Package ‘creditmodel’

January 7, 2022

Version 1.3.1
Date 2021-12-15
Title Toolkit for Credit Modeling, Analysis and Visualization
Maintainer Dongping Fan <fdp@pku.edu.cn>

Description Provides a highly efficient R tool suite for Credit Modeling, Analysis and Visualization. Contains infrastructure functionalities such as data exploration and preparation, missing values treatment, outliers treatment, variable derivation, variable selection, dimensionality reduction, grid search for hyper parameters, data mining and visualization, model evaluation, strategy analysis etc. This package is designed to make the development of binary classification models (machine learning based models as well as credit scorecard) simpler and faster. The references including: 1 Refaat, M. (2011, ISBN: 9781447511199). Credit Risk Scorecard: Development and Implementation Using SAS; 2 Bezdek, James C. FCM: The fuzzy c-means clustering algorithm. Computers & Geosciences (0098-3004),<DOI:10.1016/0098-3004(84)90020-7>.

Depends R (>= 2.10)
Imports data.table, dplyr, ggplot2, foreach, doParallel, glmnet, rpart, cli, xgboost
Suggests pdp, pmml, XML, knitr, gbm, randomForest, rmarkdown

VignetteBuilder knitr
Encoding UTF-8
ByteCompile yes
LazyData yes
LazyLoad yes
License AGPL-3
RoxygenNote 7.1.2

NeedsCompilation no
Author Dongping Fan [aut, cre]
Repository CRAN
Date/Publication 2022-01-07 11:32:41 UTC
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>creditmodel-package</td>
<td>5</td>
</tr>
<tr>
<td>address_variable</td>
<td>5</td>
</tr>
<tr>
<td>add_variable_process</td>
<td>6</td>
</tr>
<tr>
<td>analysis_nas</td>
<td>6</td>
</tr>
<tr>
<td>analysis_outliers</td>
<td>7</td>
</tr>
<tr>
<td>as_percent</td>
<td>8</td>
</tr>
<tr>
<td>auc_value</td>
<td>8</td>
</tr>
<tr>
<td>char_cor_vars</td>
<td>9</td>
</tr>
<tr>
<td>char_to_num</td>
<td>10</td>
</tr>
<tr>
<td>checking_data</td>
<td>11</td>
</tr>
<tr>
<td>city_varieble</td>
<td>11</td>
</tr>
<tr>
<td>city_varieble_process</td>
<td>12</td>
</tr>
<tr>
<td>cohort_table_plot</td>
<td>13</td>
</tr>
<tr>
<td>cor_heat_plot</td>
<td>13</td>
</tr>
<tr>
<td>cor_plot</td>
<td>14</td>
</tr>
<tr>
<td>cos_sim</td>
<td>15</td>
</tr>
<tr>
<td>customer_segmentation</td>
<td>15</td>
</tr>
<tr>
<td>cut_equal</td>
<td>17</td>
</tr>
<tr>
<td>cv_split</td>
<td>17</td>
</tr>
<tr>
<td>data_cleansing</td>
<td>18</td>
</tr>
<tr>
<td>data_exploration</td>
<td>20</td>
</tr>
<tr>
<td>date_cut</td>
<td>21</td>
</tr>
<tr>
<td>derived_interval</td>
<td>21</td>
</tr>
<tr>
<td>derived_partial_acf</td>
<td>22</td>
</tr>
<tr>
<td>derived_pct</td>
<td>22</td>
</tr>
<tr>
<td>derived_ts_vars</td>
<td>23</td>
</tr>
<tr>
<td>de_one_hot_encoding</td>
<td>24</td>
</tr>
<tr>
<td>de_percent</td>
<td>25</td>
</tr>
<tr>
<td>digits_num</td>
<td>25</td>
</tr>
<tr>
<td>entropy_weight</td>
<td>26</td>
</tr>
<tr>
<td>entry_rate_na</td>
<td>27</td>
</tr>
<tr>
<td>euclid_dist</td>
<td>27</td>
</tr>
<tr>
<td>eval_auc</td>
<td>28</td>
</tr>
<tr>
<td>ewm_data</td>
<td>28</td>
</tr>
<tr>
<td>fast_high_cor_filter</td>
<td>29</td>
</tr>
<tr>
<td>feature_selector</td>
<td>30</td>
</tr>
<tr>
<td>fuzzy_cluster_means</td>
<td>32</td>
</tr>
<tr>
<td>gather_data</td>
<td>33</td>
</tr>
<tr>
<td>gbm_filter</td>
<td>34</td>
</tr>
<tr>
<td>gbm_params</td>
<td>36</td>
</tr>
<tr>
<td>get_auc_ks_lambda</td>
<td>37</td>
</tr>
<tr>
<td>get_bins_table_all</td>
<td>38</td>
</tr>
<tr>
<td>get_breaks_all</td>
<td>39</td>
</tr>
<tr>
<td>get_correlation_group</td>
<td>42</td>
</tr>
<tr>
<td>get_iv_all</td>
<td>43</td>
</tr>
<tr>
<td>get_logistic_coef</td>
<td>45</td>
</tr>
<tr>
<td>R topics documented:</td>
<td>3</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>get_median</td>
<td>46</td>
</tr>
<tr>
<td>get_names</td>
<td>47</td>
</tr>
<tr>
<td>get_nas_random</td>
<td>48</td>
</tr>
<tr>
<td>get_psi_all</td>
<td>48</td>
</tr>
<tr>
<td>get_psi_all</td>
<td>50</td>
</tr>
<tr>
<td>get_psi_plots</td>
<td>52</td>
</tr>
<tr>
<td>get_score_card</td>
<td>54</td>
</tr>
<tr>
<td>get_shadow_nas</td>
<td>56</td>
</tr>
<tr>
<td>get_sim_sign_lambda</td>
<td>56</td>
</tr>
<tr>
<td>get_tree_breaks</td>
<td>57</td>
</tr>
<tr>
<td>get_x_list</td>
<td>58</td>
</tr>
<tr>
<td>high_cor_selector</td>
<td>59</td>
</tr>
<tr>
<td>is_date</td>
<td>59</td>
</tr>
<tr>
<td>knn_nas_imp</td>
<td>60</td>
</tr>
<tr>
<td>ks_table</td>
<td>61</td>
</tr>
<tr>
<td>ks_value</td>
<td>63</td>
</tr>
<tr>
<td>lasso_filter</td>
<td>63</td>
</tr>
<tr>
<td>lendingclub</td>
<td>65</td>
</tr>
<tr>
<td>lift_value</td>
<td>66</td>
</tr>
<tr>
<td>local_outlier_factor</td>
<td>67</td>
</tr>
<tr>
<td>log_trans</td>
<td>67</td>
</tr>
<tr>
<td>loop_function</td>
<td>68</td>
</tr>
<tr>
<td>love_color</td>
<td>69</td>
</tr>
<tr>
<td>low_variance_filter</td>
<td>70</td>
</tr>
<tr>
<td>lr_params</td>
<td>70</td>
</tr>
<tr>
<td>lr_vif</td>
<td>73</td>
</tr>
<tr>
<td>max_min_norm</td>
<td>73</td>
</tr>
<tr>
<td>merge_category</td>
<td>74</td>
</tr>
<tr>
<td>min_max_norm</td>
<td>75</td>
</tr>
<tr>
<td>model_result_plot</td>
<td>75</td>
</tr>
<tr>
<td>multi_grid</td>
<td>78</td>
</tr>
<tr>
<td>multi_left_join</td>
<td>79</td>
</tr>
<tr>
<td>null_blank_na</td>
<td>80</td>
</tr>
<tr>
<td>n_char</td>
<td>81</td>
</tr>
<tr>
<td>one_hot_encoding</td>
<td>81</td>
</tr>
<tr>
<td>outliers_detection</td>
<td>82</td>
</tr>
<tr>
<td>partial_dependence_plot</td>
<td>83</td>
</tr>
<tr>
<td>PCA_reduce</td>
<td>84</td>
</tr>
<tr>
<td>plot_colors</td>
<td>85</td>
</tr>
<tr>
<td>plot_oout_perf</td>
<td>85</td>
</tr>
<tr>
<td>plot_table</td>
<td>87</td>
</tr>
<tr>
<td>plot_theme</td>
<td>88</td>
</tr>
<tr>
<td>pred_score</td>
<td>89</td>
</tr>
<tr>
<td>process_nas</td>
<td>90</td>
</tr>
<tr>
<td>process_outliers</td>
<td>92</td>
</tr>
<tr>
<td>psi_iv_filter</td>
<td>93</td>
</tr>
<tr>
<td>p_ij</td>
<td>95</td>
</tr>
<tr>
<td>p_to_score</td>
<td>96</td>
</tr>
</tbody>
</table>
R topics documented:

- quick_as_df ................................................................. 97
- ranking_percent_proc ................................................. 97
- read_data ................................................................. 99
- reduce_high_cor_filter .............................................. 100
- remove_duplicated ..................................................... 100
- replace_value ............................................................ 101
- require_packages ...................................................... 102
- re_code ................................................................. 103
- re_name ................................................................. 103
- rf_params ................................................................. 104
- rowAny ................................................................. 105
- save_data ............................................................... 106
- score_transfer .......................................................... 107
- select_best_class ..................................................... 108
- sim_str ................................................................. 111
- split_bins .............................................................. 111
- split_bins_all .......................................................... 112
- sql_hive_text_parse .................................................. 114
- start_parallel_computing ......................................... 115
- stop_parallel_computing ............................................ 116
- str_match ............................................................... 117
- sum_table ............................................................... 117
- term_tfidf ............................................................... 118
- time_series_proc ..................................................... 119
- time_transfer ........................................................ 119
- time_variable ........................................................ 120
- time_vars_process ................................................... 121
- tnr_value ............................................................... 121
- training_model ........................................................ 122
- train_lr ................................................................. 125
- train_test_split ....................................................... 126
- train_xgb .............................................................. 127
- UCI_creditCard ........................................................ 128
- variable_process ...................................................... 130
- var_group_proc ........................................................ 130
- woe_trans_all .......................................................... 131
- xgb_data ............................................................... 133
- xgb_filter ............................................................. 133
- xgb_params ............................................................ 135
- %alike% .............................................................. 137
- %islike% .............................................................. 137

Index ................................................................. 138
creditmodel-package

creditmodel provides a highly efficient R tool suite for Credit Modeling, Analysis and Visualization. Contains infrastructure functionalities such as data exploration and preparation, missing values treatment, outliers treatment, variable derivation, variable selection, dimensionality reduction, grid search for hyper parameters, data mining and visualization, model evaluation, strategy analysis etc. This package is designed to make the development of binary classification models (machine learning based models as well as credit scorecard) simpler and faster.

Details

It has three main goals:

• creditmodel is a free and open source automated modeling R package designed to help model developers improve model development efficiency and enable many people with no background in data science to complete the modeling work in a short time. Let them focus more on the problem itself and allocate more time to decision-making.

• creditmodel covers various tools such as data preprocessing, variable processing/derivation, variable screening/dimensionality reduction, modeling, data analysis, data visualization, model evaluation, strategy analysis, etc. It is a set of customized "core" tool kit for model developers.

• ‘creditmodel’ is suitable for machine learning automated modeling of classification targets, and is more suitable for the risk and marketing data of financial credit, e-commerce, and insurance with relatively high noise and low information content.

To learn more about creditmodel, start with the WeChat Platform: hansenmode

Author(s)

Maintainer: Dongping Fan <fdp@pku.edu.cn>

address_varieble

Description

This function is not intended to be used by end user.

Usage

address_varieble(
  df,
  address_cols = NULL,
  address_pattern = NULL,
  parallel = TRUE
)
Arguments

df: A data.frame.
def:

daddress_cols: Variables of address,
def:
daddress_pattern: Regular expressions, used to match address variable names.
def:
dparallel: Logical, parallel computing. Default is TRUE.
def:

add_variable_process

Description

This function is not intended to be used by end user.

Usage

add_variable_process(add)

Arguments

add: A data.frame contained address variables.

analysis_nas

Description

# analysis_nas is for understanding the reason for missing data and understand distribution of
# missing data so we can categorise it as:
# • missing completely at random(MCAR)
# • missing at random(MAR), or
# • missing not at random, also known as IM.

Usage

analysis_nas(
  dat,
  class_var = FALSE,
  nas_rate = NULL,
  na_vars = NULL,
  mat_nas_shadow = NULL,
  dt_nas_random = NULL,
  ...
)
### Analysis of Outliers

**Arguments**

- **dat**: A data.frame with independent variables and target variable.
- **class_var**: Logical, nas analysis of the nominal variables. Default is TRUE.
- **nas_rate**: A list contains nas rate of each variable.
- **na_vars**: Names of variables which contain nas.
- **mat_nas_shadow**: A shadow matrix of variables which contain nas.
- **dt_nas_random**: A data.frame with random nas imputation.
- **...**: Other parameters.

**Value**

A data.frame with outliers analysis for each variable.

---

### Outliers Analysis

**Description**

`#' analysis_outliers is the function for outliers analysis.`

**Usage**

```r
analysis_outliers(dat, target, x, lof = NULL)
```

**Arguments**

- **dat**: A data.frame with independent variables and target variable.
- **target**: The name of target variable.
- **x**: The name of variable to process.
- **lof**: Outliers of each variable detected by `outliers_detection`.

**Value**

A data.frame with outliers analysis for each variable.
as_percent  Percent Format

Description

as_percent is a small function for making percent format.

Usage

as_percent(x, digits = 2)

Arguments

x  A numeric vector or list.
digits  Number of digits. Default: 2.

Value

x with percent format.

Examples

as_percent(0.2363, digits = 2)
as_percent(1)

auc_value  auc_value auc_value is for get best lambda required in lasso_filter.

This function required in lasso_filter

Description

auc_value auc_value auc_value is for get best lambda required in lasso_filter. This function required in lasso_filter

Usage

auc_value(target, prob)

Arguments

target  Vector of target.
prob  A list of predict probability or score.

Value

Lambda value
char_cor_vars

Cramer’s V matrix between categorical variables.

Description

char_cor_vars is function for calculating Cramer’s V matrix between categorical variables. char_cor
is function for calculating the correlation coefficient between variables by cremers ’V

Usage

char_cor_vars(dat, x)

char_cor(dat, x_list = NULL, ex_cols = "date$", parallel = FALSE, note = FALSE)

Arguments

dat A data frame.
x The name of variable to process.
x_list Names of independent variables.
ex_cols A list of excluded variables. Regular expressions can also be used to match
variable names. Default is NULL.
parallel Logical, parallel computing. Default is FALSE.
note Logical. Outputs info. Default is TRUE.

Value

A list contains correlation index of x with other variables in dat.

Examples

## Not run:
char_x_list = get_names(dat = UCICreditCard,
types = c('factor', 'character'),
ex_cols = "ID$|date$|default.payment.next.month$", get_ex = FALSE)
char_cor(dat = UCICreditCard[char_x_list])

## End(Not run)
char_to_num

Description

char_to_num is for transferring character variables which are actually numerical numbers containing strings to numeric.

Usage

char_to_num(
  dat,
  char_list = NULL,
  m = 0,
  p = 0.5,
  note = FALSE,
  ex_cols = NULL
)

Arguments

dat        A data frame
char_list  The list of characteristic variables that need to merge categories, Default is NULL. In case of NULL, merge categories for all variables of string type.
m         The minimum number of categories.
p         The max percent of categories.
note      Logical, outputs info. Default is TRUE.
ex_cols   A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.

Value

A data.frame

Examples

dat_sub = lendingclub[c('dti_joint','emp_length')]
str(dat_sub)
#variables that are converted to numbers containing strings
dat_sub = char_to_num(dat_sub)
str(dat_sub)
**checking_data**

**Description**

Checking data checking dat before processing.

**Usage**

```r
checking_data(
  dat = NULL,
  target = NULL,
  occur_time = NULL,
  note = FALSE,
  pos_flag = NULL
)
```

**Arguments**

- `dat`: A data.frame with independent variables and target variable.
- `target`: The name of target variable. Default is NULL.
- `occur_time`: The name of the variable that represents the time at which each observation takes place.
- `note`: Logical. Outputs info. Default is TRUE.
- `pos_flag`: The value of positive class of target variable, default: "1".

**Value**

data.frame

**Examples**

```r
dat = checking_data(dat = UCICreditCard, target = "default.payment.next.month")
```

---

**city_varieble**

**Description**

This function is used for city variables derivation.
Usage

city_varieble(
  df = df,
  city_cols = NULL,
  city_pattern = NULL,
  city_class = city_class,
  parallel = TRUE
)

Arguments

df A data.frame.

Arguments

city_cols Variables of city,

city_pattern Regular expressions, used to match city variable names. Default is "city$".

city_class Class or levels of cities.

Description

This function is not intended to be used by end user.

Usage

city_varieble_process(df_city, x, city_class)

Arguments

df_city A data.frame.

Arguments

gx Variables of city,

city_class Class or levels of cities.
cohort_table_plot  

cohort_table_plot is for plotting cohort(vintage) analysis table.

Description

This function is not intended to be used by end user.

Usage

cohort_table_plot(cohort_dat)

cohort_plot(cohort_dat)

Arguments

cohort_dat  A data.frame generated by cohort_analysis.

cor_heat_plot  

Correlation Heat Plot

Description

cor_heat_plot is for plotting correlation matrix

Usage

cor_heat_plot(
  cor_mat,
  low_color = love_color("deep_red"),
  high_color = love_color("light_cyan"),
  title = "Correlation Matrix"
)

Arguments

cor_mat  A correlation matrix.
low_color  color of the lowest correlation between variables.
high_color  color of the highest correlation between variables.
title  title of plot.
**Examples**

```r
train_test = train_test_split(UCICreditCard,
split_type = "Random", prop = 0.8,save_data = FALSE)
dat_train = train_test$train
dat_test = train_test$test
cor_mat = cor(dat_train[,8:12],use = "complete.obs")
cor_heat_plot(cor_mat)
```

---

**cor_plot**

**Correlation Plot**

**Description**

`cor_plot` is for plotting correlation matrix.

**Usage**

```r
cor_plot(
  dat,
  dir_path = tempdir(),
  x_list = NULL,
  gtitle = NULL,
  save_data = FALSE,
  plot_show = FALSE
)
```

**Arguments**

- **dat**: A data.frame with independent variables and target variable.
- **dir_path**: The path for periodically saved graphic files. Default is "./model/LR"
- **x_list**: Names of independent variables.
- **gtitle**: The title of the graph & The name for periodically saved graphic file. Default is "_correlation_of_variables".
- **save_data**: Logical, save results in locally specified folder. Default is TRUE
- **plot_show**: Logical, show graph in current graphic device.

**Examples**

```r
train_test = train_test_split(UCICreditCard,
split_type = "Random", prop = 0.8,save_data = FALSE)
dat_train = train_test$train
dat_test = train_test$test
cor_plot(dat_train[,8:12],plot_show = TRUE)
```
cos_sim

Description

This function is not intended to be used by end user.

Usage

cos_sim(x, y, cos_margin = 1)

Arguments

x  A list of numbers
y  A list of numbers
cos_margin  Margin of matrix, 1 for rows and 2 for cols, Default is 1.

Value

A number of cosin similarity

customer_segmentation  Customer Segmentation

Description

customer_segmentation is a function for clustering and find the best segment variable.

Usage

customer_segmentation(
    dat,
    x_list = NULL,
    ex_cols = NULL,
    cluster_control = list(meth = "Kmeans", kc = 2, nstart = 1, epsm = 1e-06, sf = 2,
        max_iter = 100),
    tree_control = list(cv_folds = 5, maxdepth = kc + 1, minbucket = nrow(dat)/(kc + 1)),
    save_data = FALSE,
    file_name = NULL,
    dir_path = tempdir() )
**Arguments**

- **dat**
  - A data.frame contained only predict variables.

- **x_list**
  - A list of x variables.

- **ex_cols**
  - A list of excluded variables. Default is NULL.

- **cluster_control**
  - A list controls cluster. kc is the number of cluster center (default is 2), nstart is the number of random groups (default is 1), max_iter max iteration number(default is 100).
    - **meth** Method of clustering. Provides two methods, "Kmeans" and "FCM(Fuzzy Cluster Means)" (default is "Kmeans").
    - **kc** Number of cluster center (default is 2).
    - **nstart** Number of random groups (default is 1).
    - **max_iter** Max iteration number (default is 100).

