Package ‘scorecardModelUtils’

October 14, 2022

Type Package

Title Credit Scorecard Modelling Utils

Version 0.0.1.0

Maintainer Arya Poddar <aryapoddar290990@gmail.com>

Description Provides infrastructure functionalities such as missing value treatment, information value calculation, GINI calculation etc. which are used for developing a traditional credit scorecard as well as a machine learning based model. The functionalities defined are standard steps for any credit underwriting scorecard development, extensively used in financial domain.

License GPL-2 | GPL-3

LazyData TRUE

RoxygenNote 6.0.1

Imports car, e1071, gbm, partykit, randomForest, reshape2, sqldf, stringr, stats, ggplot2, utils

NeedsCompilation no

Author Arya Poddar [aut, cre],
Aiana Goyal [ctb],
Kanishk Dogar [ctb]

Repository CRAN

Date/Publication 2019-04-14 20:53:03 UTC

R topics documented:
categorical_iv

cat_new_class

class_cat_class

cv_filter

cv_table

cv_test

dtree_split_val

dtree_trend_iv

fn_conf_mat
The function takes base data, target and the categorical variable for which IV is to be calculated. It returns a dataframe with the WOE and IV value of the variable.

Usage

categorical_iv(base, target, variable, event = 1)

Arguments

- **base**: input dataframe
- **target**: column / field name for the target variable to be passed as string (must be 0/1 type)
- **variable**: categorical variable name for which IV is to be calculated, to be passed as string
- **event**: (optional) the event class, to be passed as 0 or 1 (default is 1)

Value

The function returns a dataframe.
Author(s)
Arya Poddar <aryapoddar290990@gmail.com>
Aiana Goyal <aianagoel002@gmail.com>

Examples
```r
data <- iris
data$Species <- as.character(data$Species)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
cat_iv <- categorical_iv(base = data,target = "Y",variable = "Species",event = 1)
```

Description
The function groups classes of categorical variables, which have population percentage less than a threshold, with another class of similar event rate. If a class of exactly same event rate is not available, it is clubbed with the one having a higher event rate closest to it.

Usage
```
cat_new_class(base, target, cat_var_name, threshold, event = 1)
```

Arguments
- **base**: input dataframe
- **target**: column / field name for the target variable to be passed as string (must be 0/1 type)
- **cat_var_name**: column name or array of column names of categorical variable on which the operation is to be done, to be passed as string
- **threshold**: threshold population percentage below which the class will be considered to be clubbed with another class, to be provided as decimal/fraction
- **event**: (optional) the event class, to be passed as 0 or 1 (default is 1)

Value
The function returns an object of class "cat_new_class" which is a list containing the following components:
- **base_new**: a dataframe after clubbing low percentage classes with another class of similar or closest but higher event rate
- **cat_class_new**: a dataframe with mapping between original classes and new clubbed classes (if any)
Author(s)

Arya Poddar <aryapoddar290990@gmail.com>
Kanishk Dogar <Kanishkd4@gmail.com>

Examples
data <- iris[1:110,]
data$Species <- as.character(data$Species)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
data_newclass <- cat_new_class(base = data,target = "Y",cat_var_name = "Species",threshold = 0.1)

Description

The function groups classes of categorical variable, which have population percentage less than a threshold, with another class of similar event rate. If a class of exactly same event rate is not available, it is clubbed with the one having a higher event rate closest to it.

Usage

club_cat_class(base, target, variable, threshold, event = 1)

Arguments

base
input dataframe
target
column / field name for the target variable to be passed as string (must be 0/1 type)
variable
column name of categorical variable on which the operation is to be done, to be passed as string
threshold
threshold population percentage below which the class will be considered to be clubbed with another class, to be provided as decimal/fraction
event (optional) the event class, to be passed as 0 or 1 (default is 1)

Value

The function returns a dataframe after clubbing low percentage classes with another class of similar or closest but higher event rate.

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>
Kanishk Dogar <kanishkd4@gmail.com>
Examples

```r
data <- iris[1:110,]
data$Species <- as.character(data$Species)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
data_clubclass <- club_cat_class(base = data,target = "Y",variable = "Species",threshold = 0.2)
```

---

**cv_filter**

*Variable reduction based on Cramer’s V filter*

---

**Description**

The function returns a list of variables that can be dropped because of high correlation with another variable, based on Cramer’s V and IV. If V1 and V2 have a Cramer’s V value more than a user defined threshold, the variable with lower IV will be recommended to be dropped by this function. The variable which got dropped won’t be considered for dropping any more variables.

