),
    prior_mixture_threshold = 0.95,
    prior_slab_dispersion = 0.05,
    num_comp = 3,
    id = "sparse bayesian pca"
  ) |>
  prep()
```

```
tidy(ad_rec, id = "sparse bayesian pca") |>
  mutate(value = ifelse(value == 0, NA, value)) |>
  ggplot(aes(x = component, y = terms, fill = value)) +
  geom_tile() +
  scale_fill_gradient2() +
  theme(axis.text.y = element_blank())
```

---

step_pca_truncated   *Truncated PCA Signal Extraction*

Description

step_pca_truncated() creates a *specification* of a recipe step that will convert numeric data into
one or more principal components. It is truncated as it only calculates the number of components it
is asked instead of all of them as is done in `recipes::step_pca()`.

Usage

```
step_pca_truncated(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 5,
  options = list(),
  res = NULL,
  columns = NULL,
  prefix = "PC",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("pca_truncated")
)
```

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 this step. See `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.
num_comp	The number of components to retain as new predictors. If num_comp is greater than the number of columns or the number of possible components, a smaller value will be used. If num_comp = 0 is set then no transformation is done and selected variables will stay unchanged, regardless of the value of keep_original_cols.

options	A list of options to the default method for [irlba::prcomp_irlba()](). Argument defaults are set to `retx = FALSE`, `center = FALSE`, `scale. = FALSE`, and `tol = NULL`. **Note** that the argument x should not be passed here (or at all).
res	The [irlba::prcomp_irlba()]() object is stored here once this preprocessing step has be trained by [recipes::prep]().
columns	A character string of the selected variable names. This field is a placeholder and will be populated once [prep()]() is used.
prefix	A character string for the prefix of 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 [bake()]()? While all operations are baked when [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

Principal component analysis (PCA) is a transformation of a group of variables that produces a new set of artificial features or components. These components are designed to capture the maximum amount of information (i.e. variance) in the original variables. Also, the components are statistically independent from one another. This means that they can be used to combat large inter-variables correlations in a data set.

It is advisable to standardize the variables prior to running PCA. Here, each variable will be centered and scaled prior to the PCA calculation. This can be changed using the `options` argument or by using [recipes::step_center()]() and [recipes::step_scale()]().

The argument num_comp controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with prefix and a sequence of numbers. The variable names are padded with zeros. For example, if `num_comp < 10`, their names will be `PC1 - PC9`. If `num_comp = 101`, the names would be `PC1 - PC101`.

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Tidying

When you [tidy()]() this step two things can happen depending the type argument. If `type = "coef"` a tibble returned with 4 columns terms, value, component , and id:

terms character, the selectors or variables selected

value numeric, variable loading

component character, principle component

id character, id of this step

If type = "variance" a tibble returned with 4 columns terms, value, component , and id:

terms character, type of variance

value numeric, value of the variance

component integer, principle component

id character, id of this step

Tuning Parameters

This step has 1 tuning parameters:

- num_comp: # Components (type: integer, default: 5)

Case weights

This step performs an unsupervised operation that can utilize case weights. As a result, case weights are only used with frequency weights. For more information, see the documentation in recipes::case_weights and the examples on tidymodels.org.

References

Jolliffe, I. T. (2010). *Principal Component Analysis*. Springer.

See Also

recipes::step_pca(), recipes::step_kpca(), recipes::step_kpca_poly(), recipes::step_kpca_rbf(), step_pca_sparse_bayes(), step_pca_sparse()

Examples

```
rec <- recipe(~., data = mtcars)
pca_trans <- rec |>
  step_normalize(all_numeric()) |>
  step_pca_truncated(all_numeric(), num_comp = 2)
pca_estimates <- prep(pca_trans, training = mtcars)
pca_data <- bake(pca_estimates, mtcars)

rng <- extendrange(c(pca_data$PC1, pca_data$PC2))
plot(pca_data$PC1, pca_data$PC2,
  xlim = rng, ylim = rng
)

tidy(pca_trans, number = 2)
tidy(pca_estimates, number = 2)
```

---

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,
  metric = "euclidean",
  learn_rate = 1,
  epochs = NULL,
  initial = "spectral",
  target_weight = 0.5,
  options = list(verbose = FALSE, n_threads = 1),
  seed = sample(10^5, 2),
  prefix = "UMAP",
  keep_original_cols = FALSE,
  retain = deprecated(),
  object = NULL,
  skip = FALSE,
  id = rand_id("umap")
)
```

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 this step. See [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.