- **tree_control**
  - A list of controls for decision tree to find the best segment variable.
    - **cv_folds** Number of cross-validations (default is 5).
    - **maxdepth** Maximum depth of a tree (default is kc +1).
    - **minbucket** Minimum percent of observations in any terminal <leaf> node (default is nrow(dat) / (kc + 1)).

- **save_data** Logical. If TRUE, save outliers analysis file to the specified folder at dir_path.

- **file_name** The name for periodically saved segmentation file. Default is NULL.

- **dir_path** The path for periodically saved segmentation file.

**Value**

A "data.frame" object contains cluster results.

**References**


**Examples**

```r
clust = customer_segmentation(dat = lendingclub[1:10000,20:30],
  x_list = NULL, ex_cols = "id\$\|loan_status",
  cluster_control = list(meth = "FCM", kc = 2), save_data = FALSE,
  tree_control = list(minbucket = round(nrow(lendingclub) / 10)),
  file_name = NULL, dir_path = tempdir())
```
cut_equal

Generating Initial Equal Size Sample Bins

Description
cut_equal is used to generate initial breaks for equal frequency binning.

Usage
cut_equal(dat_x, g = 10, sp_values = NULL, cut_bin = "equal_depth")

Arguments
dat_x A vector of an variable x.
g numeric, number of initial bins for equal_bins.
sp_values a list of special value. Default: list(-1, "missing")
cut_bin A string, 'equal_depth' or 'equal_width', default is 'equal_depth'.

See Also
generate_breaks, generate_breaks_all, generate_tree_breaks

Examples
#equal sample size breaks
equ_breaks = cut_equal(dat = UCICreditCard[, "PAY_AMT2"], g = 10)

cv_split

Stratified Folds

Description
this function creates stratified folds for cross validation.

Usage
cv_split(dat, k = 5, occur_time = NULL, seed = 46)

Arguments
dat A data.frame.
k k is an integer specifying the number of folds.
occur_time time variable for creating OOT folds. Default is NULL.
seed A seed. Default is 46.
Value

a list of indices

Examples

```r
sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
```

Description

The `data_cleansing` function is a simpler wrapper for data cleaning functions, such as delete variables that values are all NAs; checking dat and target format, delete low variance variables replace null or NULL or blank with NA; encode variables which NAs & miss value rate is more than 95 encode variables which unique value rate is more than 95 merge categories of character variables that is more than 10; transfer time variables to dateformation; remove duplicated observations; process outliers; process NAs.

Usage

```r
data_cleansing(
  dat,
  target = NULL,
  obs_id = NULL,
  occur_time = NULL,
  pos_flag = NULL,
  x_list = NULL,
  ex_cols = NULL,
  miss_values = NULL,
  remove_dup = TRUE,
  outlier_proc = TRUE,
  missing_proc = "median",
  low_var = 0.999,
  missing_rate = 0.999,
  merge_cat = TRUE,
  note = TRUE,
  parallel = FALSE,
  save_data = FALSE,
  file_name = NULL,
  dir_path = tempdir()
)
```
**Arguments**

- **dat**
  A data frame with x and target.
- **target**
  The name of target variable.
- **obs_id**
  The name of ID of observations. Default is NULL.
- **occur_time**
  The name of occur time of observations. Default is NULL.
- **pos_flag**
  The value of positive class of target variable, default: "1".
- **x_list**
  A list of x variables.
- **ex_cols**
  A list of excluded variables. Default is NULL.
- **miss_values**
  Other extreme value might be used to represent missing values, e.g: -9999, -9998. These miss_values will be encoded to -1 or "missing".
- **remove_dup**
  Logical, if TRUE, remove the duplicated observations.
- **outlier_proc**
  Logical, process outliers or not. Default is TRUE.
- **missing_proc**
  If logical, process missing values or not. If "median", then Nas imputation with k neighbors median. If "avg_dist", the distance weighted average method is applied to determine the NAs imputation with k neighbors. If "default", assigning the missing values to -1 or "missing", otherwise, processing the missing values according to the results of missing analysis.
- **low_var**
  The maximum percent of unique values (including NAs) for filtering low variance variables.
- **missing_rate**
  The maximum percent of missing values for recoding values to missing and non_missing.
- **merge_cat**
  The minimum number of categories for merging categories of character variables.
- **note**
  Logical. Outputs info. Default is TRUE.
- **parallel**
  Logical, parallel computing or not. Default is FALSE.
- **save_data**
  Logical, save the result or not. Default is FALSE.
- **file_name**
  The name for periodically saved data file. Default is NULL.
- **dir_path**
  The path for periodically saved data file. Default is tempdir().

**Value**

A preprocessed data.frame

**See Also**

remove_duplicated, null_blank_na, entry_rate_na, low_variance_filter, process_nas, process_outliers
data_exploration

Data Exploration

Description

The data_exploration includes both univariate and bivariate analysis and ranges from univariate statistics and frequency distributions, to correlations, cross-tabulation and characteristic analysis.

Usage

data_exploration(
  dat,
  save_data = FALSE,
  file_name = NULL,
  dir_path = tempdir(),
  note = FALSE
)

Arguments

dat A data.frame with x and target.
save_data Logical. If TRUE, save files to the specified folder at dir_path
file_name The file name for periodically saved outliers analysis file. Default is NULL.
dir_path The path for periodically saved outliers analysis file. Default is tempdir().
note Logical, outputs info. Default is TRUE.

Value

A list contains both category and numeric variable analysis.

Examples

data_ex = data_exploration(dat = UCICreditCard[1:1000,])
**date_cut**

*Date Time Cut Point*

date_cut is a small function to get date point.

**Usage**

date_cut(dat_time, pct = 0.7, g = 100)

**Arguments**

dat_time  
time vectors.
pct  
the percent of cutting. Default: 0.7.
g  
Number of cuts.

**Value**

A Date.

**Examples**

date_cut(dat_time = lendingclub$issue_d, pct = 0.8)
#"2018-08-01"

**derived_interval**

derived_interval

**Description**

This function is not intended to be used by end user.

**Usage**

derived_interval(dat_s, interval_type = c("cnt_interval", "time_interval"))

**Arguments**

dat_s  
A data.frame contained only predict variables.
interval_type  
Available of c("cnt_interval", "time_interval")
derived_partial_acf

Description
This function is not intended to be used by end user.

Usage
derived_partial_acf(dat_s)

Arguments
dat_s A data.frame

derived_pct

Description
This function is not intended to be used by end user.

Usage
derived_pct(dat_s, pct_type = "total_pct")

Arguments
dat_s A data.frame contained only predict variables.
pct_type Available of "total_pct"
**derived_ts_vars**

**Derivation of Behavioral Variables**

**Description**

This function is used for derivating behavioral variables and is not intended to be used by end user.

**Usage**

```r
derived_ts_vars(
  dat,
  grx = NULL,
  td = NULL,
  ID = NULL,
  ex_cols = NULL,
  x_list = NULL,
  der = c("cvs", "sums", "means", "maxs", "max_mins", "time_intervals",
           "cnt_intervals", "total_pcts", "cum_pcts", "partial_acfs"),
  parallel = TRUE,
  note = TRUE
)
```

```r
derived_ts(
  dat,
  grx_x = NULL,
  x_list = NULL,
  td = NULL,
  ID = NULL,
  ex_cols = NULL,
  der = c("cvs", "sums", "means", "maxs", "max_mins", "time_intervals",
           "cnt_intervals", "total_pcts", "cum_pcts", "partial_acfs")
)
```

**Arguments**

- **dat** A data.frame contained only predict variables.
- **grx** Regular expressions used to match variable names.
- **td** Number of variables to derivate.
- **ID** The name of ID of observations or key variable of data. Default is NULL.
- **ex_cols** A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
- **x_list** Names of independent variables.
- **der** Variables to derivate
- **parallel** Logical, parallel computing. Default is FALSE.
- **note** Logical, outputs info. Default is TRUE.
- **grx_x** Regular expression used to match a group of variable names.
Details

The key to creating a good model is not the power of a specific modelling technique, but the breadth and depth of derived variables that represent a higher level of knowledge about the phenomena under examination.

---

### de_one_hot_encoding  
**Recovery One-Hot Encoding**

**Description**

de_one_hot_encoding is for one-hot encoding recovery processing.

**Usage**

de_one_hot_encoding(dat_one_hot, cat_vars = NULL, na_act = TRUE, note = FALSE)

**Arguments**

dat_one_hot: A data frame with the one-hot encoding variables.
cat_vars: Variables to be recovery processed, default is NULL, if NULL, find these variables through regular expressions.
na_act: Logical. If true, the missing value is assigned as "missing", if FALSE missing value is omitted, the default is TRUE.
note: Logical. Outputs info. Default is TRUE.

**Value**

A data frame with the one-hot encoding recovery character variables.

**See Also**

one_hot_encoding

**Examples**

```r
# one hot encoding
dat1 = one_hot_encoding(dat = UCICreditCard, 
cat_vars = c("SEX", "MARRIAGE"), 
merge_cat = TRUE, na_act = TRUE)
# de one hot encoding
dat2 = de_one_hot_encoding(dat_one_hot = dat1, 
cat_vars = c("SEX", "MARRIAGE"), 
na_act = FALSE)
```
Description

de_percent is a small function for recovering percent format.

Usage

de_percent(x, digits = 2)

Arguments

- x: Character with percent format.
- digits: Number of digits. Default: 2.

Value

x without percent format.

Examples

de_percent("24%")

digits_num

Number of digits

Description

digits_num is for calculating optimal digits number for numeric variables.

Usage

digits_num(dat_x)

Arguments

- dat_x: A numeric variable.

Value

A number of digits
entropy_weight

Examples

```r
## Not run:
digits_num(lendingclub[,"dti")
# 7

## End(Not run)
```

## Not run:
digits_num(lendingclub[,"dti")
# 7

## End(Not run)

### Description

entropy_weight is for calculating Entropy Weight.

### Usage

```r
entropy_weight(dat, pos_vars, neg_vars)
```

### Arguments

- **dat**: A data.frame with independent variables.
- **pos_vars**: Names or index of positive direction variables, the bigger the better.
- **neg_vars**: Names or index of negative direction variables, the smaller the better.

### Details

**Step1** Raw data normalization  
**Step2** Find out the total amount of contributions of all samples to the index Xj  
**Step3** Each element of the step generated matrix is transformed into the product of each element and the LN (element), and the information entropy is calculated.  
**Step4** Calculate redundancy.  
**Step5** Calculate the weight of each index.

### Value

A data.frame with weights of each variable.

### Examples

```r
entropy_weight(dat = ewm_data,
               pos_vars = c(6,8,9,10),
               neg_vars = c(7,11))
```
entry_rate_na  

**Description**

`entry_rate_na` is the function to recode variables with missing values up to a certain percentage with missing and non_missing.

**Usage**

`entry_rate_na(dat, nr = 0.98, note = FALSE)`

**Arguments**

- `dat`: A data frame with x and target.
- `nr`: The maximum percent of NAs.
- `note`: Logical. Outputs info. Default is TRUE.

**Value**

A data.frame

**Examples**

`datss = entry_rate_na(dat = lendingclub[1:1000, ], nr = 0.98)`

---

euclid_dist

**Description**

This function is not intended to be used by end user.

**Usage**

`euclid_dist(x, y, cos_margin = 1)`

**Arguments**

- `x`: A list
- `y`: A list
- `cos_margin`: rows or cols
**Functions of xgboost feval**

**Description**

`eval_auc`, `eval_ks`, `eval_lift`, `eval_tnr` is for getting best params of xgboost.

**Usage**

```r
eval_auc(preds, dtrain)
eval_ks(preds, dtrain)
eval_tnr(preds, dtrain)
eval_lift(preds, dtrain)
```

**Arguments**

- `preds` A list of predict probability or score.
- `dtrain` Matrix of x predictors.

**Value**

List of best value

---

**ewm_data**

**Entropy Weight Method Data**

**Description**

This data is for Entropy Weight Method examples.

**Format**

A data frame with 10 rows and 13 variables.
**Description**

*fast_high_cor_filter* In a highly correlated variable group, select the variable with the highest IV. *high_cor_filter* In a highly correlated variable group, select the variable with the highest IV.

**Usage**

```r
fast_high_cor_filter(
  dat,
  p = 0.95,
  x_list = NULL,
  com_list = NULL,
  ex_cols = NULL,
  save_data = FALSE,
  cor_class = TRUE,
  vars_name = TRUE,
  parallel = FALSE,
  note = FALSE,
  file_name = NULL,
  dir_path = tempdir(),
  ...
)
```

```r
high_cor_filter(
  dat,
  com_list = NULL,
  x_list = NULL,
  ex_cols = NULL,
  onehot = TRUE,
  parallel = FALSE,
  p = 0.7,
  file_name = NULL,
  dir_path = tempdir(),
  save_data = FALSE,
  note = FALSE,
  ...
)
```

**Arguments**

- `dat` A data.frame with independent variables.
- `p` Threshold of correlation between features. Default is 0.95.
- `x_list` Names of independent variables.
com_list  A data.frame with important values of each variable. eg : IV_list
ex_cols  A list of excluded variables. Regular expressions can also be used to match
          variable names. Default is NULL.
save_data  Logical, save results in locally specified folder. Default is FALSE.
cor_class  Calculate catagery variables’s correlation matrix. Default is FALSE.
vars_name  Logical, output a list of filtered variables or table with detailed compared value
           of each variable. Default is TRUE.
parallel  Logical, parallel computing. Default is FALSE.
note  Logical. Outputs info. Default is TRUE.
file_name  The name for periodically saved results files. Default is “Feature_selected_COR”.
dir_path  The path for periodically saved results files. Default is “./variable”.
...  Additional parameters.
onehot  one-hot-encoding independent variables.

Value  A list of selected variables.

See Also
get_correlation_group, high_cor_selector, char_cor_vars

Examples

# calculate iv for each variable.
iv_list = feature_selector(dat_train = UCICreditCard[1:1000,], dat_test = NULL,
target = "default.payment.next.month",
occur_time = "apply_date",
filter = c("IV"), cv_folds = 1, iv_cp = 0.01,
ex_cols = "ID$|date$|default.payment.next.month$",
save_data = FALSE, vars_name = FALSE)

fast_high_cor_filter(dat = UCICreditCard[1:1000,],
com_list = iv_list, save_data = FALSE,
ex_cols = "ID$|date$|default.payment.next.month$",
p = 0.9, cor_class = FALSE ,var_name = FALSE)
Usage

```r
feature_selector(
  dat_train,
  dat_test = NULL,
  x_list = NULL,
  target = NULL,
  pos_flag = NULL,
  occur_time = NULL,
  ex_cols = NULL,
  filter = c("IV", "PSI", "XGB", "COR"),
  cv_folds = 1,
  iv_cp = 0.01,
  psi_cp = 0.5,
  xgb_cp = 0,
  cor_cp = 0.98,
  breaks_list = NULL,
  hopper = FALSE,
  vars_name = TRUE,
  parallel = FALSE,
  note = TRUE,
  seed = 46,
  save_data = FALSE,
  file_name = NULL,
  dir_path = tempdir(),
  ...
)
```

Arguments

- `dat_train`: A data.frame with independent variables and target variable.
- `dat_test`: A data.frame of test data. Default is NULL.
- `x_list`: Names of independent variables.
- `target`: The name of target variable.
- `pos_flag`: The value of positive class of target variable, default: "1".
- `occur_time`: The name of the variable that represents the time at which each observation takes place.
- `ex_cols`: A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
- `filter`: The methods for selecting important and stable variables.
- `cv_folds`: Number of cross-validations. Default: 5.
- `iv_cp`: The minimum threshold of IV. 0 < iv_i ; 0.01 to 0.1 usually work. Default: 0.02
- `psi_cp`: The maximum threshold of PSI. 0 <= psi_i <=1; 0.05 to 0.2 usually work. Default: 0.1
- `xgb_cp`: Threshold of XGB feature’s Gain. 0 <= xgb_cg <=1. Default is 1/number of independent variables.
### fuzzy_cluster_means

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cor_cp</td>
<td>Threshold of correlation between features. 0 &lt;= cor_cp &lt;=1; 0.7 to 0.98 usually work. Default is 0.98.</td>
</tr>
<tr>
<td>breaks_list</td>
<td>A table containing a list of splitting points for each independent variable. Default is NULL.</td>
</tr>
<tr>
<td>hopper</td>
<td>Logical. Filtering screening. Default is FALSE.</td>
</tr>
<tr>
<td>vars_name</td>
<td>Logical, output a list of filtered variables or table with detailed IV and PSI value of each variable. Default is FALSE.</td>
</tr>
<tr>
<td>parallel</td>
<td>Logical, parallel computing. Default is FALSE.</td>
</tr>
<tr>
<td>note</td>
<td>Logical. Outputs info. Default is TRUE.</td>
</tr>
<tr>
<td>seed</td>
<td>Random number seed. Default is 46.</td>
</tr>
<tr>
<td>save_data</td>
<td>Logical, save results in locally specified folder. Default is FALSE.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name for periodically saved results files. Default is &quot;select_vars&quot;.</td>
</tr>
<tr>
<td>dir_path</td>
<td>The path for periodically saved results files. Default is &quot;/variable&quot;</td>
</tr>
<tr>
<td>...</td>
<td>Other parameters.</td>
</tr>
</tbody>
</table>

**Value**

A list of selected features

**See Also**

psi_iv_filter, xgb_filter, gbm_filter

**Examples**

```r
feature_selector(dat_train = UCICreditCard[1:1000, c(2, 8:12, 26)],
                 dat_test = NULL, target = "default.payment.next.month",
                 occur_time = "apply_date", filter = c("IV", "PSI"),
                 cv_folds = 1, iv_cp = 0.01, psi_cp = 0.1, xgb_cp = 0, cor_cp = 0.98,
                 vars_name = FALSE, note = FALSE)
```

---

**Description**

This function is used for Fuzzy Clustering.
Usage

fuzzy_cluster_means(
    dat,
    kc = 2,
    sf = 2,
    nstart = 1,
    max_iter = 100,
    epsm = 1e-06
)

gather_data(dat, x_list = NULL, ID = NULL, FUN = sum_x)

Arguments

gather_data

Arguments

gather_data

References

The key to creating a good model is not the power of a specific modelling technique, but the breadth and depth of derived variables that represent a higher level of knowledge about the phenomena under examination.

Examples

dat = data.frame(id = c(1,1,1,2,2,3,3,3,4,4,4,4,4,5,5,6,7,7,8,8,9,9,10,10,11,11,11,11,11,11),
    terms = c('a','b','c','a','c','d','d','a','c','b','c','d','d','a',
               'e','f','b','c','f','b','c','h','h','i','c','d','g','k','k',
               'a','e','f','b','c','h','h','i','c','d','g','k','k'),
    time = c(8,3,1,9,6,1,4,9,1,3,4,8,2,7,1,
            3,4,1,8,7,2,5,7,8,2,1,5,7,2,7,3))

gather_data(dat = dat, x_list = "time", ID = 'id', FUN = sum_x)
Arguments

dat A data.frame with independent variables and target variable.
target The name of target variable.
x_list Names of independent variables.
ex_cols A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
pos_flag The value of positive class of target variable, default: "1".
GBM.params Parameters of GBM.
cores_num The number of CPU cores to use.
vars_name Logical, output a list of filtered variables or table with detailed IV and PSI value of each variable. Default is TRUE.
note Logical, outputs info. Default is TRUE.
save_data Logical, save results results in locally specified folder. Default is FALSE.
file_name The name for periodically saved results files. Default is "Feature_importance_GBDT".
dir_path The path for periodically saved results files. Default is "/variable".
seed Random number seed. Default is 46.
... Other parameters to pass to gbdt_params.

Value
Selected variables.