**Usage**

```r
cv_filter(cv_table, iv_table, threshold)
```

**Arguments**

- `cv_table`: dataframe of class cv_table with three columns - var_1, var_2, cv_value
- `iv_table`: dataframe of class iv_table with two columns - Variable_name, iv
- `threshold`: Cramers’ V value above which one of the variable will be recommended to be dropped

**Value**

An object of class "cv_filter" is a list containing the following components:

- `retain_var_list`: list of variables remaining post CV filter
- `dropped_var_list`: list of variables that can be dropped based on CV filter
- `dropped_var_tab`: CV correlation value for dropped variables as a dataframe
- `threshold`: threshold CV value used as input parameter

**Author(s)**

Arya Poddar <aryapoddar290990@gmail.com>
**Examples**

```r
data <- iris
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
cv_tab_list <- cv_table(data, c("Species", "Sepal.Length"))
cv_tab <- cv_tab_list$cv_val_tab
x <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width")
iv_table_list <- iv_table(base = data,target = "Y",num_var_name = x,cat_var_name = "Species")
iv_tab <- iv_table_list$iv_table
cv_filter_list <- cv_filter(cv_table = cv_tab,iv_table = iv_tab,threshold = 0.5)
cv_filter_list$retain_var_list
cv_filter_list$dropped_var_list
cv_filter_list$dropped_var_tab
cv_filter_list$threshold
```

---

**cv_table**

*Pairwise Cramer’s V among a list of categorical variables*

**Description**

The function gives a dataframe with pairwise Cramer’s V value between all possible combination of categorical variables from the list of variables provided.

**Usage**

```r
cv_table(base, column_name)
```

**Arguments**

- `base`: input dataframe
- `column_name`: column name or array of column names for which Cramer’s V is to be calculated

**Value**

An object of class "cv_table" is a list containing the following components:

- `cv_val_tab`: pairwise Cramer’s V value as a dataframe
- `single_class_var_index`: array of column index of variables with only one class

**Author(s)**

Arya Poddar <aryapoddar290990@gmail.com>
Examples

```r
data <- iris
data$Species <- as.character(data$Species)
data$Sepal.Length <- as.character(floor(data$Sepal.Length))
cv_tab_list <- cv_table(data, c("Species", "Sepal.Length"))
cv_tab_list$cv_val_tab
cv_tab_list$single_class_var_index
```

---

**cv_test**  
*Cramer’s V value between two categorical variables*

Description

The function gives the pairwise Cramer’s V value between two input categorical variables.

Usage

```r
cv_test(base, var_1, var_2)
```

Arguments

- **base**: input dataframe
- **var_1**: categorical variable name, to be passed as string
- **var_2**: categorical variable name, to be passed as string

Value

The function returns a dataframe with pairwise CV value.

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

```r
data <- iris
data$Species <- as.character(data$Species)
data$Sepal.Length <- as.character(floor(data$Sepal.Length))
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
cv_result <- cv_test(base = data,var_1 = "Species",var_2 = "Sepal.Length")
```
dtree_split_val  Getting the split value for terminal nodes from decision tree

Description
The function takes a ctree type model, with only one numerical variable, as argument input and
gives a dataframe with the minimum and maximum value of each node. The intervals are open
ended at lower limit and closed at upper limit.

Usage
dtree_split_val(desc_model, variable)

Arguments
  desc_model  ctree class model with one variable
  variable  numerical variable name which on which decision tree was run, to be passed as
            string

Value
The function returns a dataframe giving the lower and upper bound of split values of each node.

Author(s)
Arya Poddar <aryapoddar290990@gmail.com>

Examples
  data <- iris
  data$Y <- ifelse(data$Species=="setosa",1,0)


dtree_trend_iv  Recursive Decision Tree partitioning with monotonic event rate along
with IV table for individual numerical variable

Description
The function takes base data, target and the numerical variable which is to be binned. It returns
the optimal cuts based on recursive partitioning decision tree such that the trend of event rate holds
good i.e. it is strictly monotonically increasing or decreasing. If missing values are imputed by any
extreme value, the same can be passed as an argument, and it will be shown as a different category.
The output is a dataframe with the WOE and IV value.
Usage

dtree_trend_iv(base, target, variable, num_missing = -99999,
              mincriterion = 0.1, event = 1)

Arguments

base         input dataframe
target       column / field name for the target variable to be passed as string (must be 0/1 type)
variable     numerical variable name which is to be binned into categorical buckets, to be passed as string
num_missing  (optional) imputed missing value for numerical variable or an array of values which are to be kept as different bucket in binning step (default value is -99999)
mincriterion (optional) the value of the test statistic or (1 - p-value) that must be exceeded in order to implement a split (default value is 0.1)
event        (optional) the event class, to be passed as 0 or 1 (default is 1)

Value

The function returns a dataframe with count and iv.