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. If `neighbors` is greater than the number of data points, the smaller value is used.
num_comp	An integer for the number of UMAP components. If `num_comp` is greater than the number of selected columns minus one, the smaller value is used.
min_dist	The effective minimum distance between embedded points.
metric	Character, type of distance metric to use to find nearest neighbors. See uwot::umap() for more details. Default to `"euclidean"`.
learn_rate	Positive number of the learning rate for the optimization process.
epochs	Number of iterations for the neighbor optimization. See uwot::umap() for more details.
initial	Character, Type of initialization for the coordinates. Can be one of `"spectral"`, `"normlaplacian"`, `"random"`, `"lvrandom"`, `"laplacian"`, `"pca"`, `"spca"`, `"agspectral"`, or a matrix of initial coordinates. See uwot::umap() for more details. Default to `"spectral"`.
target_weight	Weighting factor between data topology and target topology. A value of 0.0 weights entirely on data, a value of 1.0 weights entirely on target. The default of 0.5 balances the weighting equally between data and target.
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 recipes::prep or recipes::bake.
prefix	A character string for the prefix of the resulting new variables. See notes below.
keep_original_cols	
	A logical to keep the original variables in the output. Defaults to `FALSE`.
retain	Use `keep_original_cols` instead to specify whether the original predictors should be retained along with the new embedding variables.
object	An object that defines the encoding. This is `NULL` until the step is trained by recipes::prep().
skip	A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when 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

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).

The argument num_comp controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with prefix and a sequence of numbers. The variable names are padded with zeros. For example, if num_comp < 10, their names will be UMAP1 - UMAP9. If num_comp = 101, the names would be UMAP1 - UMAP101.

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

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 7 tuning parameters:

- num_comp: # Components (type: integer, default: 2)

- neighbors: # Nearest Neighbors (type: integer, default: 15)

- min_dist: Min Distance between Points (type: double, default: 0.01)

- learn_rate: Learning Rate (type: double, default: 1)

- epochs: # Epochs (type: integer, default: NULL)

- initial: UMAP Initialization (type: character, default: spectral)

- target_weight: Proportion Supervised (type: double, default: 0.5)

Case weights

The underlying operation does not allow for case weights.

Saving prepped recipe object

This recipe step may require native serialization when saving for use in another R session. To learn more about serialization for prepped recipes, see the bundle package.

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(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 = UMAP1, y = UMAP2, 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",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("woe")
)
```

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 [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 woe components columns created by the original variables will be used as predictors in a model.
outcome	The bare name of the binary outcome encased in vars().
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 known 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.
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

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 [recipes::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 respective 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.

Tidying

When you [`tidy()`](#) this step, a tibble with columns `terms` (the selectors or variables selected), `value`, `n_tot`, `n_bad`, `n_good`, `p_bad`, `p_good`, `woe` and `outcome` is returned.. See [`dictionary()`](#) for more information.

When you [`tidy()`](#) this step, a tibble is returned with columns `terms` `value`, `n_tot`, `n_bad`, `n_good`, `p_bad`, `p_good`, `woe` and `outcome` and `id`:

terms character, the selectors or variables selected

value character, level of the outcome

n_tot integer, total number

n_bad integer, number of bad examples

n_good integer, number of good examples

p_bad numeric, p of bad examples

p_good numeric, p of good examples

woe numeric, weight of evidence

outcome character, name of outcome variable

id character, id of this step

Tuning Parameters

This step has 1 tuning parameters:

- `Laplace`: Laplace Correction (type: double, default: 1e-06)

Case weights

The underlying operation does not allow for case weights.

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 = vars(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 = vars(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, call = rlang::caller_env(0))
```

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.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of rlang::abort() for more information.

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