See Also

 psi_iv_filter, xgb_filter, feature_selector

Examples

GBM.params = gbm_params(n.trees = 2, interaction.depth = 2, shrinkage = 0.1,
                        bag.fraction = 1, train.fraction = 1,
                        n.minobsinnode = 30,
                        cv.folds = 2)

## Not run:
features = gbm_filter(dat = UCICreditCard[1:1000, c(8:12, 26)],
                       target = "default.payment.next.month",
                       occur_time = "apply_date",
                       GBM.params = GBM.params,
                       vars_name = FALSE)

## End(Not run)
### gbm_params

**GBM Parameters**

**Description**

gbm_params is the list of parameters to train a GBM using in `training_model`.

**Usage**

```r
gbm_params(
  n.trees = 1000,
  interaction.depth = 6,
  shrinkage = 0.01,
  bag.fraction = 0.5,
  train.fraction = 0.7,
  n.minobsinnode = 30,
  cv.folds = 5,
  ...
)
```

**Arguments**

- `n.trees`: Integer specifying the total number of trees to fit. This is equivalent to the number of iterations and the number of basis functions in the additive expansion. Default is 100.
- `interaction.depth`: Integer specifying the maximum depth of each tree (i.e., the highest level of variable interactions allowed). A value of 1 implies an additive model, a value of 2 implies a model with up to 2-way interactions, etc. Default is 1.
- `shrinkage`: A shrinkage parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction; 0.001 to 0.1 usually work, but a smaller learning rate typically requires more trees. Default is 0.1.
- `bag.fraction`: The fraction of the training set observations randomly selected to propose the next tree in the expansion. This introduces randomness into the model fit. If bag.fraction < 1 then running the same model twice will result in similar but different fits. gbm uses the R random number generator so set.seed can ensure that the model can be reconstructed. Preferably, the user can save the returned gbm.object using save. Default is 0.5.
- `train.fraction`: The first train.fraction * nrows(data) observations are used to fit the gbm and the remainder are used for computing out-of-sample estimates of the loss function.
- `n.minobsinnode`: Integer specifying the minimum number of observations in the terminal nodes of the trees. Note that this is the actual number of observations, not the total weight.
- `cv.folds`: Number of cross-validation folds to perform. If cv.folds > 1 then gbm, in addition to the usual fit, will perform a cross-validation, calculate an estimate of generalization error returned in cv.error.
- `...`: Other parameters...
**get_auc_ks_lambda**

### Details

See details at: gbm

### Value

A list of parameters.

### See Also

`training_model, lr_params, xgb_params, rf_params`

---

**Description**

`get_auc_ks_lambda` is for getting the best lambda required in `lasso_filter`. This function is required in `lasso_filter`.

**Usage**

```r
get_auc_ks_lambda(
  lasso_model,
  x_test,
  y_test,
  save_data = FALSE,
  plot_show = TRUE,
  file_name = NULL,
  dir_path = tempdir()
)
```

### Arguments

- **lasso_model**: A lasso model generated by glmnet.
- **x_test**: A matrix of test dataset with x.
- **y_test**: A matrix of y test dataset with y.
- **save_data**: Logical, save results in locally specified folder. Default is FALSE.
- **plot_show**: Logical, if TRUE plot the results. Default is TRUE.
- **file_name**: The name for periodically saved results files. Default is NULL.
- **dir_path**: The path for periodically saved results files.

### Value

Lambda values with max K-S and AUC.
See Also

lasso_filter, get_sim_sign_lambda

---

**get_bins_table_all**  
*Table of Binning*

**Description**

`get_bins_table` is used to generates summary information of varaibles. `get_bins_table_all` can generates bins table for all specified independent variables.

**Usage**

```r
get_bins_table_all(
  dat,
  x_list = NULL,
  target = NULL,
  pos_flag = NULL,
  dat_test = NULL,
  ex_cols = NULL,
  breaks_list = NULL,
  parallel = FALSE,
  note = FALSE,
  bins_total = TRUE,
  save_data = FALSE,
  file_name = NULL,
  dir_path = tempdir()
)

get_bins_table(
  dat,
  x,
  target = NULL,
  pos_flag = NULL,
  dat_test = NULL,
  breaks = NULL,
  breaks_list = NULL,
  bins_total = TRUE,
  note = FALSE
)
```

**Arguments**

- **dat**: A data.frame with independent variables and target variable.
- **x_list**: Names of independent variables.
- **target**: The name of target variable.
get_breaks_all

Generates Best Breaks for Binning

Description

get_breaks is for generating optimal binning for numerical and nominal variables. The get_breaks_all is a simpler wrapper for get_breaks.

Usage

get_breaks_all(
  dat,
  target = NULL,
  x_list = NULL,
  ex_cols = NULL,
  pos_flag = NULL,
  occur_time = NULL,
  parallel = FALSE,
  note = TRUE,
  bins_total = FALSE,
  save_data = FALSE,
  file_name = "bins_table",
  dir_path = "./variable",
  g = 5,
  equal_bins = TRUE,
  best = FALSE,
  target_flag = "1"
)

See Also

get_iv, get_iv_all, get_psi, get_psi_all

Examples

breaks_list = get_breaks_all(dat = UCICreditCard, x_list = names(UCICreditCard)[3:4],
  target = "default.payment.next.month", equal_bins = TRUE,best = FALSE,g=5,
  ex_cols = "ID|apply_date", save_data = FALSE)
get_bins_table_all(dat = UCICreditCard, breaks_list = breaks_list,
  target = "default.payment.next.month")
get_breaks_all

oot_pct = 0.7,
best = TRUE,
equal_bins = FALSE,
cut_bin = "equal_depth",
g = 10,
sp_values = NULL,
tree_control = list(p = 0.05, cp = 1e-06, xval = 5, maxdepth = 10),
bins_control = list(bins_num = 10, bins_pct = 0.05, b_chi = 0.05, b_odds = 0.1, b.psi = 0.05, b_or = 0.15, mono = 0.3, odds.psi = 0.2, kc = 1),
parallel = FALSE,
note = FALSE,
save_data = FALSE,
file_name = NULL,
dir_path = tempdir(),
...)

get_breaks(  
  dat,
  x,
  target = NULL,
  pos_flag = NULL,
  best = TRUE,
equal_bins = FALSE,
cut_bin = "equal_depth",
g = 10,
sp_values = NULL,
occur_time = NULL,
oot_pct = 0.7,
tree_control = NULL,
bins_control = NULL,
note = FALSE,
...)

Arguments

dat A data frame with x and target.
target The name of target variable.
x_list A list of x variables.
ex_cols A list of excluded variables. Default is NULL.
pos_flag The value of positive class of target variable, default: "1".
occur_time The name of the variable that represents the time at which each observation takes place.
oot_pct Percentage of observations retained for overtime test (especially to calculate PSI). Default is 0.7
best Logical, if TRUE, merge initial breaks to get optimal breaks for binning.
equal_bins Logical, if TRUE, equal sample size initial breaks generates. If FALSE, tree breaks generates using desison tree.
cut_bin A string, if equal_bins is TRUE, 'equal_depth' or 'equal_width', default is 'equal_depth'.
g Integer, number of initial bins for equal_bins.
sp_values A list of missing values.
tree_control the list of tree parameters.
  • p the minimum percent of observations in any terminal <leaf> node. 0 < p< 1; 0.01 to 0.1 usually work.
  • cp complexity parameter. the larger, the more conservative the algorithm will be. 0 < cp< 1 ; 0.0001 to 0.0000001 usually work.
  • xval number of cross-validations. Default: 5
  • max_depth maximum depth of a tree. Default: 10
bins_control the list of parameters.
  • bins_num The maximum number of bins. 5 to 10 usually work. Default: 10
  • bins_pct The minimum percent of observations in any bins. 0 < bins_pct < 1 ; 0.01 to 0.1 usually work. Default: 0.02
  • b.chi The minimum threshold of chi-square merge. 0 < b.chi< 1; 0.01 to 0.1 usually work. Default: 0.02
  • b.odds The minimum threshold of odds merge. 0 < b.odds < 1; 0.05 to 0.2 usually work. Default: 0.1
  • b.psi The maximum threshold of PSI in any bins. 0 < b.psi < 1 ; 0 to 0.1 usually work. Default: 0.05
  • b.or The maximum threshold of G/B index in any bins. 0 < b.or < 1 ; 0.05 to 0.3 usually work. Default: 0.15
  • odds.psi The maximum threshold of Training and Testing G/B index PSI in any bins. 0 < odds.psi < 1 ; 0.01 to 0.3 usually work. Default: 0.1
  • mono Monotonicity of all bins, the larger, the more nonmonotonic the bins will be. 0 < mono < 0.5 ; 0.2 to 0.4 usually work. Default: 0.2
  • kc number of cross-validations. 1 to 5 usually work. Default: 1
parallel Logical, parallel computing or not. Default is FALSE.
note Logical. Outputs info. Default is TRUE.
save_data Logical, save results in locally specified folder. Default is TRUE
file_name File name that save results in locally specified folder. Default is "breaks_list".
dir_path Path to save results. Default is "/variable"
... Additional parameters.
x The Name of an independent variable.

Value
A table containing a list of splitting points for each independent variable.
get_correlation_group

See Also

get_tree_breaks, cut_equal, select_best_class, select_best_breaks

Examples

# controls
tree_control = list(p = 0.02, cp = 0.000001, xval = 5, maxdepth = 10)
bins_control = list(bins_num = 10, bins_pct = 0.02, b_chi = 0.02, b_odds = 0.1,
b_psi = 0.05, b_or = 15, mono = 0.2, odds_psi = 0.1, kc = 5)

# get category variable breaks
b = get_breaks(dat = UCICreditCard[1:1000,], x = "MARRIAGE",
target = "default.payment.next.month",
occur_time = "apply_date",
sp_values = list(-1, "missing"),
tree_control = tree_control, bins_control = bins_control)

# get numeric variable breaks
b2 = get_breaks(dat = UCICreditCard[1:1000,], x = "PAY_2",
target = "default.payment.next.month",
occur_time = "apply_date",
sp_values = list(-1, "missing"),
tree_control = tree_control, bins_control = bins_control)

# get breaks of all predictive variables
b3 = get_breaks_all(dat = UCICreditCard[1:1000,], target = "default.payment.next.month",
x_list = c("MARRIAGE", "PAY_2"),
occur_time = "apply_date", ex_cols = "ID",
sp_values = list(-1, "missing"),
tree_control = tree_control, bins_control = bins_control,
save_data = FALSE)

description

get_correlation_group is function for obtaining highly correlated variable groups. select_cor_group
is function for selecting highly correlated variable group. select_cor_list is function for selecting
highly correlated variable list.

Usage

get_correlation_group(cor_mat, p = 0.8)

select_cor_group(cor_vars)

select_cor_list(cor_vars_list)
Arguments

- `cor_mat`: A correlation matrix of independent variables.
- `p`: Threshold of correlation between features. Default is 0.7.
- `cor_vars`: Correlated variables.
- `cor_vars_list`: List of correlated variable

Value

A list of selected variables.

Examples

```r
## Not run:
cor_mat = cor(UCICreditCard[8:20],
use = "complete.obs", method = "spearman")
get_correlation_group(cor_mat, p = 0.6 )

## End(Not run)
```

get_iv_all

Calculate Information Value (IV) get_iv is used to calculate Information Value (IV) of an independent variable. get_iv_all can loop through IV for all specified independent variables.

Description

Calculate Information Value (IV) get_iv is used to calculate Information Value (IV) of an independent variable. get_iv_all can loop through IV for all specified independent variables.

Usage

```r
get_iv_all(
  dat,
  x_list = NULL,
  ex_cols = NULL,
  breaks_list = NULL,
  target = NULL,
  pos_flag = NULL,
  best = TRUE,
  equal_bins = FALSE,
  tree_control = NULL,
  bins_control = NULL,
  g = 10,
  parallel = FALSE,
  note = FALSE
)
```
get_iv(
  dat,
  x,
  target = NULL,
  pos_flag = NULL,
  breaks = NULL,
  breaks_list = NULL,
  best = TRUE,
  equal_bins = FALSE,
  tree_control = NULL,
  bins_control = NULL,
  g = 10,
  note = FALSE
)

Arguments

dat  A data.frame with independent variables and target variable.

x_list  Names of independent variables.

ex_cols  A list of excluded variables. Regular expressions can also be used to match
  variable names. Default is NULL.

breaks_list  A table containing a list of splitting points for each independent variable. De-
  fault is NULL.

target  The name of target variable.

pos_flag  Value of positive class. Default is "1".

best  Logical, merge initial breaks to get optimal breaks for binning.

equal_bins  Logical, generates initial breaks for equal frequency binning.

tree_control  Parameters of using Decision Tree to segment initial breaks. See detials: get_tree_breaks

bins_control  Parameters used to control binning. See detials: select_best_class, select_best_breaks

g  Number of initial breakpoints for equal frequency binning.

parallel  Logical, parallel computing. Default is FALSE.

note  Logical, outputs info. Default is TRUE.

x  The name of an independent variable.

breaks  Splitting points for an independent variable. Default is NULL.

Details

IV Rules of Thumb for evaluating the strength a predictor
Less than 0.02:unpredictive
0.02 to 0.1:weak
0.1 to 0.3:medium
0.3+:strong

References

Information Value Statistic:Bruce Lund, Magnify Analytics Solutions, a Division of Marketing
Associates, Detroit, MI(Paper AA - 14 - 2013)
**get_logistic_coef**

get_logistic_coef is for getting logistic coefficients.

### Usage

```r
ger_logistic_coef(  
  lg_model,  
  file_name = NULL,  
  dir_path = tempdir(),  
  save_data = FALSE  
)
```

### Arguments

- **lg_model**: An object of logistic model.
- **file_name**: The name for periodically saved coefficient file. Default is "LR_coef".
- **dir_path**: The Path for periodically saved coefficient file. Default is "/model".
- **save_data**: Logical, save the result or not. Default is FALSE.

### Value

A data.frame with logistic coefficients.

**See Also**

- `get_iv`
- `get_iv_all`
- `get_psi`
- `get_psi_all`

**Examples**

```r
generate_iv_all(dat = UCICreditCard,  
x_list = names(UCICreditCard)[3:10],  
equal_bins = TRUE, best = FALSE,  
target = "default.payment.next.month",  
ex_cols = "ID|apply_date")
generate_iv(UCICreditCard, x = "PAY_3",  
equal_bins = TRUE, best = FALSE,  
target = "default.payment.next.month")
```
Examples

```r
# dataset splitting
sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,
# rename the target variable
dat = re_name(dat, "default.payment.next.month", "target")
dat = data_cleansing(dat, target = "target", obs_id = "ID",
occur_time = "apply_date", miss_values = list("", -1))
# train test splitting
train_test = train_test_split(dat, split_type = "OOT", prop = 0.7,
occur_time = "apply_date")
dat_train = train_test$train
dat_test = train_test$test
# get breaks of all predictive variables
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "EDUCATION", "PAY_3", "PAY_2")
breaks_list = get_breaks_all(dat = dat_train, target = "target",
x_list = x_list, occur_time = "apply_date", ex_cols = "ID",
save_data = FALSE, note = FALSE)
# WOE transforming
train_woe = woe_trans_all(dat = dat_train,
target = "target",
breaks_list = breaks_list,
woe_name = FALSE)
test_woe = woe_trans_all(dat = dat_test,
target = "target",
breaks_list = breaks_list,
note = FALSE)
Formula = as.formula(paste("target", paste(x_list, collapse = \'+\', sep = '\'-\')))
set.seed(46)
lr_model = glm(Formula, data = train_woe[, c("target", x_list)], family = binomial(logit))
# get LR coefficient
dt_imp_LR = get_logistic_coef(lg_model = lr_model, save_data = FALSE)
bins_table = get_bins_table_all(dat = dat_train, target = "target",
x_list = x_list, dat_test = dat_test,
breaks_list = breaks_list, note = FALSE)
# score card
LR_score_card = get_score_card(lg_model = lr_model, bins_table, target = "target")
# scoring
train_pred = dat_train[, c("ID", "apply_date", "target")]
test_pred = dat_test[, c("ID", "apply_date", "target")]
train_pred$pred_LR = score_transfer(model = lr_model,
	tbl_woe = train_woe,
save_data = TRUE)[, "score"]
test_pred$pred_LR = score_transfer(model = lr_model,
tbl_woe = test_woe, save_data = FALSE)[, "score"]
```

get_median

get central value.
**get_names**

Description

This function is not intended to be used by end user.

Usage

get_median(x, weight_avg = NULL)

Arguments

- **x**: A vector or list.
- **weight_avg**: avg weight to calculate means.

---

**get_names**

Get Variable Names

Description

get_names is for getting names of particular classes of variables

Usage

get_names(
  dat,
  types = c("logical", "factor", "character", "numeric", "integer64", "integer",
            "double", "Date", "POSIXlt", "POSIXct", "POSIXt"),
  ex_cols = NULL,
  get_ex = FALSE
)

Arguments

- **dat**: A data.frame with independent variables and target variable.
- **types**: The class or types of variables which names to get. Default: c('numeric', 'integer', 'double')
- **ex_cols**: A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
- **get_ex**: Logical. If TRUE, return a list contains names of excluded variables.

Value

A list contains names of variables

See Also

- **get_x_list**
Examples

```r
x_list = get_names(dat = UCICreditCard, types = c('factor', 'character'),
ex_cols = c("default.payment.next.month", "ID$|date$"), get_ex = FALSE)
x_list = get_names(dat = UCICreditCard, types = c('numeric', 'character', "integer"),
ex_cols = c("default.payment.next.month", "ID$|SEX"), get_ex = FALSE)
```

---

**get_nas_random**

Description

This function is not intended to be used by end user.

Usage

```
get_nas_random(dat)
```

Arguments

dat  A data.frame contained only predict variables.

---

**get_psi_all**

Calculate Population Stability Index (PSI) get_psi is used to calculate Population Stability Index (PSI) of an independent variable. get_psi_all can loop through PSI for all specified independent variables.

Description

Calculate Population Stability Index (PSI) get_psi is used to calculate Population Stability Index (PSI) of an independent variable. get_psi_all can loop through PSI for all specified independent variables.