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>
Aiana Goyal <aianagoel002@gmail.com>

Examples

data <- iris
data$Y <- ifelse(data$Species=="setosa",1,0)
dtree_trend_tab <- dtree_trend_iv(base = data, target = "Y", variable = "Sepal.Length", event = 1)

fn_conf_mat

Creates confusion matrix and its related measures

Description

The function takes the base dataframe with observed/actual and predicted columns. The actual/predicted class preferably should be binary and if not, it will be considered as event vs rest. It computes the performance measures like accuracy, precision, recall, sensitivity, specificity and f1 score.

Usage

fn_conf_mat(base, observed_col, predicted_col, event)
Arguments

- base: input dataframe
- observed_col: column / field name of the observed event
- predicted_col: column / field name of the predicted event
- event: the event class, to be passed as string

Value

An object of class "fn_conf_mat" is a list containing the following components:

- confusion_mat: confusion matrix as a table
- accuracy: accuracy measure
- precision: precision measure
- recall: recall measure
- sensitivity: sensitivity measure
- specificity: specificity measure
- f1_score: F1 score

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris
data$Species <- as.character(data$Species)
suppressWarnings(RNGversion("3.5.0"))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
data$Y_pred <- sample(0:1,size=nrow(data),replace=TRUE)
fn_conf_mat_list <- fn_conf_mat(base = data, observed_col = "Y", predicted_col = "Y_pred", event = 1)
fn_conf_mat_list$confusion_mat
fn_conf_mat_list$accuracy
fn_conf_mat_list$precision
fn_conf_mat_list$recall
fn_conf_mat_list$sensitivity
fn_conf_mat_list$specificity
fn_conf_mat_list$f1_score
fn_cross_index

Creates random index for k-fold cross validation

Description

The function base and returns a list of length k, to be used for k-fold cross validation sampling. Each element of the returned list is an array of random index for sampling for k-fold cross validation.

Usage

fn_cross_index(base, k)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>input dataframe</td>
</tr>
<tr>
<td>k</td>
<td>number of cross validation</td>
</tr>
</tbody>
</table>

Value

The function a list of length k, each holding an array of index/row number for sampling the base.

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris
data$Species <- as.character(data$Species)
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1, size=nrow(data), replace=TRUE)
data$Y_pred <- sample(0:1, size=nrow(data), replace=TRUE)
data_k_list <- fn_cross_index(base = data, k = 5)
data_k_list$index1
data_k_list$index2
data_k_list$index3
data_k_list$index4
data_k_list$index5
fn_error

Computes error measures between observed and predicted values

Description

The function takes the input dataframe with observed and predicted columns and computes mean absolute error, mean squared error and root mean squared error terms.

Usage

fn_error(base, observed_col, predicted_col)

Arguments

base
input dataframe

observed_col
column / field name of the observed event

predicted_col
column / field name of the predicted event

Value

An object of class "fn_error" is a list containing the following components:

mean_abs_error
mean absolute error between observed and predicted value

mean_sq_error
mean squared error between observed and predicted value

root_mean_sq_error
root mean squared error between observed and predicted value

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris
data$Species <- as.character(data$Species)
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
data$Y_pred <- sample(0:1,size=nrow(data),replace=TRUE)
fn_error_list <- fn_error(base = data,observed_col = "Y",predicted_col = "Y_pred")
fn_error_list$mean_abs_error
fn_error_list$mean_sq_error
fn_error_list$root_mean_sq_error
**fn_mode**

*Calculating mode value of a vector*

**Description**

The function returns the mode of a vector. The vector can be of any datatype ie. numerical or categorical.

**Usage**

```r
fn_mode(x)
```

**Arguments**

- `x` a vector of string or number

**Value**

The function returns the mode value of the input vector.

**Author(s)**

Arya Poddar <aryapoddar290990@gmail.com>

**Examples**

```r
fn_mode(c(1,2,3,1,4,1,7))
```

---

**fn_target**

*Redefines target value*

**Description**

The function redefines the "binary" target variable to be used for modelling. It takes the variable or field name of the target and the event class. It changes the target field name to "Target", changes the events into 1 and non-events as 0 and places the target column at the end of the dataframe before returning it as output.