Usage

```
get_psi_all(
  dat,
  x_list = NULL,
  target = NULL,
  dat_test = NULL,
  breaks_list = NULL,
  occur_time = NULL,
  start_date = NULL,
  cut_date = NULL,
  oot_pct = 0.7,
  pos_flag = NULL,
)
get_psi_all

get_psi_all =
parallel = FALSE,
ex_cols = NULL,
as_table = FALSE,
g = 10,
bins_no = TRUE,
note = FALSE
)

get_psi(
    dat,
    x,
    target = NULL,
    dat_test = NULL,
    occur_time = NULL,
    start_date = NULL,
    cut_date = NULL,
    pos_flag = NULL,
    breaks = NULL,
    breaks_list = NULL,
    oot_pct = 0.7,
    g = 10,
    as_table = TRUE,
    note = FALSE,
    bins_no = TRUE
)

Arguments

dat A data.frame with independent variables and target variable.
x_list Names of independent variables.
target The name of target variable.
dat_test A data.frame of test data. Default is NULL.
breaks_list A table containing a list of splitting points for each independent variable. Default is NULL.
occur_time The name of the variable that represents the time at which each observation takes place.
start_date The earliest occurrence time of observations.
cut_date Time points for splitting data sets, e.g. : splitting Actual and Expected data sets.
oot_pct Percentage of observations retained for overtime test (especially to calculate PSI). Default is 0.7
pos_flag Value of positive class, Default is "1".
parallel Logical, parallel computing. Default is FALSE.
ex_cols Names of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
as_table Logical, output results in a table. Default is TRUE.
get_psi_iv_all

<table>
<thead>
<tr>
<th>g</th>
<th>Number of initial breakpoints for equal frequency binning.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bins_no</td>
<td>Logical, add serial numbers to bins. Default is TRUE.</td>
</tr>
<tr>
<td>note</td>
<td>Logical, outputs info. Default is TRUE.</td>
</tr>
<tr>
<td>x</td>
<td>The name of an independent variable.</td>
</tr>
<tr>
<td>breaks</td>
<td>Splitting points for an independent variable. Default is NULL.</td>
</tr>
</tbody>
</table>

Details

PSI Rules for evaluating the stability of a predictor
Less than 0.02: Very stable
0.02 to 0.1: Stable
0.1 to 0.2: Unstable
0.2 to 0.5: Change more than 0.5: Great change

See Also

get_iv, get_iv_all, get_psi, get_psi_all

Examples

# dat_test is null
get_psi(dat = UCICreditCard, x = "PAY_3", occur_time = "apply_date")

# dat_test is not all
# train_test split
train_test = train_test_split(dat = UCICreditCard, prop = 0.7, split_type = "OOT", occur_time = "apply_date", start_date = NULL, cut_date = NULL, save_data = FALSE, note = FALSE)

dat_ex = train_test$train
dat_ac = train_test$test

# generate psi table
get_psi(dat = dat_ex, dat_test = dat_ac, x = "PAY_3", occur_time = "apply_date", bins_no = TRUE)

get_psi_iv_all

Calculate IV & PSI

Description

get_iv_psi is used to calculate Information Value (IV) and Population Stability Index (PSI) of an independent variable. get_iv_psi_all can loop through IV & PSI for all specified independent variables.

Usage

get_psi_iv_all(
  dat,
  dat_test = NULL,
  x_list = NULL,
  target,
  ex_cols = NULL,
get_psi_iv_all

pos_flag = NULL,
breaks_list = NULL,
occur_time = NULL,
oot_pct = 0.7,
equal_bins = FALSE,
cut_bin = "equal_depth",
tree_control = NULL,
bins_control = NULL,
bins_total = FALSE,
best = TRUE,
g = 10,
as_table = TRUE,
note = FALSE,
parallel = FALSE,
bins_no = TRUE
)

get_psi_iv(
dat,
dat_test = NULL,
x,
target,
pos_flag = NULL,
breaks = NULL,
breaks_list = NULL,
occur_time = NULL,
oot_pct = 0.7,
equal_bins = FALSE,
cut_bin = "equal_depth",
tree_control = NULL,
bins_control = NULL,
bins_total = FALSE,
best = TRUE,
g = 10,
as_table = TRUE,
note = FALSE,
bins_no = TRUE
)

Arguments

dat A data.frame with independent variables and target variable.
dat_test A data.frame of test data. Default is NULL.
x_list Names of independent variables.
target The name of target variable.
ex_cols A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
**pos_flag**

The value of positive class of target variable, default: "1".

**breaks_list**

A table containing a list of splitting points for each independent variable. Default is NULL.

**occur_time**

The name of the variable that represents the time at which each observation takes place.

**oot_pct**

Percentage of observations retained for overtime test (especially to calculate PSI). Default is 0.7

**equal_bins**

Logical, generates initial breaks for equal frequency or width binning.

**cut_bin**

A string, if equal_bins is TRUE, 'equal_depth' or 'equal_width', default is 'equal_depth'.

**tree_control**

Parameters of using Decision Tree to segment initial breaks. See details: get_tree_breaks

**bins_control**

Parameters used to control binning. See details: select_best_class, select_best_breaks

**bins_total**

Logical, total sum for each variable.

**best**

Logical, merge initial breaks to get optimal breaks for binning.

**g**

Number of initial breakpoints for equal frequency binning.

**as_table**

Logical, output results in a table. Default is TRUE.

**note**

Logical, outputs info. Default is TRUE.

**parallel**

Logical, parallel computing. Default is FALSE.

**bins_no**

Logical, add serial numbers to bins. Default is FALSE.

**x**

The name of an independent variable.

**breaks**

Splitting points for an independent variable. Default is NULL.

**See Also**

get_iv, get_iv_all, get_psi, get_psi_all

**Examples**

```r
iv_list = get_psi_iv_all(dat = UCICreditCard[1:1000, ],
x_list = names(UCICreditCard)[3:5], equal_bins = TRUE,
target = "default.payment.next.month", ex_cols = "ID|apply_date")
get_psi_iv(UCICreditCard, x = "PAY_3",
target = "default.payment.next.month", bins_total = TRUE)
```

---

**get_psi_plots**

*Plot PSI (Population Stability Index)*

**Description**

You can use the psi_plot to plot PSI of your data. get_psi_plots can loop through plots for all specified independent variables.
get_psi_plots

Usage

get_psi_plots(
    dat_train,
    dat_test = NULL,
    x_list = NULL,
    ex_cols = NULL,
    breaks_list = NULL,
    occur_time = NULL,
    g = 10,
    plot_show = TRUE,
    save_data = FALSE,
    file_name = NULL,
    parallel = FALSE,
    g_width = 8,
    dir_path = tempdir()
)

psi_plot(
    dat_train,
    x,
    dat_test = NULL,
    occur_time = NULL,
    g_width = 8,
    breaks_list = NULL,
    breaks = NULL,
    g = 10,
    plot_show = TRUE,
    save_data = FALSE,
    dir_path = tempdir()
)

Arguments

dat_train A data.frame with independent variables.
dat_test A data.frame of test data. Default is NULL.
x_list Names of independent variables.
ex_cols A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
breaks_list A table containing a list of splitting points for each independent variable. Default is NULL.
occur_time The name of occur time.
g Number of initial breakpoints for equal frequency binning.
plot_show Logical, show model performance in current graphic device. Default is FALSE.
save_data Logical, save results in locally specified folder. Default is FALSE.
file_name The name for periodically saved data file. Default is NULL.
get_score_card

parallel Logical, parallel computing. Default is FALSE.
g_width The width of graphs.
dir_path The path for periodically saved graphic files.
x The name of an independent variable.
breaks Splitting points for a continues variable.

Examples

```r
train_test = train_test_split(UCICreditCard[1:1000,], split_type = "Random", prop = 0.8, save_data = FALSE)
dat_train = train_test$train
dat_test = train_test$test
get_psi_plots(dat_train[, c(8, 9)], dat_test = dat_test[, c(8, 9)])
```

Description

get_score_card is for generating a standard scorecard

Usage

```r
get_score_card(
  lg_model,
  target,
  bins_table,
  a = 600,
  b = 50,
  file_name = NULL,
  dir_path = tempdir(),
  save_data = FALSE
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lg_model</td>
<td>An object of glm model.</td>
</tr>
<tr>
<td>target</td>
<td>The name of target variable.</td>
</tr>
<tr>
<td>bins_table</td>
<td>a data.frame generated by <code>get_bins_table</code></td>
</tr>
<tr>
<td>a</td>
<td>Base line of score.</td>
</tr>
<tr>
<td>b</td>
<td>Numeric. Increased scores from doubling Odds.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name for periodically saved scorecard file. Default is &quot;LR_Score_Card&quot;.</td>
</tr>
<tr>
<td>dir_path</td>
<td>The path for periodically saved scorecard file. Default is &quot;/model&quot;</td>
</tr>
<tr>
<td>save_data</td>
<td>Logical, save results in locally specified folder. Default is FALSE.</td>
</tr>
</tbody>
</table>
get_score_card

Value

scorecard

Examples

# dataset splitting
sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
# rename the target variable
dat = re_name(dat, "default.payment.next.month", "target")
dat = data_cleansing(dat, target = "target", obs_id = "ID",
occur_time = "apply_date", miss_values = list("", -1))
# train test pliting
train_test = train_test_split(dat, split_type = "OOT", prop = 0.7,
occur_time = "apply_date")
dat_train = train_test$train
dat_test = train_test$test
# get breaks of all predictive variables
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "EDUCATION", "PAY_3", "PAY_2")
breaks_list = get_breaks_all(dat = dat_train, target = "target",
x_list = x_list, occur_time = "apply_date", ex_cols = "ID",
save_data = FALSE, note = FALSE)
# woe transforming
train_woe = woe_trans_all(dat = dat_train,
target = "target",
breaks_list = breaks_list,
woe_name = FALSE)
test_woe = woe_trans_all(dat = dat_test,
target = "target",
breaks_list = breaks_list,
note = FALSE)
Formula = as.formula(paste("target", paste(x_list, collapse = " + "), sep = " ~ "))
set.seed(46)
lr_model = glm(Formula, data = train_woe[, c("target", x_list)], family = binomial(logit))
# get LR coefficient
dt_imp_LR = get_logistic_coef(lg_model = lr_model, save_data = FALSE)
bins_table = get_bins_table_all(dat = dat_train, target = "target",
dat_test = dat_test,
x_list = x_list,
breaks_list = breaks_list, note = FALSE)
# score card
LR_score_card = get_score_card(lg_model = lr_model, bins_table, target = "target")
# scoring
train_pred = dat_train[, c("ID", "apply_date", "target")]
test_pred = dat_test[, c("ID", "apply_date", "target")]
train_pred$pred_LR = score_transfer(model = lr_model,
tbl_woe = train_woe,
save_data = FALSE)[, "score"]
test_pred$pred_LR = score_transfer(model = lr_model,
tbl_woe = test_woe, save_data = FALSE)[, "score"]
get_sim_sign_lambda

Description
This function is not intended to be used by end user.

Usage
get_shadow_nas(dat)

Arguments
- **dat**: A data.frame contained only predict variables.

get_sim_sign_lambda
get_sim_sign_lambda is for get Best lambda required in lasso_filter. This function required in lasso_filter

Description
get_sim_sign_lambda get_sim_sign_lambda is for get Best lambda required in lasso_filter. This function required in lasso_filter

Usage
get_sim_sign_lambda(lasso_model, sim_sign = "negtive")

Arguments
- **lasso_model**: A lasso model generated by glmnet.
- **sim_sign**: Default is "negtive". This is related to pos_plag. If pos_flag equals 1 or 1, the value must be set to negative. If pos_flag equals 0 or 0, the value must be set to positive.

Details
lambda.sim_sign give the model with the same positive or negative coefficients of all variables.

Value
Lanmbda value
get_tree_breaks

Description

get_tree_breaks is for generating initial breaks by decision tree for a numerical or nominal variable. The get_breaks function is a simpler wrapper for get_tree_breaks.

Usage

get_tree_breaks(
  dat,
  x,
  target,
  pos_flag = NULL,
  tree_control = list(p = 0.02, cp = 1e-06, xval = 5, maxdepth = 10),
  sp_values = NULL
)

Arguments

dat A data frame with x and target.
x name of variable to cut breaks by tree.
target The name of target variable.
pos_flag The value of positive class of target variable, default: "1".
tree_control the list of parameters to control cutting initial breaks by decision tree.
  • p the minimum percent of observations in any terminal <leaf> node. 0 < p < 1; 0.01 to 0.1 usually work.
  • cp complexity parameter. the larger, the more conservative the algorithm will be. 0 < cp < 1; 0.0001 to 0.000001 usually work.
  • xval number of cross-validations. Default: 5
  • max_depth maximum depth of a tree. Default: 10
sp_values A list of special value. Default: NULL.

See Also

get_breaks, get_breaks_all

Examples

# tree breaks
tree_control = list(p = 0.02, cp = 0.000001, xval = 5, maxdepth = 10)
tree_breaks = get_tree_breaks(dat = UCICreditCard, x = "MARRIAGE", target = "default.payment.next.month", tree_control = tree_control)
get_x_list  

Get X List.

Description

get_x_list is for getting intersect names of x_list, train and test.

Usage

get_x_list(
  dat_train = NULL,
  dat_test = NULL,
  x_list = NULL,
  ex_cols = NULL,
  note = FALSE
)

Arguments

dat_train    A data.frame with independent variables.
dat_test     Another data.frame.
x_list       Names of independent variables.
ex_cols      A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
note         Logical. Outputs info. Default is TRUE.

Value

A list contains names of variables

See Also

get_names

Examples

x_list = get_x_list(x_list = NULL, dat_train = UCICreditCard,
ex_cols = c("default.payment.next.month","ID$|_date$"))
**high_cor_selector**

*Compare the two highly correlated variables*

**Description**

`high_cor_selector` is a function for comparing the two highly correlated variables, select a variable with the largest IV value.

**Usage**

```r
high_cor_selector(
  cor_mat,
  p = 0.95,
  x_list = NULL,
  com_list = NULL,
  retain = TRUE
)
```

**Arguments**

- `cor_mat`: A correlation matrix.
- `p`: The threshold of high correlation.
- `x_list`: Names of independent variables.
- `com_list`: A data.frame with important values of each variable. eg: IV_list.
- `retain`: Logical, output selected variables, if FALSE, output filtered variables.

**Value**

A list of selected variables.

---

**is_date**

*is_date*

**Description**

`is_date` is a small function for distinguishing time formats

**Usage**

```r
is_date(x)
```

**Arguments**

- `x`: list or vectors
**knn_nas_imp**

### Value
A Date.

### Examples

```r
is_date(lendingclub$issue_d)
```

---

### knn_nas_imp

**Impute nas using KNN**

### Description
This function is not intended to be used by end user.

### Usage

```r
knn_nas_imp(
  dat,
  x,
  nas_rate = NULL,
  mat_nas_shadow = NULL,
  dt_nas_random = NULL,
  k = 10,
  scale = FALSE,
  method = "median",
  miss_value_num = -1
)
```

### Arguments

- **dat**: A data.frame with independent variables.
- **x**: The name of variable to process.
- **nas_rate**: A list contains nas rate of each variable.
- **mat_nas_shadow**: A shadow matrix of variables which contain nas.
- **dt_nas_random**: A data.frame with random nas imputation.
- **k**: Number of neighbors of each obs which x is missing.
- **scale**: Logical. Standardization of variable.
- **method**: The methods of imputation by knn. "median" is knn imputation with k neighbors median, "avg_dist" is knn imputation with k neighbors of distance weighted mean.
- **miss_value_num**: Default value of missing data imputation for numeric variables, Default is -1.
ks_table

Description

ks_table is for generating a model performance table. ks_table_plot is for plotting the table generated by ks_table. ks_psi_plot is for K-S & PSI distribution plotting.

Usage

ks_table(
  train_pred,
  test_pred = NULL,
  target = NULL,
  score = NULL,
  g = 10,
  breaks = NULL,
  pos_flag = list("1", "1", "Bad", 1)
)

ks_table_plot(
  train_pred,
  test_pred,
  target = "target",
  score = "score",
  g = 10,
  plot_show = TRUE,
  g_width = 12,
  file_name = NULL,
  save_data = FALSE,
  dir_path = tempdir(),
  gtitle = NULL
)

ks_psi_plot(
  train_pred,
  test_pred,
  target = "target",
  score = "score",
  gtitle = NULL,
  plot_show = TRUE,
  g_width = 12,
  save_data = FALSE,
  breaks = NULL,
  g = 10,
  dir_path = tempdir()
)
model_key_index(tb_pred)

Arguments

- **train_pred**: A data frame of training with predicted prob or score.
- **test_pred**: A data frame of validation with predict prob or score.
- **target**: The name of target variable.
- **score**: The name of prob or score variable.
- **g**: Number of breaks for prob or score.
- **breaks**: Splitting points of prob or score.
- **pos_flag**: The value of positive class of target variable, default: "1".
- **plot_show**: Logical, show model performance in current graphic device. Default is FALSE.
- **g_width**: Width of graphs.
- **file_name**: The name for periodically saved data file. Default is NULL.
- **save_data**: Logical, save results in locally specified folder. Default is FALSE.
- **dir_path**: The path for periodically saved graphic files.
- **gtitle**: The title of the graph & The name for periodically saved graphic file. Default is "_ks_psi_table".
- **tb_pred**: A table generated by codeks_table

Examples

```r
sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
dat = re_name(dat, "default.payment.next.month", "target")
dat = data_cleansing(dat, target = "target", obs_id = "ID",
  occur_time = "apply_date", miss_values = list("", -1))
train_test = train_test_split(dat, split_type = "OOT", prop = 0.7,
  occur_time = "apply_date")
dat_train = train_test$train
dat_test = train_test$test
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "PAY_3", "PAY_2")
Formula = as.formula(paste("target", paste(x_list, collapse = " + "), sep = " - "))
set.seed(46)
lr_model = glm(Formula, data = dat_train[, c("target", x_list)], family = binomial(logit))

dat_train$pred_LR = round(predict(lr_model, dat_train[, x_list], type = "response"), 5)
dat_test$pred_LR = round(predict(lr_model, dat_test[, x_list], type = "response"), 5)
# model evaluation
ks_psi_plot(train_pred = dat_train, test_pred = dat_test,
  score = "pred_LR", target = "target",
  plot_show = TRUE)
tb_pred = ks_table_plot(train_pred = dat_train, test_pred = dat_test,
  score = "pred_LR", target = "target",
  g = 10, g_width = 13, plot_show = FALSE)
key_index = model_key_index(tb_pred)
```
### ks_value

**Description**

ks_value is for get K-S value for a prob or score.

**Usage**

```r
ks_value(target, prob)
```

**Arguments**

- `target`: Vector of target.
- `prob`: A list of predict probability or score.

**Value**

KS value

### lasso_filter

**Description**

lasso_filter filter variables by lasso.

**Usage**

```r
lasso_filter(
  dat_train,
  dat_test = NULL,
  target = NULL,
  x_list = NULL,
  pos_flag = NULL,
  ex_cols = NULL,
  sim_sign = "negative",
  best_lambda = "lambda.auc",
  save_data = FALSE,
  plot.it = TRUE,
  seed = 46,
  file_name = NULL,
  dir_path = tempdir(),
  note = FALSE
)
```
lasso_filter

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dat_train</td>
<td>A data.frame with independent variables and target variable.</td>
</tr>
<tr>
<td>dat_test</td>
<td>A data.frame of test data. Default is NULL.</td>
</tr>
<tr>
<td>target</td>
<td>The name of target variable.</td>
</tr>
<tr>
<td>x_list</td>
<td>Names of independent variables.</td>
</tr>
<tr>
<td>pos_flag</td>
<td>The value of positive class of target variable, default: &quot;1&quot;.</td>
</tr>
<tr>
<td>ex_cols</td>
<td>A list of excluded variables. Regular expressions can also be used to match</td>
</tr>
<tr>
<td></td>
<td>variable names. Default is NULL.</td>
</tr>
<tr>
<td>sim_sign</td>
<td>The coefficients of all variables should be all negative or positive, after</td>
</tr>
<tr>
<td></td>
<td>turning to woe. Default is &quot;negative&quot; for pos_flag is &quot;1&quot;.</td>
</tr>
<tr>
<td>best_lambda</td>
<td>Methods of best lambda standards using to filter variables by LASSO. There</td>
</tr>
<tr>
<td></td>
<td>are 3 methods: (&quot;lambda.auc&quot;, &quot;lambda.ks&quot;, &quot;lambda.sim_sign&quot;) . Default is</td>
</tr>
<tr>
<td></td>
<td>&quot;lambda.auc&quot;.</td>
</tr>
<tr>
<td>save_data</td>
<td>Logical, save results in locally specified folder. Default is FALSE</td>
</tr>
<tr>
<td>plot.it</td>
<td>Logical, shrinkage plot. Default is TRUE.</td>
</tr>
<tr>
<td>seed</td>
<td>Random number seed. Default is 46.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name for periodically saved results files. Default is &quot;Feature_selected_LASSO&quot;.</td>
</tr>
<tr>
<td>dir_path</td>
<td>The path for periodically saved results files. Default is &quot;/variable&quot;.</td>
</tr>
<tr>
<td>note</td>
<td>Logical, outputs info. Default is FALSE.</td>
</tr>
</tbody>
</table>

**Value**

A list of filtered x variables by lasso.

**Examples**

```r
sub = cv_split(UCICreditCard, k = 40)[[1]]
dat = UCICreditCard[sub,]
dat = re_name(dat, "default.payment.next.month", "target")
dat_train = data_cleansing(dat, target = "target", obs_id = "ID", occur_time = "apply_date", miss_values = list("", -1))
dat_train = process_nas(dat_train)
#get breaks of all predictive variables
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "EDUCATION", "PAY_3", "PAY_2")
breaks_list = get_breaks_all(dat = dat_train, target = "target",
                           x_list = x_list, occur_time = "apply_date", ex_cols = "ID",
                           save_data = FALSE, note = FALSE)
#woe transform
train_woe = woe_trans_all(dat = dat_train,x_list = x_list,
                          target = "target", breaks_list = breaks_list,
                          woe_name = FALSE)
lasso_filter(dat_train = train_woe,
             target = "target", x_list = x_list,
             save_data = FALSE, plot.it = FALSE)
```
Description

This data contains complete loan data for all loans issued through the time period stated, including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information. The data containing loan data through the "present" contains complete loan data for all loans issued through the previous completed calendar quarter (time period: 2018Q1:2018Q4).

Format

A data frame with 63532 rows and 145 variables.

Details

- id: A unique LC assigned ID for the loan listing.
- issue_d: The month which the loan was funded.
- loan_status: Current status of the loan.
- addr_state: The state provided by the borrower in the loan application.
- acc_open_past_24mths: Number of trades opened in past 24 months.
- all_util: Balance to credit limit on all trades.
- annual_inc: The self-reported annual income provided by the borrower during registration.
- avg_cur_bal: Average current balance of all accounts.
- bc_open_to_buy: Total open to buy on revolving bankcards.
- bc_util: Ratio of total current balance to high credit/credit limit for all bankcard accounts.
- dti: A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower’s self-reported monthly income.
- dti_joint: A ratio calculated using the co:borrowers’ total monthly payments on the total debt obligations, excluding mortgages and the requested LC loan, divided by the co:borrowers’ combined self:reported monthly income.
- emp_length: Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
- emp_title: The job title supplied by the Borrower when applying for the loan.
- funded_amnt_inv: The total amount committed by investors for that loan at that point in time.
- grade: LC assigned loan grade
- inq_last_12m: Number of credit inquiries in past 12 months
- installment: The monthly payment owed by the borrower if the loan originates.
- max_bal_bc: Maximum current balance owed on all revolving accounts
- mo_sin_old_il_acct: Months since oldest bank installment account opened
lift_value

- mo_sin_old_rev_tl_op: Months since oldest revolving account opened
- mo_sin_rcnt_rev_tl_op: Months since most recent revolving account opened
- mo_sin_rcnt_tl: Months since most recent account opened
- mort_acc: Number of mortgage accounts.
- pct_tl_nvr_dlq: Percent of trades never delinquent
- percent_bc_gt_75: Percentage of all bankcard accounts > 75
- purpose: A category provided by the borrower for the loan request.
- sub_grade: LC assigned loan subgrade
- term: The number of payments on the loan. Values are in months and can be either 36 or 60.
- tot_cur_bal: Total current balance of all accounts
- tot_hi_cred_lim: Total high credit/credit limit
- total_acc: The total number of credit lines currently in the borrower’s credit file
- total_bal_ex_mort: Total credit balance excluding mortgage
- total_be_limit: Total bankcard high credit/credit limit
- total_cu_tl: Number of finance trades
- total_il_high_credit_limit: Total installment high credit/credit limit
- verification_status_joint: Indicates if the co:borrowers’ joint income was verified by LC, not verified, or if the income source was verified
- zip_code: The first 3 numbers of the zip code provided by the borrower in the loan application.

See Also

UCICreditCard

lift_value

Description

lift_value is for getting max lift value for a prob or score.

Usage

lift_value(target, prob)

Arguments

target: Vector of target.
prob: A list of predict probability or score.

Value

Max lift value
local_outlier_factor

local_outlier_factor local_outlier_factor is function for calculating the lof factor for a data set using knn. This function is not intended to be used by end user.

Description

local_outlier_factor local_outlier_factor is function for calculating the lof factor for a data set using knn. This function is not intended to be used by end user.

Usage

local_outlier_factor(dat, k = 10)

Arguments

dat A data.frame contained only predict variables.
k Number of neighbors for LOF. Default is 10.

log_trans

Description

log_trans is for logarithmic transformation

Usage

log_trans(dat, target, x_list = NULL, cor_dif = 0.01, ex_cols = NULL, note = TRUE)

log_vars(dat, x_list = NULL, target = NULL, cor_dif = 0.01, ex_cols = NULL)
Arguments

- **dat**: A data.frame.
- **target**: The name of target variable.
- **x_list**: A list of x variables.
- **cor_dif**: The correlation coefficient difference with the target of logarithm transformed variable and original variable.
- **ex_cols**: Names of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
- **note**: Logical, outputs info. Default is TRUE.

Value

Log transformed data.frame.

Examples

```r
dat = log_trans(dat = UCICreditCard, target = "default.payment.next.month", x_list = NULL, cor_dif = 0.01, ex_cols = "ID", note = TRUE)
```

Description

Loop Function. `loop_function` is an iterator to loop through

Usage

```r
loop_function(
  func = NULL,
  args = list(data = NULL),
  x_list = NULL,
  bind = "rbind",
  parallel = TRUE,
  as_list = FALSE
)
```

Arguments

- **func**: A function.
- **args**: A list of arguments required by function.
- **x_list**: Names of objects to loop through.
- **bind**: Compile results, "rbind" & "cbind" are available.
- **parallel**: Logical, parallel computing.
- **as_list**: Logical, whether outputs to be a list.
**Value**

A data.frame or list

**Examples**

```r
dat = UCICreditCard[24:26]
num_x_list = get_names(dat = dat, types = c('numeric', 'integer', 'double'),
                       ex_cols = NULL, get_ex = FALSE)
dat[,num_x_list] = loop_function(func = outliers_kmeans_lof, x_list = num_x_list,
                                args = list(dat = dat),
                                bind = "cbind", as_list = FALSE,
                                parallel = FALSE)
```

---

**Description**

`love_color` is for get plots for a variable.

**Usage**

```r
love_color(color = NULL, type = "Blues", n = 10, ...)
```

**Arguments**

- **color**
  The name of colors.

- **type**
  The type of colors. "deep", or the name of palette:. The sequential palettes names are Blues BuGn BuPu GnBu Greens Greys OrRd PuBu PuBuGn PuRd Purples RdPu Reds YiGn YiGnBu YiOrBr YiOrRd The diverging palettes are BrBG PiYG PRGn PuOr RdBu RdGy RdYlBu RdYlGn Spectral The qualitative palettes are Accent, Dark2, Paired, Pastel1, Pastel2, Set1, Set2, Set3

- **n**
  Number of different colors, minimum is 1.

- **...**
  Other parameters.

**Examples**

```r
love_color(color="dark_cyan")
```
low_variance_filter  Filtering Low Variance Variables

Description

low_variance_filter is for removing variables with repeated values up to a certain percentage.

Usage

```r
low_variance_filter(
  dat,
  lvp = 0.97,
  only_NA = FALSE,
  note = FALSE,
  ex_cols = NULL
)
```

Arguments

dat  A data frame with x and target.
lvp  The maximum percent of unique values (including NAs).
only_NA  Logical, only process variables which NA's rate are more than lvp.
note  Logical. Outputs info. Default is TRUE.
ex_cols  A list of excluded variables. Default is NULL.

Value

A data.frame

Examples

```r
dat = low_variance_filter(lendingclub[1:1000, ], lvp = 0.9)
```

lr_params  Logistic Regression & Scorecard Parameters

Description

lr_params is the list of parameters to train a LR model or Scorecard using in training_model. lr_params_search is for searching the optimal parameters of logistic regression, if any parameters of params in lr_params is more than one.
**Usage**

```r
lr_params(
  tree_control = list(p = 0.02, cp = 1e-08, xval = 5, maxdepth = 10),
  bins_control = list(bins_num = 10, bins_pct = 0.05, b_chi = 0.02, b_odds = 0.1, b.psi
      = 0.03, b_or = 0.15, mono = 0.2, odds.psi = 0.15, kc = 1),
  f_eval = "ks",
  best_lambda = "lambda.ks",
  method = "random_search",
  iters = 10,
  lasso = TRUE,
  step_wise = TRUE,
  score_card = TRUE,
  sp_values = NULL,
  forced_in = NULL,
  obsweight = c(1, 1),
  thresholds = list(cor_p = 0.8, iv_i = 0.02, psi_i = 0.1, cos_i = 0.5),
  ...
)
```

```r
lr_params_search(
  method = "random_search",
  dat_train,
  target,
  dat_test = NULL,
  occur_time = NULL,
  x_list = NULL,
  prop = 0.7,
  iters = 10,
  tree_control = list(p = 0.02, cp = 0, xval = 1, maxdepth = 10),
  bins_control = list(bins_num = 10, bins_pct = 0.05, b_chi = 0.02, b_odds = 0.1, b.psi
      = 0.05, b_or = 0.1, mono = 0.1, odds.psi = 0.03, kc = 1),
  thresholds = list(cor_p = 0.8, iv_i = 0.02, psi_i = 0.1, cos_i = 0.6),
  step_wise = FALSE,
  lasso = FALSE,
  f_eval = "ks"
)
```

**Arguments**

- `tree_control` the list of parameters to control cutting initial breaks by decision tree. See details at: `get_tree_breaks`
- `bins_control` the list of parameters to control merging initial breaks. See details at: `select_best_breaks`, `select_best_class`
- `f_eval` Customized evaluation function, "ks" & "auc" are available.
- `best_lambda` Methods of best lambda standards using to filter variables by LASSO. There are 3 methods: ("lambda.auc", "lambda.ks", "lambda.sim_sign") . Default is "lambda.auc".
Method of searching optimal parameters. "random_search","grid_search","local_search" are available.

Number of iterations of "random_search" optimal parameters.

Logical, if TRUE, variables filtering by LASSO. Default is TRUE.

Logical, stepwise method. Default is TRUE.

Logical, transfer woe to a standard scorecard. If TRUE, Output scorecard, and score prediction, otherwise output probability. Default is TRUE.

Values will be in separate bins.e.g. list(-1, "missing") means that -1 & missing as special values.Default is NULL.

Names of forced input variables. Default is NULL.

An optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector. If you oversample or cluster different datasets to training the LR model, you need to set this parameter to ensure that the probability of logistic regression output is the same as that before oversampling or segmentation. e.g.: There are 10,000 0 obs and 500 1 obs before oversampling or under-sampling, 5,000 0 obs and 3,000 1 obs after oversampling. Then this parameter should be set to c(10000/5000, 500/3000). Default is NULL..

Thresholds for selecting variables.

- cor_p The maximum threshold of correlation. Default: 0.8.
- iv_i The minimum threshold of IV. 0.01 to 0.1 usually work. Default: 0.02
- psi_i The maximum threshold of PSI. 0.1 to 0.3 usually work. Default: 0.1.
- cos_i cos_similarity of positive rate of train and test. 0.7 to 0.9 usually work.Default: 0.5.

... Other parameters

data.frame of train data. Default is NULL.

name of target variable.

data.frame of test data. Default is NULL.

The name of the variable that represents the time at which each observation takes place. Default is NULL.

names of independent variables. Default is NULL.

Percentage of train-data after the partition. Default: 0.7.

A list of parameters.

See Also

training_model, xgb_params, gbm_params, rf_params
lr_vif

Variance-Inflation Factors

Description

lr_vif is for calculating Variance-Inflation Factors.

Usage

lr_vif(lr_model)

Arguments

lr_model An object of logistic model.

Examples

sub = cv_split(UCICreditCard, k = 30)[[1]]
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "PAY_3", "PAY_2")
dat = re_name(UCICreditCard[sub,], "default.payment.next.month", "target")
dat = dat[,c("target",x_list)]

dat = data_cleansing(dat, miss_values = list("", -1))

train_test = train_test_split(dat, prop = 0.7)
dat_train = train_test$train
dat_test = train_test$test

Formula = as.formula(paste("target", paste(x_list, collapse = " + "), sep = " - "))
set.seed(46)
lr_model = glm(Formula, data = dat_train[, c("target", x_list)], family = binomial(logit))
lr_vif(lr_model)
get_logistic_coef(lr_model)
class(dat)
mod = lr_model
lr_vif(lr_model)

max_min_norm

Max Min Normalization

Description

max_min_norm is for normalizing each column vector of matrix 'x' using max_min normalization

Usage

max_min_norm(x)
merge_category

Arguments

x       Vector

Value

Normalized vector

Examples

dat_s = apply(UCICreditCard[,12:14], 2, max_min_norm)

Description

merge_category is for merging category of nominal variables which number of categories is more than m or percent of samples in any categories is less than p.

Usage

merge_category(dat, char_list = NULL, ex_cols = NULL, m = 10, note = TRUE)

Arguments

dat       A data frame with x and target.
char_list  The list of characteristic variables that need to merge categories. Default is NULL. In case of NULL, merge categories for all variables of string type.
ex_cols   A list of excluded variables. Default is NULL.
m         The minimum number of categories.
note      Logical, outputs info. Default is TRUE.

Value

A data.frame with merged category variables.

Examples

#merge_category
dat = merge_category(lendingclub, ex_cols = "id$|.d$")
char_list = get_names(dat = dat, types = c(’factor’, ’character’),
ex_cols = "id$|.d$", get_ex = FALSE)
str(dat[,char_list])
**min_max_norm**

*Min Max Normalization*

**Description**

*min_max_norm* is for normalizing each column vector of matrix 'x' using min_max normalization

**Usage**

```
min_max_norm(x)
```

**Arguments**

- **x**  
  Vector

**Value**

Normalized vector

**Examples**

```
dat_s = apply(UCICreditCard[,12:14], 2, min_max_norm)
```
Usage

```r
model_result_plot(
    train_pred,
    score,
    target,
    test_pred = NULL,
    gtitle = NULL,
    perf_dir_path = NULL,
    save_data = FALSE,
    plot_show = TRUE,
    total = TRUE,
    g = 10,
    cut_bin = "equal_depth",
    digits = 4
)
```

```r
perf_table(
    train_pred,
    test_pred = NULL,
    target = NULL,
    score = NULL,
    g = 10,
    cut_bin = "equal_depth",
    breaks = NULL,
    digits = 2,
    pos_flag = list("1", "1", "Bad", 1),
    total = FALSE,
    binsNO = FALSE
)
```

```r
ks_plot(
    train_pred,
    test_pred = NULL,
    target = NULL,
    score = NULL,
    gtitle = NULL,
    breaks = NULL,
    g = 10,
    cut_bin = "equal_width",
    perf_tb = NULL
)
```

```r
lift_plot(
    train_pred,
    test_pred = NULL,
    target = NULL,
    score = NULL,
    gtitle = NULL,
)
model_result_plot

breaks = NULL,
g = 10,
cut_bin = "equal_depth",
perf_tb = NULL
)

roc_plot(
    train_pred,
    test_pred = NULL,
    target = NULL,
    score = NULL,
gtitle = NULL
)

score_distribution_plot(
    train_pred,
    test_pred,
    target,
    score,
gtitle = NULL,
    breaks = NULL,
g = 10,
cut_bin = "equal_depth",
perf_tb = NULL
)

Arguments

train_pred A data frame of training with predicted prob or score.
score The name of prob or score variable.
target The name of target variable.
test_pred A data frame of validation with predict prob or score.
gtitle The title of the graph & The name for periodically saved graphic file.
perf_dir_path The path for periodically saved graphic files.
save_data Logical, save results in locally specified folder. Default is FALSE.
plot_show Logical, show model performance in current graphic device. Default is TRUE.
total Whether to summarize the table. default: TRUE.
g Number of breaks for prob or score.
cut_bin A string, if equal_bins is TRUE, 'equal_depth' or 'equal_width', default is 'equal_depth'.
digits Digits of numeric,default is 4.
breaks Splitting points of prob or score.
pos_flag The value of positive class of target variable, default: "1".
binsNO Bins Number.Default is FALSE.
perf_tb Performance table.
Examples

```r
sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
dat = re_name(dat, "default.payment.next.month", "target")
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "PAY_3", "PAY_2")
dat = data_cleansing(dat, target = "target", obs_id = "ID", x_list = x_list,
occur_time = "apply_date", miss_values = list("", -1))
dat = process_nas(dat, default_miss = TRUE)
train_test = train_test_split(dat, split_type = "OOT", prop = 0.7,
occur_time = "apply_date")
dat_train = train_test$train
dat_test = train_test$test
Formula = as.formula(paste("target", paste(x_list, collapse = ' + '), sep = ' - '))
set.seed(46)
lr_model = glm(Formula, data = dat_train[, c("target", x_list)], family = binomial(logit))

dat_train$pred_LR = round(predict(lr_model, dat_train[, x_list], type = "response"), 5)
dat_test$pred_LR = round(predict(lr_model, dat_test[, x_list], type = "response"), 5)
# model evaluation
perf_table(train_pred = dat_train, test_pred = dat_test, target = "target", score = "pred_LR")
ks_plot(train_pred = dat_train, test_pred = dat_test, target = "target", score = "pred_LR")
roc_plot(train_pred = dat_train, test_pred = dat_test, target = "target", score = "pred_LR")
#lift_plot(train_pred = dat_train, test_pred = dat_test, target = "target", score = "pred_LR")
#score_distribution_plot(train_pred = dat_train, test_pred = dat_test,
#target = "target", score = "pred_LR")
#model_result_plot(train_pred = dat_train, test_pred = dat_test,
#target = "target", score = "pred_LR")
```

---

multi_grid

Arrange list of plots into a grid

Description

Plot multiple ggplot-objects as a grid-arranged single plot.

Usage

```r
multi_grid(..., grobs = list(...), nrow = NULL, ncol = NULL)
```

Arguments

- `...`: Other parameters.
- `grobs`: A list of ggplot-objects to be arranged into the grid.
- `nrow`: Number of rows in the plot grid.
- `ncol`: Number of columns in the plot grid.
multi_left_join

Details

This function takes a list of ggplot-objects as argument. Plotting functions of this package that produce multiple plot objects (e.g., when there is an argument facet.grid) usually return multiple plots as list.

Value

An object of class gtable.

Examples

```r
library(ggplot2)
sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
dat = re_name(dat, "default.payment.next.month", "target")
dat = data_cleansing(dat, target = "target", obs_id = "ID", occur_time = "apply_date", miss_values = list("", -1))
dat = process_nas(dat)
train_test = train_test_split(dat, split_type = "OOT", prop = 0.7, occur_time = "apply_date")

x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "PAY_3", "PAY_2")
Formula = as.formula(paste("target", paste(x_list, collapse = " + ", sep = " - ")))
set.seed(46)
lr_model = glm(Formula, data = dat_train[, c("target", x_list)], family = binomial(logit))

dat_train$pred_LR = round(predict(lr_model, dat_train[, x_list], type = "response"), 5)
dat_test$pred_LR = round(predict(lr_model, dat_test[, x_list], type = "response"), 5)

# model evaluation
p1 = ks_plot(train_pred = dat_train, test_pred = dat_test, target = "target", score = "pred_LR")
p2 = roc_plot(train_pred = dat_train, test_pred = dat_test, target = "target", score = "pred_LR")
p3 = lift_plot(train_pred = dat_train, test_pred = dat_test, target = "target", score = "pred_LR")
p4 = score_distribution_plot(train_pred = dat_train, test_pred = dat_test, target = "target", score = "pred_LR")
p_plots = multi_grid(p1, p2, p3, p4)
plot(p_plots)
```

multi_left_join

multi_left_join

Description

multi_left_join is for left jion a list of datasets fast.

Usage

```r
multi_left_join(..., df_list = list(...), key_dt = NULL, by = NULL)
```
null_blank_na

Arguments

... Datasets need join
df_list A list of datasets.
key_dt Name or index of Key table to left join.
by Name of Key columns to join.

Examples

multi_left_join(UCICreditCard[1:10, 1:10], UCICreditCard[1:10, c(1,8:14)],
UCICreditCard[1:10, c(1,20:25)], by = "ID")

datss = null_blank_na(dat = UCICreditCard[1:1000, ], miss_values =list(-1,-2))

null_blank_na Encode NAs

Description

null_blank_na is the function to replace null ,NULL, blank or other missing values with NA.

Usage

null_blank_na(dat, miss_values = NULL, note = FALSE)

Arguments

dat A data frame with x and target.
miss_values Other extreme value might be used to represent missing values, e.g: -9999, -9998. These miss_values will be encoded to -1 or "missing".

Value

A data.frame

Examples

datss = null_blank_na(dat = UCICreditCard[1:1000, ], miss_values =list(-1,-2))
**n_char**

*The length of a string.*

---

**Description**

Returns the number of "code points", in a string.

**Usage**

```
 n_char(string)
```

**Arguments**

- **string**
  
  A string.

**Value**

A numeric vector giving number of characters (code points) in each element of the character vector. Missing string have missing length.

**Examples**

```
n_char(letters)
n_char(NA)
```

---

**one_hot_encoding**

*One-Hot Encoding*

---

**Description**

one_hot_encoding is for converting the factor or character variables into multiple columns

**Usage**