**Usage**

```r
fn_target(base, target, event)
```

**Arguments**

- `base` input dataframe
- `target` column / field name for the target variable, to be passed as string
- `event` the event class, to be passed as string
**Value**

The function returns a dataframe after changing the target classes to 0 or 1.

**Author(s)**

Arya Poddar <aryapoddar290990@gmail.com>

**Examples**

```r
data <- iris
data$Species <- as.character(data$Species)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)

data2 <- fn_target(base = data,target = "Y",event = 1)
```

---

**gini_table**

*Performance measure table with Gini coefficient, KS-statistics and Gini lift curve*

**Description**

The function takes a dataframe along with a model or the name of a column with predicted value. If a model (only lm or glm works is guaranteed to work perfectly) is provided as argument, the response on the data is predicted. Otherwise, if the data already contains a predicted column, it can be referred as an argument. The predicted column, thus obtained, is classified into bands to get the Gini coefficient, Kolmogorov-Smirnov statistics and Gini lift curve. The number of bands required can be passed as argument, with default value as 10 i.e. decile binning is done. Otherwise, the cutpoints for converting the predicted value into bands can also be specified.

**Usage**

```r
gini_table(base, target, col_pred = F, model = F, brk = F, quantile_pt = 10, event_rate_direction = "decreasing")
```

**Arguments**

- `base`: input dataframe
- `target`: column / field name for the target variable to be passed as string (must be 0/1 type)
- `col_pred`: (optional) column name which contains the predicted value, not required if "model"=TRUE (default value is FALSE)
- `model`: (optional) object of type lm or glm model, required only if "col_pred"=FALSE (default value is FALSE)
- `brk`: (optional) array of break points of predicted value (default value is FALSE)
- `quantile_pt`: (optional) number of quantiles to divide the predicted value range (default value is 10)
event_rate_direction
(optional) directionality of event rate with increasing value of predicted column,
to be chosen among "increasing" or "decreasing" (default value is decreasing)

Value

An object of class "gini_table" is a list containing the following components:

- prediction: base with the predicted value as a dataframe
- gini_tab: gini table as a dataframe
- gini_value: gini coefficient value
- gini_plot: gini curve plot
- ks_value: Kolmogorov-Smirnov statistic
- breaks: break points

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>
Aiana Goyal <aianagoel002@gmail.com>

Examples

data <- iris
data$Species <- as.character(data$Species)
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y_pred <- sample(300:900,size=nrow(data),replace=TRUE)
gini_tab_list <- gini_table(base = data,target = "Y",col_pred = "Y_pred",quantile_pt = 10)
gini_tab_list$prediction
gini_tab_list$gini_tab
gini_tab_list$gini_value
gini_tab_list$gini_plot
gini_tab_list$ks_value
gini_tab_list$breaks
Description

The function runs a grid search with k-fold cross validation to arrive at best parameter decided by some performance measure. The parameters that can be tuned using this function for gradient boosting regression modelling algorithm are - ntree, depth, shrinkage, min_obs and bag_fraction. The objective function to be minimised is the error (mean absolute error / mean squared error / root mean squared error). For the grid search, the possible values of each tuning parameter needs to be passed as an array into the function.

Usage

`gradient_boosting_parameters(base, target, ntree, depth, shrinkage, min_obs, bag_fraction, error = "rmse", cv = 1)`

Arguments

- **base**: input dataframe
- **target**: column / field name for the target variable to be passed as string (must be 0/1 type)
- **ntree**: number of trees to be fitted
- **depth**: maximum depth of variable interactions
- **shrinkage**: learning rate
- **min_obs**: minimum size of terminal nodes
- **bag_fraction**: fraction of the training set observations randomly selected for next tree
- **error**: (optional) error measure as objective function to be minimised, to be chosen among "mae", "mse" and "rmse" (default value is "rmse")
- **cv**: (optional) k value for k-fold cross validation to be performed (default value is 1 ie. without cross validation)

Value

An object of class "gradient_boosting_parameters" is a list containing the following components:

- **error_tab_detailed**: error summary for each cross validation sample of the parameter combinations iterated during grid search as a dataframe
- **error_tab_summary**: error summary for each combination of parameters as a dataframe
- **best_ntree**: ntree parameter of the optimal solution
- **best_depth**: depth parameter of the optimal solution
- **best_shrinkage**: shrinkage parameter of the optimal solution
- **best_min_obs**: cost min_obs of the optimal solution
- **best_bag_fraction**: bag_fraction parameter of the optimal solution
- **runtime**: runtime of the entire process
iv_filter

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
gbm_params_list <- gradient_boosting_parameters(base = data,target = "Y",ntree = 2,depth = 2,
   shrinkage = 0.1,min_obs = 0.1,bag_fraction = 0.7)
gbm_params_list$error_tab_detailed
gbm_params_list$error_tab_summary
gbm_params_list$best_ntree
gbm_params_list$best_depth
gbm_params_list$best_shrinkage
gbm_params_list$best_min_obs
gbm_params_list$best_bag_fraction
gbm_params_list$runtime

iv_filter Variable reduction based on Information Value filter

Description

The function returns a list of variables that can be dropped because of low discriminatory power,
based on Information Value. If IV for a variable is less than a user defined threshold, the variable
will be recommended to be dropped by this function.

Usage

iv_filter(base, iv_table, threshold)

Arguments

base input dataframe
iv_table dataframe of class iv_table with two columns - Variable_name, iv
threshold threshold IV value below which the variable will be recommended to be dropped

Value

An object of class "iv_filter" is a list containing the following components:

retain_var_tab variables remaining post IV filter as a dataframe
retain_var_name array of column names of variables to be retained
dropped_var_tab variables that can be dropped based on IV filter as a dataframe
threshold threshold IV value used as input parameter
Author(s)
Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
x <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width")
iv_table_list <- iv_table(base = data,target = "Y",num_var_name = x,cat_var_name = "Species")
ivf_list <- iv_filter(base = data,iv_table = iv_table_list$iv_table,threshold = 0.02)
ivf_list$retain_var_tab
ivf_list$retain_var_name
ivf_list$dropped_var_tab
ivf_list$threshold

---

iv_table  
WOE and IV table for list of numerical and categorical variables

Description

The function takes column indices of categorical and numerical variables and returns a list with four dataframes - WOE table of numerical variables, categorical variables, consolidated table of both numerical & categorical variables and a IV table.

Usage

iv_table(base, target, num_var_name = F, num_missing = -99999, cat_var_name = F, mincriterion = 0.1, event = 1)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>input dataframe</td>
</tr>
<tr>
<td>target</td>
<td>column / field name for the target variable to be passed as string (must be 0/1 type)</td>
</tr>
<tr>
<td>num_var_name</td>
<td>column name or array of column names of numerical variable for which IV is to be calculated, to be passed as string</td>
</tr>
<tr>
<td>num_missing</td>
<td>(optional) imputed missing value for numerical variable or an array of values which are to be kept as different bucket in binning step (default value is -99999)</td>
</tr>
<tr>
<td>cat_var_name</td>
<td>column name or array of column names of categorical variable for which IV is to be calculated, to be passed as string</td>
</tr>
<tr>
<td>mincriterion</td>
<td>(optional) the value of the test statistic or (1 - p-value) that must be exceeded in order to implement a split (default value is 0.1)</td>
</tr>
<tr>
<td>event</td>
<td>(optional) the event class, to be passed as 0 or 1 (default is 1)</td>
</tr>
</tbody>
</table>
Value

An object of class "iv_table" is a list containing the following components:

- **num_woe_table**: numerical woe table with IV as a dataframe
- **cat_woe_table**: categorical woe table with IV as a dataframe
- **woe_table**: numerical and categorical woe table with IV as a dataframe
- **iv_table**: Variable with IV value as a dataframe

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>
Aiana Goyal <aianagoel002@gmail.com>
Kanishk Dogar <kanishkdogar4@gmail.com>

Examples

data <- iris
data$Species <- as.character(data$Species)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
x <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width")
iv_table_list <- iv_table(base = data,target = "Y",num_var_name = x,cat_var_name = "Species")
iv_table_list$num_woe_table
iv_table_list$cat_woe_table
iv_table_list$woe_table
iv_table_list$iv_table

Usage

```r
missing_val(base, target, num_missing = -99999,
cat_missing = "missing_value")
```
num_to_cat

Binning numerical variables based on cuts from IV table

Description

The function takes the num_woe_table output from a class "iv_table". Based on the split points from the num_woe_table, the numerical variables are binned into categories.