```
one_hot_encoding(
  dat,
  cat_vars = NULL,
  ex.cols = NULL,
  merge.cat = TRUE,
  na.act = TRUE,
  note = FALSE
)
```
outliers_detection

Arguments

dat A data frame.
cat_vars The name or column index list to be one_hot encoded.
ex_cols Variables to be excluded, use regular expression matching.
merge_cat Logical. If TRUE, to merge categories greater than 8, default is TRUE.
na_act Logical. If true, the missing value is processed, if FALSE missing value is omitted.

Value

A data frame with the one hot encoding applied to all the variables with type as factor or character.

See Also
de_one_hot_encoding

Examples

dat1 = one_hot_encoding(dat = UCICreditCard,
cat_vars = c("SEX", "MARRIAGE"),
merge_cat = TRUE, na_act = TRUE)
dat2 = de_one_hot_encoding(dat_one_hot = dat1,
cat_vars = c("SEX", "MARRIAGE"), na_act = FALSE)

Description

Outliers Detection outliers_detection is for outliers detecting using Kmeans and Local Outlier Factor (lof)

Usage

outliers_detection(dat, x, kc = 3, kn = 5)

Arguments

dat A data.frame with independent variables.
x The name of variable to process.
kc Number of clustering centers for Kmeans
kn Number of neighbors for LOF.
Value

Outliers of each variable.

Description

partial_dependence_plot is for generating a partial dependence plot. get_partial_dependence_plots is for plotting partial dependence of all variables in x_list.

Usage

partial_dependence_plot(model, x, x_train, n.trees = NULL)

get_partial_dependence_plots(
    model,
    x_train,
    x_list,
    n.trees = NULL,
    dir_path = getwd(),
    save_data = TRUE,
    plot_show = FALSE,
    parallel = FALSE
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>A data frame of training with predicted prob or score.</td>
</tr>
<tr>
<td>x</td>
<td>The name of an independent variable.</td>
</tr>
<tr>
<td>x_train</td>
<td>A data frame with independent variables.</td>
</tr>
<tr>
<td>n.trees</td>
<td>Number of trees for best.iter of gbm.</td>
</tr>
<tr>
<td>x_list</td>
<td>Names of independent variables.</td>
</tr>
<tr>
<td>dir_path</td>
<td>The path for periodically saved graphic files.</td>
</tr>
<tr>
<td>save_data</td>
<td>Logical, save results in locally specified folder. Default is FALSE.</td>
</tr>
<tr>
<td>plot_show</td>
<td>Logical, show model performance in current graphic device. Default is FALSE.</td>
</tr>
<tr>
<td>parallel</td>
<td>Logical, parallel computing. Default is FALSE.</td>
</tr>
</tbody>
</table>
Examples

sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
dat = re_name(dat, "default.payment.next.month", "target")
dat = data_cleansing(dat, target = "target", obs_id = "ID",
  occur_time = "apply_date", miss_values = list("", -1))

train_test = train_test_split(dat, split_type = "OOT", prop = 0.7,
  occur_time = "apply_date")
dat_train = train_test$train
dat_test = train_test$test
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "PAY_3", "PAY_2")
Formula = as.formula(paste("target", paste(x_list, collapse = '+'), sep = '- '))
set.seed(46)
lr_model = glm(Formula, data = dat_train[, c("target", x_list)], family = binomial(logit))
#plot partial dependency of one variable
partial_dependence_plot(model = lr_model, x = "LIMIT_BAL", x_train = dat_train)
#plot partial dependency of all variables
pd_list = get_partial_dependence_plots(model = lr_model, x_list = x_list[1:2],
  x_train = dat_train, save_data = FALSE, plot_show = TRUE)

PCA_reduce

PCA Dimension Reduction

Description

PCA_reduce is used for PCA reduction of high dimension data.

Usage

PCA_reduce(train = train, test = NULL, mc = 0.9)

Arguments

train  A data.frame with independent variables and target variable.

test   A data.frame of test data.

mc      Threshold of cumulative imp.

Examples

## Not run:
num_x_list = get_names(dat = UCICreditCard, types = c('numeric'),
ex_cols = "ID$|date$|default.payment.next.month$", get_ex = FALSE)
PCA_dat = PCA_reduce(train = UCICreditCard[num_x_list])

## End(Not run)
plot_colors

**Plot Colors**

**Description**

You can use the `plot_colors` to show colors on the graph device.

**Usage**

```r
plot_colors(colors)
color_ramp_palette(colors)
```

**Arguments**

- `colors` A vector of colors.

**Examples**

```r
plot_colors(rgb(158,122,122, maxColorValue = 255 ))
```

---

plot_oot_perf

**plot_oot_perf** `plot_oot_perf` is for plotting performance of cross time samples in the future

**Description**

`plot_oot_perf` is for plotting performance of cross time samples in the future

**Usage**

```r
plot_oot_perf(
  dat_test,
  x, 
  occur_time, 
  target, 
  k = 3, 
  g = 10, 
  period = "month", 
  best = FALSE, 
  equal_bins = TRUE, 
  pl = "rate", 
  breaks = NULL, 
  cut_bin = "equal_depth", 
  gtitle = NULL, 
  perf_dir_path = NULL,
)```
Arguments

dat_test  A data frame of testing dataset with predicted prob or score.
x  The name of prob or score variable.
occur_time  The name of the variable that represents the time at which each observation takes place.
target  The name of target variable.
k  If period is NULL, number of equal frequency samples.
g  Number of breaks for prob or score.
period  OOT period, ‘weekly’ and ‘month’ are available. If NULL, use k equal frequency samples.
best  Logical, merge initial breaks to get optimal breaks for binning.
equal_bins  Logical, generates initial breaks for equal frequency or width binning.
pl  ’lift’ is for lift chart plot, ’rate’ is for positive rate plot.
breaks  Splitting points of prob or score.
cut_bin  A string, if equal_bins is TRUE, ‘equal_depth’ or ‘equal_width’, default is ‘equal_depth’.
gtitle  The title of the graph & The name for periodically saved graphic file.
perf_dir_path  The path for periodically saved graphic files.
save_data  Logical, save results in locally specified folder. Default is FALSE.
plot_show  Logical, show model performance in current graphic device. Default is TRUE.

Examples

sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
dat = re_name(dat, "default.payment.next.month", "target")
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "PAY_3", "PAY_2")
dat = data_cleansing(dat, target = "target", obs_id = "ID", x_list = x_list,
occur_time = "apply_date", miss_values = list("", -1))
dat = process_nas(dat)
train_test = train_test_split(dat, split_type = "OOT", prop = 0.7,
          occur_time = "apply_date")
dat_train = train_test$train
dat_test = train_test$test
Formula = as.formula(paste("target", paste(x_list, collapse = " + "), sep = " - "))
set.seed(46)
lr_model = glm(Formula, data = dat_train[, c("target", x_list)], family = binomial(logit))

dat_train$pred_LR = round(predict(lr_model, dat_train[, x_list], type = "response"), 5)
dat_test$pred_LR = round(predict(lr_model, dat_test[, x_list], type = "response"), 5)
plot_oot_perf(dat_test = dat_test, occur_time = "apply_date", target = "target", x = "pred_LR")
**Description**

`plot_table` is for table visualization.

**Usage**

```r
plot_table(
  grid_table,
  theme = c("cyan", "grey", "green", "red", "blue", "purple"),
  title = NULL,
  title.size = 12,
  title.color = "black",
  title.face = "bold",
  title.position = "middle",
  subtitle = NULL,
  subtitle.size = 8,
  subtitle.color = "black",
  subtitle.face = "plain",
  subtitle.position = "middle",
  tile.color = "white",
  tile.size = 1,
  colname.size = 3,
  colname.color = "white",
  colname.face = "bold",
  colname.fill.color = love_color("dark_cyan"),
  text.size = 3,
  text.color = love_color("dark_grey"),
  text.face = "plain",
  text.fill.color = c("white", love_color("pale_grey"))
)
```

**Arguments**

- `grid_table`: A data.frame or table
- `theme`: The theme of color, "cyan", "grey", "green", "red", "blue", "purple" are available.
- `title`: The title of table
- `title.size`: The title size of plot.
- `title.color`: The title color.
- `title.face`: The title face, such as "plain", "bold".
- `title.position`: The title position, such as "left", "middle", "right".
- `subtitle`: The subtitle of table
- `subtitle.size`: The subtitle size.
subtitle.color The subtitle color.
subtitle.face The subtitle face, such as "plain", "bold", default is "bold".
subtitle.position The subtitle position, such as "left", "middle", "right", default is "middle".
tile.color The color of table lines, default is 'white'.
tile.size The size of table lines, default is 1.
colname.size The size of colnames, default is 3.
colname.color The color of colnames, default is 'white'.
colname.face The face of colnames, default is 'bold'.
colname.fill.color The fill color of colnames, default is love_color("dark_cyan").
text.size The size of text, default is 3.
text.color The color of text, default is love_color("dark_grey").
text.face The face of text, default is 'plain'.
text.fill.color The fill color of text, default is c('white', love_color("pale_grey").

Examples

iv_list = get_psi_iv_all(dat = UCICreditCard[1:1000, ],
                        x_list = names(UCICreditCard)[3:5],
                        equal_bins = TRUE,
                        target = "default.payment.next.month", ex_cols = "ID|apply_date")
iv_dt = get_psi_iv(UCICreditCard, x = "PAY_3",
                   target = "default.payment.next.month", bins_total = TRUE)
plot_table(iv_dt)

plot_theme

Description

plot_theme is a simper wrapper of theme for ggplot2.

Usage

plot_theme(
    legend.position = "top",
    angle = 30,
    legend_size = 7,
    axis_size_y = 8,
    axis_size_x = 8,
    axis_title_size = 10,
    title_size = 11,
pred_score

title_vjust = 0,
title_hjust = 0,
linetype = "dotted",
face = "bold"
}

Arguments

legend.position
angle
legend.size
axis.size.y
axis.size.x
axis.title.size

title.size
title.vjust
title.hjust
linetype
face

Details

see details at: codetheme

Description

pred_score is for using logistic regression model model to predict new data.

Usage

pred_score(
  model,
  dat,
  x_list = NULL,
  bins_table = NULL,
  obs_id = NULL,
  miss_values = list(-1, "-1", "NULL", "-1", "-9999", "-9996", "-9997", "-9995",
                       "-9998", -9999, -9998, -9997, -9996, -9995),
  woe_name = FALSE
)
process_nas

Arguments

model Logistic Regression Model generated by `training_model`.
dat Dataframe of new data.
x_list Into the model variables.
bins_table a data.frame generated by `get_bins_table`.
obs_id The name of ID of observations or key variable of data. Default is NULL.
miss_values Special values.
woe_name Logical. Whether woe variable’s name contains ’woe’. Default is FALSE.

Value

new scores.

See Also

`training_model`, `lr_params`, `xgb_params`, `rf_params`

Description

`process_nas_var` is for missing value analysis and treatment using knn imputation, central imputation and random imputation. `process_nas` is a simpler wrapper for `process_nas_var`.

Usage

```r
process_nas(
  dat,
  x_list = NULL,
  class_var = FALSE,
  miss_values = list(-1, "missing"),
  default_miss = list(-1, "missing"),
  parallel = FALSE,
  ex_cols = NULL,
  method = "median",
  note = FALSE,
  save_data = FALSE,
  file_name = NULL,
  dir_path = tempdir(),
  ...
)

process_nas_var(
  dat = dat,
```
x,
missing_type = NULL,
method = "median",
nas_rate = NULL,
default_miss = list("missing", -1),
mat_nas_shadow = NULL,
dt_nas_random = NULL,
note = FALSE,
save_data = FALSE,
file_name = NULL,
dir_path = tempdir(),
...)

Arguments

dat A data.frame with independent variables.
x_list Names of independent variables.
class_var Logical, nas analysis of the nominal variables. Default is TRUE.
miss_values Other extreme value might be used to represent missing values, e.g:-1, -9999, -9998. These miss_values will be encoded to NA.
default_miss Default value of missing data imputation, Defualt is list(-1,'missing').
parallel Logical, parallel computing. Default is FALSE.
ex_cols A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
method The methods of imputation by knn. If "median", then Nas imputation with k neighbors median. If "avg_dist", the distance weighted average method is applied to determine the NAs imputation with k neighbors. If "default", assigning the missing values to -1 or "missing", otherwise ,processing the missing values according to the results of missing analysis.
note Logical, outputs info. Default is TRUE.
save_data Logical. If TRUE, save missing analysis to dir_path
file_name The file name for periodically saved missing analysis file. Default is NULL.
dir_path The path for periodically saved missing analysis file. Default is "./variable".
... Other parameters.
x The name of variable to process.
missing_type Type of missing. generated by codeanalysis_nas
nas_rate A list contains nas rate of each variable.
mat_nas_shadow A shadow matrix of variables which contain nas.
dt_nas_random A data.frame with random nas imputation.

Value

A dat frame with no NAs.
process_outliers

Examples

dat_na = process_nas(dat = UCICreditCard[1:1000,],
parallel = FALSE, ex_cols = "ID$", method = "median")

---

Process Outliers

Outliers Treatment

Description

`outliers_kmeans_lof` is for outliers detection and treatment using Kmeans and Local Outlier Factor (lof). `process_outliers` is a simpler wrapper for `outliers_kmeans_lof`.

Usage

```r
process_outliers(
  dat,
  target,
  ex_cols = NULL,
  kc = 3,
  kn = 5,
  x_list = NULL,
  parallel = FALSE,
  note = FALSE,
  process = TRUE,
  save_data = FALSE,
  file_name = NULL,
  dir_path = tempdir()
)
```

```r
outliers_kmeans_lof(
  dat,
  x,
  target = NULL,
  kc = 3,
  kn = 5,
  note = FALSE,
  process = TRUE,
  save_data = FALSE,
  file_name = NULL,
  dir_path = tempdir()
)
```

Arguments

dat Dataset with independent variables and target variable.

target The name of target variable.
psi_iv_filter

**ex_cols**  A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.

**kc**  Number of clustering centers for Kmeans

**kn**  Number of neighbors for LOF.

**x_list**  Names of independent variables.

**parallel**  Logical, parallel computing.

**note**  Logical, outputs info. Default is TRUE.

**process**  Logical, process outliers, not just analysis.

**save_data**  Logical. If TRUE, save outliers analysis file to the specified folder at dir_path.

**file_name**  The file name for periodically saved outliers analysis file. Default is NULL.

**dir_path**  The path for periodically saved outliers analysis file. Default is "./variable".

**x**  The name of variable to process.

**Value**

A data frame with outliers process to all the variables.

**Examples**

```
dat_out = process_outliers(UCICreditCard[1:10000,c(18:21,26)],
                           target = "default.payment.next.month",
                           ex_cols = "date$", kc = 3, kn = 10,
                           parallel = FALSE,note = TRUE)
```

---

**psi_iv_filter**  Variable reduction based on Information Value & Population Stability Index filter

**Description**

**psi_iv_filter** is for selecting important and stable features using IV & PSI.

**Usage**

```
psi_iv_filter(
    dat,
    dat_test = NULL,
    target,
    x_list = NULL,
    breaks_list = NULL,
    pos_flag = NULL,
    ex_cols = NULL,
    occur_time = NULL,
    best = FALSE,
    equal_bins = TRUE,
)```
g = 10,
sp_values = NULL,
tree_control = list(p = 0.05, cp = 1e-06, xval = 5, maxdepth = 10),
bins_control = list(bins_num = 10, bins_pct = 0.05, b_chi = 0.05, b_odds = 0.1, b_psi
  = 0.05, b_or = 0.15, mono = 0.3, odds_psi = 0.2, kc = 1),
oot_pct = 0.7,
psi_i = 0.1,
iv_i = 0.01,
cos_i = 0.7,
vars_name = FALSE,
note = TRUE,
parallel = FALSE,
save_data = FALSE,
file_name = NULL,
dir_path = tempdir(),
...)

Arguments

dat          A data.frame with independent variables and target variable.
dat_test     A data.frame of test data. Default is NULL.
target       The name of target variable.
x_list       Names of independent variables.
breaks_list  A table containing a list of splitting points for each independent variable. Default is NULL.
pos_flag     The value of positive class of target variable, default: "1".
ex_cols      A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
occur_time   The name of the variable that represents the time at which each observation takes place.
best         Logical, if TRUE, merge initial breaks to get optimal breaks for binning.
equal_bins   Logical, if TRUE, equal sample size initial breaks generates. If FALSE, tree breaks generates using desison tree.
g           Integer, number of initial bins for equal_bins.
sp_values    A list of missing values.
tree_control the list of tree parameters.
bins_control the list of parameters.
oot_pct      Percentage of observations retained for overtime test (especially to calculate PSI). Default is 0.7
psi_i        The maximum threshold of PSI. 0 <= psi_i <= 1; 0.05 to 0.2 usually work. Default: 0.1
iv_i         The minimum threshold of IV. 0 < iv_i ; 0.01 to 0.1 usually work. Default: 0.01
Description

This function is not intended to be used by end user.

Usage

\[ p_{ij}(x) \]

\[ e_{ij}(x) \]
Arguments

x       A numeric vector.

Value

A numeric vector of entropy.

Description

p_to_score is for transforming probability to score.

Usage

p_to_score(p, PDO = 20, base = 600, ratio = 1)

Arguments

p       Probability.
PDO     Point-to-Double Odds.
base    Base Point.
ratio   The corresponding odds when the score is base.

Value

new prob.

See Also

training_model, pred_score
**quick_as_df**

List data.frame quickly

---

**Description**

`quick_as_df` is a function for fast dataframe transformation.

**Usage**

```r
quick_as_df(df_list)
```

**Arguments**

- `df_list` A list of data.

**Value**

- packages installed and library.

**Examples**

```r
UCICreditCard = quick_as_df(UCICreditCard)
```

---

**ranking_percent_proc**  

*Ranking Percent Process*

---

**Description**

`ranking_percent_proc` is for processing ranking percent variables. `ranking_percent_dict` is for generating ranking percent dictionary.

**Usage**

```r
ranking_percent_proc(
  dat,
  ex_cols = NULL,
  x_list = NULL,
  rank_dict = NULL,
  pct = 0.01,
  parallel = FALSE,
  note = FALSE,
  save_data = FALSE,
  file_name = NULL,
  dir_path = tempdir(),
)```
... 
)

ranking_percent_proc_x(dat, x, rank_dict = NULL, pct = 0.01)

ranking_percent_dict(
  dat,
  x_list = NULL,
  ex_cols = NULL,
  pct = 0.01,
  parallel = FALSE,
  save_data = FALSE,
  file_name = NULL,
  dir_path = tempdir(),
  ...
)

ranking_percent_dict_x(dat, x = NULL, pct = 0.01)

Arguments

dat  A data.frame.
ex_cols Names of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
x_list A list of x variables.
rank_dict The dictionary of rank_percent generated by ranking_percent_dict.
pct Percent of rank. Default is 0.01.
parallel Logical, parallel computing. Default is FALSE.
note Logical, outputs info. Default is TRUE.
save_data Logical, save results in locally specified folder. Default is FALSE.
file_name The name for periodically saved rank_percent data file. Default is "dat_rank_percent".
dir_path The path for periodically saved rank_percent data file Default is "tempdir()"
...
Additional parameters.
x The name of an independent variable.

Value

Data.frame with new processed variables.

Examples

rank_dict = ranking_percent_dict(dat = UCICreditCard[1:1000,],
x_list = c("LIMIT_BAL", "BILL_AMT2", "PAY_AMT3"), ex_cols = NULL )
UCICreditCard_new = ranking_percent_proc(dat = UCICreditCard[1:1000,],
x_list = c("LIMIT_BAL", "BILL_AMT2", "PAY_AMT3"), rank_dict = rank_dict, parallel = FALSE)
Description

`read_data` is for loading data, formats like csv, txt, data and so on.

Usage

```r
read_data(  
  path,  
  pattern = NULL,  
  encoding = "unknown",  
  header = TRUE,  
  sep = "auto",  
  stringsAsFactors = FALSE,  
  select = NULL,  
  drop = NULL,  
  nrows = Inf  
)

check_data_format(path)
```

Arguments

- `path` : Path to file or file name in working directory & path to file.
- `pattern` : An optional regular expression. Only file names which match the regular expression will be returned.
- `encoding` : Default is "unknown". Other possible options are "UTF-8" and "Latin-1".
- `header` : Does the first data line contain column names?
- `sep` : The separator between columns.
- `stringsAsFactors` : Logical. Convert all character columns to factors?
- `select` : A vector of column names or numbers to keep, drop the rest.
- `drop` : A vector of column names or numbers to drop, keep the rest.
- `nrows` : The maximum number of rows to read.
reduce_high_cor_filter

*Filtering highly correlated variables with reduce method*

**Description**

`reduce_high_cor_filter` is function for filtering highly correlated variables with reduce method.

**Usage**

```r
reduce_high_cor_filter(
  dat,
  x_list = NULL,
  size = ncol(dat)/10,
  p = 0.95,
  com_list = NULL,
  ex_cols = NULL,
  cor_class = TRUE,
  parallel = FALSE
)
```

**Arguments**

- **dat**: A data.frame with independent variables.
- **x_list**: Names of independent variables.
- **size**: Size of variable group.
- **p**: Threshold of correlation between features. Default is 0.7.
- **com_list**: A data.frame with important values of each variable. eg : IV_list
- **ex_cols**: A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
- **cor_class**: Calculate categorical variables’s correlation matrix. Default is FALSE.
- **parallel**: Logical, parallel computing. Default is FALSE.

---

remove_duplicated

*Remove Duplicated Observations*

**Description**

`remove_duplicated` is the function to remove duplicated observations.
replace_value

Usage

```r
remove_duplicated(
  dat = dat,
  obs_id = NULL,
  occur_time = NULL,
  target = NULL,
  note = FALSE
)
```

Arguments

- **dat**: A data frame with x and target.
- **obs_id**: The name of ID of observations. Default is NULL.
- **occur_time**: The name of occur time of observations. Default is NULL.
- **target**: The name of target variable.
- **note**: Logical. Outputs info. Default is TRUE.

Value

A data frame

Examples

```r
datts = remove_duplicated(dat = UCICreditCard,
  target = "default.payment.next.month",
  obs_id = "ID", occur_time = "apply_date")
```

replace_value

Replace Value

Description

`replace_value` is for replacing values of some variables. `replace_value_x` is for replacing values of a variable.

Usage

```r
replace_value(
  dat = dat,
  x_list = NULL,
  x_pattern = NULL,
  replace_dat,
  MARGIN = 2,
  VALUE = if (MARGIN == 2) colnames(replace_dat) else rownames(replace_dat),
  RE_NAME = TRUE,
  parallel = FALSE
)```
replace_value_x(
    dat,
    x,
    replace_dat,
    MARGIN = 2,
    VALUE = if (MARGIN == 2) colnames(replace_dat) else rownames(replace_dat),
    RE_NAME = TRUE
)

Arguments

dat A data.frame.
x_list Names of variables to replace value.
x_pattern Regular expressions, used to match variable names.
replace_dat A data.frame contains value to replace.
MARGIN A vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names.
VALUE Values to replace.
RE_NAME Logical, rename the replaced variable.
parallel Logical, parallel computing. Default is TRUE.
x Name of variable to replace value.

Description

require_packages is function for librarying required packages and installing missing packages if needed.

Usage

require_packages(..., pkg = as.character(substitute(list(...))))

Arguments

... Packages need loaded
pkg A list or vector of names of required packages.

Value

packages installed and library.
Examples

```r
## Not run:
require_packages(data.table, ggplot2, dplyr)
## End(Not run)
```

---

**Description**

`re_code` searches for matches to argument pattern within each element of a character vector.

**Usage**

```r
re_code(x, codes)
```

**Arguments**

- `x`: Variable to recode.
- `codes`: A data.frame of original value & recode value.

**Examples**

```r
SEX = sample(c("F","M"),1000,replace = TRUE)
codes= data.frame(ori_value = c('F','M'), code = c(0,1) )
SEX_re = re_code(SEX,codes)
```

---

**Description**

`re_name` is for renaming variables.

**Usage**

```r
re_name(dat, oldname = c(), newname = c())
```

**Arguments**

- `dat`: A data frame with variables to rename.
- `oldname`: Old names of variables.
- `newname`: New names of variables.
Value

data with new variable names.

Examples

dt = re_name(dat = UCICreditCard, "default.payment.next.month", "target")
names(dt['target'])

---

### rf_params

**Random Forest Parameters**

#### Description

rf_params is the list of parameters to train a Random Forest using in training_model.

#### Usage

rf_params(ntree = 100, nodesize = 30, samp_rate = 0.5, tune_rf = FALSE, ...)

#### Arguments

- **ntree**
  - Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.
- **nodesize**
  - Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time). Note that the default values are different for classification (1) and regression (5).
- **samp_rate**
  - Percentage of sample to draw. Default is 0.2.
- **tune_rf**
  - A logical. If TRUE, then tune Random Forest model. Default is FALSE.
- **...**
  - Other parameters

#### Details

See details at: [https://www.stat.berkeley.edu/~breiman/Using_random_forests_V3.1.pdf](https://www.stat.berkeley.edu/~breiman/Using_random_forests_V3.1.pdf)

#### Value

A list of parameters.

#### See Also

training_model, lr_params, gbm_params, xgb_params
rowAny

Description
Functions for vector operation.

Usage
rowAny(x)
rowAllnas(x)
colAllnas(x)
colAllzeros(x)
rowAll(x)
rowCVs(x, na.rm = FALSE)
rowSds(x, na.rm = FALSE)
colSds(x, na.rm = TRUE)
rowMaxs(x, na.rm = FALSE)
rowMins(x, na.rm = FALSE)
rowMaxMins(x, na.rm = FALSE)
colMaxMins(x, na.rm = FALSE)
cnt_x(x)
sum_x(x)
max_x(x)
min_x(x)
avg_x(x)

Arguments
x    A data.frame or Matrix.
na.rm Logical, remove NAs.
Value

A data.frame or Matrix.

Examples

#any row has missing values
row_amy = rowAny(UCICreditCard[8:10])
#rows which is all missing values
row_na = rowAllnas(UCICreditCard[8:10])
#cols which is all missing values
col_na = colAllnas(UCICreditCard[8:10])
#cols which is all zeros
row_zero = colAllzeros(UCICreditCard[8:10])
#sum all numbers of a row
row_all = rowAll(UCICreditCard[8:10])
#caculate cv of a row
row_cv = rowCVs(UCICreditCard[8:10])
#caculate sd of a row
row_sd = rowSds(UCICreditCard[8:10])
#caculate sd of a column
col_sd = colSds(UCICreditCard[8:10])

Description

save_data is for saving a data.frame or a list fast.

Usage

save_data(
  ..., 
  files = list(...),
  file_name = as.character(substitute(list(...))),
  dir_path = getwd(),
  note = FALSE,
  as_list = FALSE,
  row_names = FALSE,
  append = FALSE
)

Arguments

... datasets
files A dataset or a list of datasets.
file_name The file name of data.
score_transfer

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir_path</td>
<td>A string. The dir path to save breaks_list.</td>
</tr>
<tr>
<td>note</td>
<td>Logical. Outputs info. Default is TRUE.</td>
</tr>
<tr>
<td>as_list</td>
<td>Logical. List format or data.frame format to save. Default is FALSE.</td>
</tr>
<tr>
<td>row_names</td>
<td>Logical. Retain rownames.</td>
</tr>
<tr>
<td>append</td>
<td>Logical. Append new data to old.</td>
</tr>
</tbody>
</table>

Examples

```r
save_data(UCICreditCard,"UCICreditCard", tempdir())
```

---

**score_transfer**

**Score Transformation**

**Description**

score_transfer is for transfer woe to score.

**Usage**

```r
score_transfer(
  model,
  tbl_woe,
  a = 600,
  b = 50,
  file_name = NULL,
  dir_path = tempdir(),
  save_data = FALSE
)
```

**Arguments**

- **model**: A data frame with x and target.
- **tbl_woe**: A data frame with woe variables.
- **a**: Base line of score.
- **b**: Numeric. Increased scores from doubling Odds.
- **file_name**: The name for periodically saved score file. Default is "dat_score".
- **dir_path**: The path for periodically saved score file. Default is "/data".
- **save_data**: Logical, save results in locally specified folder. Default is FALSE.

**Value**

A data frame with variables which values transferred to score.
Examples

# dataset splitting
sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
#rename the target variable
dat = re_name(dat, "default.payment.next.month", "target")
dat = data_cleaning(dat, target = "target", obs_id = "ID",
                  occur_time = "apply_date", miss_values = list("", -1))
#train_test_spliting
train_test = train_test_split(dat, split_type = "OOT", prop = 0.7,
                              occur_time = "apply_date")
dat_train = train_test$train
dat_test = train_test$test
#get breaks of all predictive variables
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "EDUCATION", "PAY_3", "PAY_2")
b breaks_list = get_breaks_all(dat = dat_train, target = "target",
                               x_list = x_list, occur_time = "apply_date", ex_cols = "ID",
                               save_data = FALSE, note = FALSE)
#woe transforming
train_woe = woe_trans_all(dat = dat_train,
                          target = "target",
                          breaks_list = breaks_list,
                          woe_name = FALSE)
test_woe = woe_trans_all(dat = dat_test,
                         target = "target",
                         breaks_list = breaks_list,
                         note = FALSE)
Formula = as.formula(paste("target", paste(x_list, collapse = " + "), sep = " ~ "))
set.seed(46)
lr_model = glm(Formula, data = train_woe[, c("target", x_list)], family = binomial(logit))
#get LR coefficient
dt_imp_LR = get_logistic_coef(lg_model = lr_model, save_data = FALSE)
bins_table = get_bins_table_all(dat = dat_train, target = "target",
                                 x_list = x_list, dat_test = dat_test,
                                 breaks_list = breaks_list, note = FALSE)
#score card
LR_score_card = get_score_card(lg_model = lr_model, bins_table, target = "target")
#scoring
train_pred = dat_train[, c("ID", "apply_date", "target")]
test_pred = dat_test[, c("ID", "apply_date", "target")]
train_pred$pred_LR = score_transfer(model = lr_model,
                                     tbl_woe = train_woe,
                                     save_data = FALSE)[, "score"]
test_pred$pred_LR = score_transfer(model = lr_model,
                                    tbl_woe = test_woe, save_data = FALSE)[, "score"]
Description

`select_best_class` & `select_best_breaks` are for merging initial breaks of variables using chi-square, odds-ratio, PSI, G/B index and so on. The `get_breaks` is a simpler wrapper for `select_best_class` & `select_best_class`.

Usage

```r
select_best_class(
  dat,
  x,
  target,
  breaks = NULL,
  occur_time = NULL,
  oot_pct = 0.7,
  pos_flag = NULL,
  bins_control = NULL,
  sp_values = NULL,
  ...
)

select_best_breaks(
  dat,
  x,
  target,
  breaks = NULL,
  pos_flag = NULL,
  sp_values = NULL,
  occur_time = NULL,
  oot_pct = 0.7,
  bins_control = NULL,
  ...
)
```

Arguments

- `dat` A data frame with `x` and `target`.
- `x` The name of variable to process.
- `target` The name of target variable.
- `breaks` Splitting points for an independent variable. Default is `NULL`.
- `occur_time` The name of the variable that represents the time at which each observation takes place.
- `oot_pct` The percentage of Actual and Expected set for PSI calculating.
- `pos_flag` The value of positive class of target variable, default: "1".
- `bins_control` the list of parameters.
  - `bins_num` The maximum number of bins. 5 to 10 usually work. Default: 10
select_best_class

- **bins_pct** The minimum percent of observations in any bins. 0 < bins_pct < 1, 0.01 to 0.1 usually work. Default: 0.02.
- **b_chi** The minimum threshold of chi-square merge. 0 < b_chi< 1; 0.01 to 0.1 usually work. Default: 0.02.
- **b_odds** The minimum threshold of odds merge. 0 < b_odds < 1; 0.05 to 0.2 usually work. Default: 0.1.
- **b_psi** The maximum threshold of PSI in any bins. 0 < b_psi < 1 ; 0 to 0.1 usually work. Default: 0.05.
- **b_or** The maximum threshold of G/B index in any bins. 0 < b_or < 1 ; 0.05 to 0.3 usually work. Default: 0.15.
- **odds_psi** The maximum threshold of Training and Testing G/B index PSI in any bins. 0 < odds_psi < 1 ; 0.01 to 0.3 usually work. Default: 0.1.
- **mono** Monotonicity of all bins, the larger, the more nonmonotonic the bins will be. 0 < mono < 0.5 ; 0.2 to 0.4 usually work. Default: 0.2.
- **kc** number of cross-validations. 1 to 5 usually work. Default: 1.

*sp_values* A list of special value.

... Other parameters.

**Details**

The following is the list of Reference Principles

- 1. The increasing or decreasing trend of variables is consistent with the actual business experience. (The percent of Non-monotonic intervals of which are not head or tail is less than 0.35)
- 2. Maximum 10 intervals for a single variable.
- 3. Each interval should cover more than 2
- 4. Each interval needs at least 30 or 1
- 5. Combining the values of blank, missing or other special value into the same interval called missing.
- 6. The difference of Chi effect size between intervals should be at least 0.02 or more.
- 7. The difference of absolute odds ratio between intervals should be at least 0.1 or more.
- 8. The difference of positive rate between intervals should be at least 1/10 of the total positive rate.
- 9. The difference of G/B index between intervals should be at least 15 or more.
- 10. The PSI of each interval should be less than 0.1.

**Value**

A list of breaks for x.

See Also

get_tree_breaks, cut_equal, get_breaks
**Examples**

```r
# equal sample size breaks
equ_breaks = cut_equal(dat = UCICreditCard[, "PAY_AMT2"], g = 10)

# select best bins
bins_control = list(bins_num = 10, bins_pct = 0.02, b_chi = 0.02,
b_odds = 0.1, b_psi = 0.05, b_or = 0.15, mono = 0.3, odds_psi = 0.1, kc = 1)
select_best_breaks(dat = UCICreditCard, x = "PAY_AMT2", breaks = equ_breaks,
target = "default.payment.next.month", occur_time = "apply_date",
sp_values = NULL, bins_control = bins_control)
```

**split_bins**

**Usage**

```r
split_bins(dat, x, breaks = NULL, bins_no = TRUE, as_factor = FALSE, labels = NULL,
use_NA = TRUE, char_free = FALSE)
```

---

**Description**

This function is not intended to be used by end user.

**Usage**

```r
sim_str(a, b, sep = "\_[.][A-Z]"")
```

**Arguments**

- `a`: A string
- `b`: A string
- `sep`: Seprater of strings. Default is "\_[.][A-Z]".

---

**split_bins**

**Description**

split_bins is for binning using breaks.

**Usage**

```r
split_bins(
  dat, 
  x, 
  breaks = NULL, 
  bins_no = TRUE, 
  as_factor = FALSE, 
  labels = NULL, 
  use_NA = TRUE, 
  char_free = FALSE 
)
```
split_bins_all

Arguments

- **dat**: A data.frame with independent variables.
- **x**: The name of an independent variable.
- **breaks**: Breaks for binning.
- **bins_no**: Number the generated bins. Default is TRUE.
- **as_factor**: Whether to convert to factor type.
- **labels**: Labels of bins.
- **use_NA**: Whether to process NAs.
- **char_free**: Logical, if TRUE, characters are not splitted.

Value

A data.frame with Bined x.

Examples

```r
bins = split_bins(dat = UCICreditCard, 
                  x = "PAY_AMT1", breaks = NULL, bins_no = TRUE)
```

Description

split_bins is for transforming data to bins. The split_bins_all function is a simpler wrapper for split_bins.

Usage

```r
split_bins_all( 
  dat, 
  x_list = NULL, 
  ex_cols = NULL, 
  breaks_list = NULL, 
  bins_no = TRUE, 
  note = FALSE, 
  return_x = FALSE, 
  char_free = FALSE, 
  save_data = FALSE, 
  file_name = NULL, 
  dir_path = tempdir(), 
  ...
)
```
split_bins_all

Arguments

- **dat**
  A data.frame with independent variables.
- **x_list**
  A list of x variables.
- **ex_cols**
  Names of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
- **breaks_list**
  A list contains breaks of variables. it is generated by code `get_breaks_all`, `codeget_breaks`
- **bins_no**
  Number the generated bins. Default is TRUE.
- **note**
  Logical, outputs info. Default is TRUE.
- **return_x**
  Logical, return data.frame containing only variables in x_list.
- **char_free**
  Logical, if TRUE, characters are not splitted.
- **save_data**
  Logical, save results in locally specified folder. Default is TRUE
- **file_name**
  The name for periodically saved woe file. Default is "dat_woe".
- **dir_path**
  The path for periodically saved woe file. Default is "./data"
- **...**
  Additional parameters.

Value

A data.frame with splitted bins.

See Also

- `get_tree_breaks`, `cut_equal`, `select_best_class`, `select_best_breaks`

Examples

```r
sub = cv_split(UCICreditCard, k = 30)[[1]]
Dat = UCICreditCard[sub,]
Dat = re_name(dat, "default.payment.next.month", "target")
Dat = data_cleansing(dat, target = "target", obs_id = "ID", occur_time = "apply_date", miss_values = list("", -1))

train_test = train_test_split(dat, split_type = "OOT", prop = 0.7, occur_time = "apply_date")

Dat_train = train_test$train
Dat_test = train_test$test
# get breaks of all predictive variables
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "EDUCATION", "PAY_3", "PAY_2")
breaks_list = get_breaks_all(dat = Dat_train, target = "target",
                            x_list = x_list, occur_time = "apply_date", ex_cols = "ID",
                            save_data = FALSE, note = FALSE)
# woe transform
train_bins = split_bins_all(dat = Dat_train, breaks_list = breaks_list, woe_name = FALSE)

test_bins = split_bins_all(dat = Dat_test, breaks_list = breaks_list, note = FALSE)
```
Automatic production of hive SQL

Description

Returns text parse of hive SQL.

Usage

```r
sql_hive_text_parse(
  sql_dt,
  key_sql = NULL,
  key_table = NULL,
  key_id = NULL,
  key_where = c("dt = date_add(current_date(),-1)"),
  only_key = FALSE,
  left_id = NULL,
  left_where = c("dt = date_add(current_date(),-1)"),
  new_name = NULL,
  ...
)
```

Arguments

- `sql_dt` The data dictionary has three columns: table, map and feature.
- `key_sql` You can write your own SQL for the main table.
- `key_table` Key table.
- `key_id` Primary key id.
- `key_where` Key table conditions.
- `only_key` Only key table.
- `left_id` Right table’s key id.
- `left_where` Right table conditions.
- `new_name` A string. Rename all variables except primary key with suffix ‘new_name’.
- `...` Other params.

Value

Text parse of hive SQL.
Examples

#sql_dt: table, map and feature
sql_dt = data.frame(table = c("table_1", "table_1", "table_1", "table_1", "table_1", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_2", "table_3", "table_3", "table_3", "table_3", "table_3"),
    map = c("all", "all", "all", "all", "all", "all", "all", "all", "all", "all", "all", "all", "all", "all", "all", "all", "all", "id_card_info", "id_card_info", "id_card_info", "mobile_info", "mobile_info", "mobile_info", "all", "all", "all", "all", "all"),
    feature = c("user_id", "real_name", "id_card_encode", "mobile_encode", "dt", "user_id", "type_code", "first_channel", "second_channel", "user_name", "user_sex", "user_birthday", "user_age", "card_province", "card_zone", "card_city", "city", "province", "carrier", "user_id", "biz_id", "biz_code", "apply_time", "dt"))

#sample 1
sql_hive_text_parse(sql_dt = sql_dt,
    key_sql = NULL,
    key_table = "table_2",
    key_where = c("user_sex = 'male'",
                  "user_age > 20"),
    only_key = FALSE,
    key_id = "user_id",
    left_id = "user_id",
    left_where = c("dt = date_add(current_date(),-1)",
                   "apply_time >= '2020-05-01' "
    ),
    new_name = "basic"
)

#sample 2
sql_hive_text_parse(sql_dt = subset(sql_dt),
    key_sql = "SELECT
        user_id,
        max(apply_time) as max_apply_time
    FROM table_3
WHERE dt = date_add(current_date(),-1)
    GROUP BY user_id",
    key_id = "user_id",
    left_id = "user_id",
    left_where = c("dt = date_add(current_date(),-1)"
    ),
    new_name = NULL)
Description

This function is not intended to be used by end user.

Usage

start_parallel_computing(parallel = TRUE)

Arguments

parallel A logical, default is TRUE.

Value

parallel works.

stop_parallel_computing

Stop parallel computing

Description

This function is not intended to be used by end user.

Usage

stop_parallel_computing(cluster)

Arguments

cluster Parallel works.

Value

stop clusters.
str_match

string match #’ str_match search for matches to argument pattern within each element of a character vector:

Description

string match #’ str_match search for matches to argument pattern within each element of a character vector:

Usage

str_match(pattern, str_r)

Arguments

pattern character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Coerced by as.character to a character string if possible. If a character vector of length 2 or more is supplied, the first element is used with a warning. missing values are allowed except for regexpr and gregexpr.

str_r a character vector where matches are sought, or an object which can be coerced by as.character to a character vector. Long vectors are supported.

Examples

original_nam = c("12mdd","11mdd","10mdd")
str_match(str_r = original_nam,pattern= "\d+")

sum_table

Summary table

Description

#’The sum_table includes both univariate and bivariate analysis and ranges from univariate statistics and frequency distributions, to correlations, cross-tabulation and characteristic analysis.

Usage

sum_table(dat, ..., x_s = as.character(substitute(list(...))), x_list = NULL)

Arguments

dat A data.frame with x and target.
...
x of dat
x_s A list of x.
x_list Names of dat.
Value

A list contains both category and numeric variable analysis.

Examples