Usage

```
num_to_cat(base, num_woe_table, num_missing = -99999)
```
others_class

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>input dataframe</td>
</tr>
<tr>
<td>num_woe_table</td>
<td>num_woe_table class from iv table output</td>
</tr>
<tr>
<td>num_missing</td>
<td>(optional) imputed missing value for numerical variable (default value is -99999)</td>
</tr>
</tbody>
</table>

Value

The function returns a dataframe after converting the numerical variables into categorical classes.

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
x <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width")
iv_table_list <- iv_table(base = data,target = "Y",num_var_name = x,cat_var_name = "Species")
num_cat <- num_to_cat(base = data,num_woe_table = iv_table_list$num_woe_table)

others_class

Clubbing of classes of categorical variable with low population percentage into one class

Description

The function groups the classes of a categorical variable which have population percentage less than a threshold as "Low_pop_perc". The user can choose whether to club the missing class or keep it as separate class. The default setting is that missing classes are not treated separately.

Usage

others_class(base, target, column_name, threshold, char_missing = NA)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>input dataframe</td>
</tr>
<tr>
<td>target</td>
<td>column / field name for the target variable to be passed as string (must be 0/1 type)</td>
</tr>
<tr>
<td>column_name</td>
<td>column name or array of column names of the dataframe on which the operation is to be done</td>
</tr>
<tr>
<td>threshold</td>
<td>threshold population percentage below which the class is to be classified as others, to be provided as decimal/fraction</td>
</tr>
<tr>
<td>char_missing</td>
<td>(optional) imputed missing value for categorical variable if its to be kept separate (default value is NA)</td>
</tr>
</tbody>
</table>
random_forest_parameters

Value

- **base**: a dataframe after converting all low percentage classes into "Low_pop_perc" class
- **mapping_table**: a dataframe with mapping between original classes which are now "Low_pop_perc" class (if any)

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris[c(1:110),]
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
data$Species <- as.character(data$Species)
data_otherclass <- others_class(base = data,target = "Y",column_name = "Species",threshold = 0.15)

random_forest_parameters

Hyperparameter optimisation or parameter tuning for Random Forest by grid search

Description

The function runs a grid search with k-fold cross validation to arrive at best parameter decided by some performance measure. The parameters that can be tuned using this function for random forest algorithm are - ntree, mtry, maxnodes and nodesize. The objective function to be minimised is the error (mean absolute error / mean squared error / root mean squared error). For the grid search, the possible values of each tuning parameter needs to be passed as an array into the function.

Usage

```r
random_forest_parameters(base, target, model_type, ntree, mtry,
                          maxnodes = NULL, nodesize, error = "rmse", cv = 1)
```

Arguments

- **base**: input dataframe
- **target**: column / field name for the target variable to be passed as string (must be 0/1 type)
- **model_type**: to be chosen among "regression" or "classification"
- **ntree**: number of trees to be fitted
- **mtry**: number of variable to be sampled as split criteria at each node
- **maxnodes**: (optional) Maximum number of terminal nodes (default is NULL ie. no restriction on depth of the trees)
nodesize  minimum size of terminal nodes

error  (optional) error measure as objective function to be minimised, to be chosen among "mae", "mse" and "rmse" (default value is "rmse")

cv  (optional) k value for k-fold cross validation to be performed (default value is 1 ie. without cross validation)

Value

An object of class "random_forest_parameters" is a list containing the following components:

error_tab_detailed  error summary for each cross validation sample of the parameter combinations iterated during grid search as a dataframe

error_tab_summary  error summary for each combination of parameters as a dataframe

best_ntree  ntree parameter of the optimal solution

best_mtry  mtry parameter of the optimal solution

maxnodes  maxnodes parameter of the optimal solution

best_nodesize  nodesize parameter of the optimal solution

runtime  runtime of the entire process

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Aiana Goyal <aianagoel002@gmail.com>

Examples

data <- iris
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
rf_params_list <- random_forest_parameters(base = data,target = "Y",
model_type = "classification",ntree = 2,mtry = 1,nodesize = 3)
rf_params_list$error_tab_detailed
rf_params_list$error_tab_summary
rf_params_list$best_ntree
rf_params_list$best_mtry
rf_params_list$maxnodes
rf_params_list$best_nodesize
rf_params_list$runtime
Sampling

Random sampling of data into train and test

Description

The function does random sampling of the data and split it into train and test datasets. Training base percentage and seed value (optional) is taken as arguments. If seed value is not specified, random seed will be generated on different iterations.

Usage

sampling(base, train_perc = 0.7, seed = NA, replace = FALSE)

Arguments

base
input dataframe

train_perc
(optional) percentage of total base to be kept as training sample, to be provided as decimal/fraction (default percentage is 0.7)

seed
(optional) seed value (if not given random seed is generated)

replace
(optional) whether replacement will be with or without replacement (default is FALSE i.e. without replacement)

Value

An object of class "sampling" is a list containing the following components:

train_sample
training sample as a dataframe

test_sample
test sample as a dataframe

seed
seed used

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris
sampling_list <- sampling(base = data, train_perc = 0.7, seed = 1234)
sampling_list$train
sampling_list$test
sampling_list$seed
Converting coefficients of logistic regression into scores for scorecard building

**Description**

The function takes a logistic model as input and scales the coefficients into scores to be used for scorecard generation. The

**Usage**

```
scalling(base, target, model, point = 15, factor = 2, setscore = 660)
```

**Arguments**

- `base` : base input dataframe
- `target` : column / field name for the target variable to be passed as string (must be 0/1 type)
- `model` : input logistic model from which the coefficients are to be picked
- `point` : (optional) points after which the log odds will get multiplied by "factor" (default value is 15)
- `factor` : (optional) factor by which the log odds must get multiplied after a step of "points" (default value is 2)
- `setscore` : (optional) input for setting offset (default value is 660)

**Value**

The function returns a dataframe with the coefficients and scaled scores for each class of all explanatory variables of the model.

**Author(s)**

Arya Poddar <aryapoddar290990@gmail.com>

**Examples**

```r
data <- iris
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
x <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width")
iv_table_list <- iv_table(base = data,target = "Y",num_var_name = x,cat_var_name = "Species")
num_cat <- num_to_cat(base = data,num_woe_table = iv_table_list$num_woe_table)
log_model <- glm(Y ~ ., data = num_cat, family = "binomial")
scaling_tab <- scalling(base = num_cat,target = "Y",model = log_model)
```
Scoring a dataset with class based on a scaling logic to arrive at final score

**Description**

The function takes the data, with each variable as class. The dataframe of class scaling is used to convert the class into scores and finally arrive at the row level final scores by adding up the score values.

**Usage**

```r
scoring(base, target, scalling)
```

**Arguments**

- `base`: input dataframe with classes same as scaling logic
- `target`: column / field name for the target variable to be passed as string (must be 0/1 type)
- `scalling`: dataframe of class scaling with at least two columns - Variable, Category, Coefficient, D(i,j)_hat, Score

**Value**

The function returns a dataframe with classes converted to scores and the final score for each record in the input dataframe.

**Author(s)**

Arya Poddar <aryapoddar290990@gmail.com>

**Examples**

```r
data <- iris
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
x <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width")
iv_table_list <- iv_table(base = data,target = "Y",num_var_name = x,cat_var_name = "Species")
num_cat <- num_to_cat(base = data,num_woe_table = iv_table_list$num_woe_table)
log_model <- glm(Y ~ ., data = num_cat, family = "binomial")
scaling_tab <- scalling(base = num_cat,target = "Y",model = log_model)
score_tab <- scoring(base = num_cat,target = "Y",scalling = scaling_tab)
```
**support_vector_parameters**

*Hyperparameter optimisation or parameter tuning for Support Vector Machine by grid search*

**Description**

The function runs a grid search with k-fold cross validation to arrive at best parameter decided by some performance measure. The parameters that can be tuned using this function for support vector machine algorithm are - kernel (linear / polynomial / radial / sigmoid), degree of polynomial, gamma and cost. The objective function to be minimised is the error (mean absolute error / mean squared error / root mean squared error). For the grid search, the possible values of each tuning parameter needs to be passed as an array into the function.

**Usage**

```
support_vector_parameters(base, target, scale = T, kernel, degree = 2, gamma, cost, error = "rmse", cv = 1)
```

**Arguments**

- **base**: input dataframe
- **target**: column / field name for the target variable to be passed as string (must be 0/1 type)
- **scale**: (optional) logical vector indicating the variables to be scaled (default value is TRUE)
- **kernel**: an array of kernels to be iterated on; kernel used in training and predicting, to be chosen among "linear", "polynomial", "radial" and "sigmoid"
- **degree**: (optional) an array of degree of polynomial to be iterated on; parameter needed for kernel of type "polynomial" (default value is 2)
- **gamma**: an array of gamma values to be iterated on; parameter needed for all kernels except linear
- **cost**: an array of cost to be iterated on; cost of constraints violation
- **error**: (optional) error measure as objective function to be minimised, to be chosen among "mae", "mse" and "rmse" (default value is "rmse")
- **cv**: (optional) k value for k-fold cross validation to be performed (default value is 1 ie. without cross validation)

**Value**

An object of class "support_vector_parameters" is a list containing the following components:

- **error_tab_detailed**: error summary for each cross validation sample of the parameter combinations iterated during grid search as a dataframe
error_tab_summary

error summary for each combination of parameters as a dataframe

best_kernel

kernel parameter of the optimal solution

best_degree

degree parameter of the optimal solution

best_gamma

gamma parameter of the optimal solution

best_cost

cost parameter of the optimal solution

runtime

runtime of the entire process

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

```r
data <- iris
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
svm_params_list <- support_vector_parameters(base = data,target = "Y",gamma = 0.1,
cost = 0.1,kernel = "radial")
svm_params_list$error_tab_detailed
svm_params_list$error_tab_summary
svm_params_list$best_kernel
svm_params_list$best_degree
svm_params_list$best_gamma
svm_params_list$best_cost
svm_params_list$runtime
```

univariate

Univariate analysis of variables

Description

The function gives univariate analysis of the variables as output dataframe. The univariate statistics includes - minimum, maximum, mean, median, number of distinct values, variable type, counts of null value, percentage of null value, maximum population percentage among all classes/values, correlation with target. It also returns the list of names of character and numerical variable types along with variable name with population concentration more than a threshold at a class/value.

Usage

```r
univariate(base, target, threshold)
```

Arguments

- base: input dataframe
- target: column / field name for the target variable to be passed as string (must be 0/1 type)
- threshold: sparsity threshold, to be provided as decimal/fraction
vif_filter

Value

The function returns an object of class "univariate" which is a list containing the following components:

- **univar_table**: univariate summary of variables
- **num_var_name**: array of column names of numerical type variables
- **char_var_name**: array of column names of categorical type variables
- **sparse_var_name**: array of column names where population concentration at a class or value is more than the sparsity threshold

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris
data$Species <- as.character(data$Species)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)

univariate_list <- univariate(base = data,target = "Y",threshold = 0.95)
univariate_list$univar_table
univariate_list$num_var_name
univariate_list$char_var_name
univariate_list$sparse_var_name

vif_filter

Removing multicollinearity from a model using vif test

Description

The function takes a dataset with the starting variables and target only. The vif is calculated and if the maximum vif value is more than the threshold, the variable is dropped from the model and the vif’s are recomputed. These steps of computing vif and dropping variable keep iterating till the maximum vif value is less than or equal to the threshold.

Usage

vif_filter(base, target, threshold = 2)

Arguments

- **base**: input dataframe with set of final variables only along with target
- **target**: column / field name for the target variable to be passed as string (must be 0/1 type)
- **threshold**: threshold value for vif (default value is 2)
An object of class "vif_filter" is a list containing the following components:

- `vif_table`: vif table post vif filtering
- `model`: the model used for vif calculation
- `retain_var_list`: variables remaining in the model post vif filter as an array
- `dropped_var_list`: variables dropped from the model in vif filter step
- `threshold`: threshold

Author(s)

Arya Poddar <aryapoddar290990@gmail.com>

Examples

data <- iris
suppressWarnings(RNGversion('3.5.0'))
set.seed(11)
data$Y <- sample(0:1,size=nrow(data),replace=TRUE)
vif_data_list <- vif_filter(base = data,target = "Y")
vif_data_list$vif_table
vif_data_list$model
vif_data_list$retain_var_list
vif_data_list$dropped_var_list
vif_data_list$threshold
# Index

- cat_new_class, 3
- categorical_iv, 2
- club_cat_class, 4
- cv_filter, 5
- cv_table, 6
- cv_test, 7
- dtree_split_val, 8
- dtree_trend_iv, 8
- fn_conf_mat, 9
- fn_cross_index, 11
- fn_error, 12
- fn_mode, 13
- fn_target, 13
- gini_table, 14
- gradient_boosting_parameters, 15
- iv_filter, 17
- iv_table, 18
- missing_val, 19
- num_to_cat, 20
- others_class, 21
- random_forest_parameters, 22
- sampling, 24
- scalling, 25
- scoring, 26
- support_vector_parameters, 27
- univariate, 28
- vif_filter, 29