```
sum_table(UCICreditCard)
sum_table(UCICreditCard,LIMIT_BAL,AGE,EDUCATION,SEX)
```

---

**term_tfidf**

**TF-IDF**

Description

The `term_filter` is for filtering stop_words and low frequency words. The `term_idf` is for computing idf(inverse documents frequency) of terms. The `term_tfidf` is for computing tf-idf of documents.

Usage

```
term_tfidf(term_df, idf = NULL)
term_idf(term_df, n_total = NULL)
term_filter(term_df, low_freq = 0.01, stop_words = NULL)
```

Arguments

- **term_df**: A data.frame with id and term.
- **idf**: A data.frame with idf.
- **n_total**: Number of documents.
- **low_freq**: Use rate of terms or use numbers of terms.
- **stop_words**: Stop words.

Value

A data.frame

Examples

```
term_df = data.frame(id = c(1,1,1,2,2,3,3,4,4,4,4,4,5,5,6,7,7,8,8,8,9,9,9,10,10,11,11,11,11,11,11,11,11,11),
                    terms = c('a','b','c','a','c','d','d','a','b','c','a','c','d','a','c','b','c','h','c','d','g','k','k'))
term_df = term_filter(term_df = term_df, low_freq = 1)
idf = term_idf(term_df)
tf_idf = term_tfidf(term_df,idf = idf)
```
Process time series data

Description

This function is used for time series data processing.

Usage

\[
\text{time\_series\_proc}(\text{dat, ID} = \text{NULL, group} = \text{NULL, time} = \text{NULL})
\]

Arguments

- \text{dat}: A data.frame contained only predict variables.
- \text{ID}: The name of ID of observations or key variable of data. Default is NULL.
- \text{group}: The group of behavioral or status variables.
- \text{time}: The name of variable which is time when behavior was happened.

Details

The key to creating a good model is not the power of a specific modelling technique, but the breadth and depth of derived variables that represent a higher level of knowledge about the phenomena under examination.

Examples

\[
\text{dat} = \text{data.frame(id} = \text{c(1,1,1,2,2,3,3,4,4,4,4,4,5,5,6,7,7,}
\text{8,8,9,9,9,10,10,11,11,11,11,11,11,11,11,11,11)),
\text{terms} = \text{c('a','b','c','a','c','d','d','a','c','}
\text{'b','c','a','e','d','a','c','}
\text{'d','a','e','f','b','c','f','b','}
\text{'c','h','h','i','c','d','g','k','k'),
\text{time} = \text{c(8,3,1,9,6,1,4,9,1,3,4,8,2,7,1,}
\text{3,4,1,8,7,2,5,7,8,8,2,1,5,7,2,7,3))}
\]

\[
\text{time\_series\_proc}(\text{dat} = \text{dat}, \text{ID} = \text{'id'}, \text{group} = \text{'terms',time} = \text{'time'})
\]

Time Format Transfering

Description

\text{time\_transfer} is for transferring time variables to time format.
time_variable

Usage

time_transfer(dat, date_cols = NULL, ex_cols = NULL, note = FALSE)

Arguments

dat A data frame
date_cols Names of time variable or regular expressions for finding time variables. Default is "DATE$|time$|date$|timestamp$|stamp$".
ex_cols Names of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
note Logical, outputs info. Default is TRUE.

Value

A data.frame with transferred time variables.

Examples

#transfer a variable.
dat = time_transfer(dat = lendingclub, date_cols = "issue_d")
class(dat[, "issue_d"])
#transfer a group of variables with similar name.
#transfer all time variables.
dat = time_transfer(dat = lendingclub[1:3], date_cols = "_d$")
class(dat[, "issue_d"])

time_variable

Description

This function is not intended to be used by end user.

Usage

time_variable(
  dat,
  date_cols = NULL,
  enddate = NULL,
  units = c("secs", "mins", "hours", "days", "weeks")
)

Arguments

dat A data.frame.
date_cols Time variables.
enddate End time.
units Units of diff_time, "secs", "mins", "hours", "days", "weeks" is available.
time_vars_process  Processing of Time or Date Variables

Description
This function is not intended to be used by end user.

Usage
```r
time_vars_process(
  df_tm = df_tm,
  x,
  enddate = NULL,
  units = c("secs", "mins", "hours", "days", "weeks")
)
```

Arguments
- **df_tm**: A data.frame
- **x**: Time variable.
- **enddate**: End time.
- **units**: Units of diff_time, "secs", "mins", "hours", "days", "weeks" is available.

----------

tnr_value  tnr_value

Description
tnr_value is for get true negative rate for a prob or score.

Usage
tnr_value(prob, target)

Arguments
- **prob**: A list of predict probability or score.
- **target**: Vector of target.

Value
True Positive Rate
Description

training_model Model builder

Usage

training_model(
    model_name = "mymodel",
    dat,
    dat_test = NULL,
    target = NULL,
    occur_time = NULL,
    obs_id = NULL,
    x_list = NULL,
    ex_cols = NULL,
    pos_flag = NULL,
    prop = 0.7,
    split_type = if (!is.null(occur_time)) "OOT" else "Random",
    preproc = TRUE,
    low_var = 0.99,
    missing_rate = 0.98,
    merge_cat = 30,
    remove_dup = TRUE,
    outlier_proc = TRUE,
    missing_proc = "median",
    default_miss = list(-1, "missing"),
    miss_values = NULL,
    one_hot = FALSE,
    trans_log = FALSE,
    feature_filter = list(filter = c("IV", "PSI", "COR", "XGB"), iv_cp = 0.02, psi_cp =
        0.1, xgb_cp = 0, cv_folds = 1, hopper = FALSE),
    algorithm = list("LR", "XGB", "GBM", "RF"),
    LR.params = lr_params(),
    XGB.params = xgb_params(),
    GBM.params = gbm_params(),
    RF.params = rf_params(),
    breaks_list = NULL,
    parallel = FALSE,
    cores_num = NULL,
    save_pmml = FALSE,
    plot_show = FALSE,
    vars_plot = TRUE,
    model_path = tempdir(),
    seed = 46,
Arguments

model_name: A string, name of the project. Default is "mymodel".
dat: A data.frame with independent variables and target variable.
dat_test: A data.frame of test data. Default is NULL.
target: The name of target variable.
occur_time: The name of the variable that represents the time at which each observation takes place. Default is NULL.
obs_id: The name of ID of observations or key variable of data. Default is NULL.
x_list: Names of independent variables. Default is NULL.
ex_cols: Names of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
pos_flag: The value of positive class of target variable, default: "1".
prop: Percentage of train-data after the partition. Default: 0.7.
split_type: Methods for partition. See details at: train_test_split.
preproc: Logical. Preprocess data. Default is TRUE.
low_var: Logical, delete low variance variables or not. Default is TRUE.
missing_rate: The maximum percent of missing values for recoding values to missing and non_missing.
merge_cat: merge categories of character variables that is more than m.
remove_dup: Logical, if TRUE, remove the duplicated observations.
outlier_proc: Logical, process outliers or not. Default is TRUE.
missing_proc: If logical, process missing values or not. If "median", then Nas imputation with k neighbors median. If "avg_dist", the distance weighted average method is applied to determine the NAs imputation with k neighbors. If "default", assigning the missing values to -1 or "missing", otherwise processing the missing values according to the results of missing analysis.
default_miss: Default value of missing data imputation, Defualt is list(-1,'missing').
miss_values: Other extreme value might be used to represent missing values, e.g: -9999, -9998. These miss_values will be encoded to -1 or "missing".
one_hot: Logical. If TRUE, one-hot_encoding of category variables. Default is FALSE.
trans_log: Logical, Logarithmic transformation. Default is FALSE.
feature_filter: Parameters for selecting important and stable features. See details at: feature_selector.
algorithm: Algorithms for training a model. list("LR", "XGB", "GBDT", "RF") are available.
LR.params: Parameters of logistic regression & scorecard. See details at: lr_params.
XGB.params: Parameters of xgboost. See details at: xgb_params.
GBM.params: Parameters of GBM. See details at: gbm_params.
RF.params Parameters of Random Forest. See details at: rf_params.
breaks_list A table containing a list of splitting points for each independent variable. Default is NULL.
parallel Default is FALSE.
cores_num The number of CPU cores to use.
save_pmml Logical, save model in PMML format. Default is TRUE.
plot_show Logical, show model performance in current graphic device. Default is FALSE.
vars_plot Logical, if TRUE, plot distribution ,correlation or partial dependence of model input variables. Default is TRUE.
model_path The path for periodically saved data file. Default is tempdir().
seed Random number seed. Default is 46.
... Other parameters.

Value
A list containing Model Objects.

See Also
train_test_split, data_cleansing, feature_selector, lr_params, xgb_params, gbm_params,
rf_params, fast_high_cor_filter, get_breaks_all, lasso_filter, woe_trans_all, get_logistic_coef,
score_transfer, get_score_card, model_key_index, ks_psi_plot, ks_table_plot

Examples
sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
x_list = c("LIMIT_BAL")
B_model = training_model(dat = dat,
model_name = "UCICreditCard",
target = "default.payment.next.month",
x_list = x_list,
occurs_time = NULL,
obs_id = NULL,

dat_test = NULL,

preproc = FALSE,
outlier_proc = FALSE,
missing_proc = FALSE,
feature_filter = NULL,
algorith $\text{m} = \text{list}("LR")$,
LR.params = lr_params(lasso = FALSE,
step_wise = FALSE,

score_card = FALSE),

breaks_list = NULL,
parallel = FALSE,
cores_num = NULL,
save_pmml = FALSE,
plot_show = FALSE,
train_lr

vars_plot = FALSE,
model_path = tempdir(),
seed = 46)

---

train_lr  Trainig LR model

Description

train_lr is for training the logistic regression model using in training_model.

Usage

train_lr(
  dat_train, 
  dat_test = NULL, 
  target, 
  x_list = NULL, 
  occur_time = NULL, 
  prop = 0.7, 
  tree_control = list(p = 0.02, cp = 1e-08, xval = 5, maxdepth = 10), 
  bins_control = list(bins_num = 10, bins_pct = 0.05, b_chisq = 0.02, b_odds = 0.1, b_psi = 0.03, b_or = 0.15, mono = 0.2, odds_psi = 0.15, kc = 1), 
  thresholds = list(cor_p = 0.8, iv_i = 0.02, psi_i = 0.1, cos_i = 0.6), 
  lasso = TRUE, 
  step_wise = TRUE, 
  best_lambda = "lambda.auc", 
  seed = 1234, 
  ...
)

Arguments

dat_train  data.frame of train data. Default is NULL.

dat_test  data.frame of test data. Default is NULL.

target  name of target variable.

x_list  names of independent variables. Default is NULL.

occur_time  The name of the variable that represents the time at which each observation takes place. Default is NULL.

prop  Percentage of train-data after the partition. Default: 0.7.

tree_control  the list of parameters to control cutting initial breaks by decision tree. See details at: get_tree_breaks

bins_control  the list of parameters to control merging initial breaks. See details at: select_best_breaks, select_best_class

thresholds  Thresholds for selecting variables.
train_test_split

- cor_p The maximum threshold of correlation. Default: 0.8.
- iv_i The minimum threshold of IV. 0.01 to 0.1 usually work. Default: 0.02
- psi_i The maximum threshold of PSI. 0.1 to 0.3 usually work. Default: 0.1.
- cos_i cos_similarity of positive rate of train and test. 0.7 to 0.9 usually work. Default: 0.5.

lasso Logical, if TRUE, variables filtering by LASSO. Default is TRUE.
step_wise Logical, stepwise method. Default is TRUE.
best_lambda Methods of best lambda standards using to filter variables by LASSO. There are 3 methods: ("lambda.auc", "lambda.ks", "lambda.sim_sign"). Default is "lambda.auc".
seed Random number seed. Default is 1234.

train_test_split

Description

train_test_split Functions for partition of data.

Usage

train_test_split(
  dat,
  prop = 0.7,
  split_type = "Random",
  occur_time = NULL,
  cut_date = NULL,
  start_date = NULL,
  save_data = FALSE,
  dir_path = tempdir(),
  file_name = NULL,
  note = FALSE,
  seed = 43
)

Arguments

dat A data.frame with independent variables and target variable.
prop The percentage of train data samples after the partition.
split_type Methods for partition.
  • "Random" is to split train & test set randomly.
  • "OOT" is to split by time for observation over time test.
train_xgb

- "byRow" is to split by rownumbers.

**occur_time**  The name of the variable that represents the time at which each observation takes place. It is used for "OOT" split.

**cut_date**  Time points for splitting data sets, e.g.: splitting Actual and Expected data sets.

**start_date**  The earliest occurrence time of observations.

**save_data**  Logical, save results in locally specified folder. Default is FALSE.

**dir_path**  The path for periodically saved data file. Default is "/data".

**file_name**  The name for periodically saved data file. Default is "dat".

**note**  Logical. Outputs info. Default is TRUE.

**seed**  Random number seed. Default is 46.

**Value**

A list of indices (train-test)

**Examples**

```r
train_test = train_test_split(lendingclub,
    split_type = "OOT", prop = 0.7,
    occur_time = "issue_d", seed = 12, save_data = FALSE)

dat_train = train_test$train

dat_test = train_test$test
```

**Description**

train_xgb is for training a xgb model using in training_model.

**Usage**

```r
train_xgb(
    seed_number = 1234,
    dtrain,
    nthread = 2,
    nfold = 1,
    watchlist = NULL,
    nrounds = 100,
    f_eval = "ks",
    early_stopping_rounds = 10,
    verbose = 0,
    params = NULL,
    ...
)
```
Arguments

- seed_number: Random number seed. Default is 1234.
- dtrain: Train-data of xgb.DMatrix datasets.
- nthread: Number of threads.
- nfold: Number of the cross validation of xgboost.
- nrounds: Max number of boosting iterations.
- f_eval: Customized evaluation function, "ks" & "auc" are available.
- early_stopping_rounds: If NULL, the early stopping function is not triggered. If set to an integer k, training with a validation set will stop if the performance doesn’t improve for k rounds.
- verbose: If 0, xgboost will stay silent. If 1, it will print information about performance.

... Other parameters

UCICreditCard  

UCI Credit Card data

Description

This research aimed at the case of customers’s default payments in Taiwan and compares the predictive accuracy of probability of default among six data mining methods. This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 24 variables as explanatory variables.

Format

A data frame with 30000 rows and 26 variables.

Details

- ID: Customer id
- apply_date: This is a fake occur time.
- LIMIT_BAL: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
- SEX: Gender (male; female).
- EDUCATION: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- MARRIAGE: Marital status (1 = married; 2 = single; 3 = others).
• AGE: Age (year) History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows:
  • PAY_0: the repayment status in September
  • PAY_2: the repayment status in August
  • PAY_3: ...
  • PAY_4: ...
  • PAY_5: ...
  • PAY_6: the repayment status in April The measurement scale for the repayment status is:
    -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months;...;8 = payment delay for eight months; 9 = payment delay for nine months and above. Amount of bill statement (NT dollar)
  • BILL_AMT1: amount of bill statement in September
  • BILL_AMT2: mount of bill statement in August
  • BILL_AMT3: ...
  • BILL_AMT4: ...
  • BILL_AMT5: ...
  • BILL_AMT6: amount of bill statement in April Amount of previous payment (NT dollar)
  • PAY_AMT1: amount paid in September
  • PAY_AMT2: amount paid in August
  • PAY_AMT3: ...
  • PAY_AMT4: ...
  • PAY_AMT5: ...
  • PAY_AMT6: amount paid in April
  • default.payment.next.month: default payment (Yes = 1, No = 0), as the response variable

Source

http://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

See Also

lendingclub
variable_process

Description
This function is not intended to be used by end user.

Usage
variable_process(add)

Arguments
add A data.frame

var_group_proc

Process group numeric variables

Description
This function is used for grouped numeric data processing.

Usage
var_group_proc(dat, ID = NULL, group = NULL, num_var = NULL)

Arguments
dat A data.frame contained only predict variables.
ID The name of ID of observations or key variable of data. Default is NULL.
group The group of behavioral or status variables.
um_var The name of numeric variable to process.

Examples
dat = data.frame(id = c(1,1,1,2,2,3,3,4,4,4,4,4,4,5,5,6,7,7,
8,8,9,9,10,10,11,11,11,11,11,11,11,11,11,11),
terms = c('a','b','c','a','c','d','d','a','c','b','c','a','c','d','a','c','d','a','c','b','c','h','h','i','c','d','g','k','k'),
time = c(8,3,1,9,6,1,4,9,1,3,4,8,2,7,1,
3,4,1,8,7,2,5,7,8,8,2,1,5,7,2,7,3))
time_series_proc(dat = dat, ID = 'id', group = 'terms', time = 'time')
**WOE Transformation**

**Description**

`woe_trans` is for transforming data to woe. The `woe_trans_all` function is a simpler wrapper for `woe_trans`.

**Usage**

```r
woe_trans_all(
  dat,
  x_list = NULL,
  ex_cols = NULL,
  bins_table = NULL,
  target = NULL,
  breaks_list = NULL,
  note = FALSE,
  save_data = FALSE,
  parallel = FALSE,
  woe_name = FALSE,
  file_name = NULL,
  dir_path = tempdir(),
  ...
)

woe_trans(
  dat,
  x,
  bins_table = NULL,
  target = NULL,
  breaks_list = NULL,
  woe_name = FALSE
)
```

**Arguments**

- **dat**: A data.frame with independent variables.
- **x_list**: A list of x variables.
- **ex_cols**: Names of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
- **bins_table**: A table contains woe of each bin of variables, it is generated by `codeget_bins_table_all`, `codeget_bins_table`.
- **target**: The name of target variable. Default is NULL.
- **breaks_list**: A list contains breaks of variables. it is generated by `codeget_breaks_all`, `codeget_breaks`.
- **note**: Logical, outputs info. Default is TRUE.
save_data Logical, save results in locally specified folder. Default is TRUE
parallel Logical, parallel computing. Default is FALSE.
woe_name Logical. Add ".woe" at the end of the variable name.
file_name The name for periodically saved woe file. Default is "dat_woe".
dir_path The path for periodically saved woe file. Default is ".data"
... Additional parameters.
x The name of an independent variable.

Value
A list of breaks for each variable.

See Also

get_tree_breaks, cut_equal, select_best_class, select_best_breaks

Examples

```r
sub = cv_split(UCICreditCard, k = 30)[[1]]
dat = UCICreditCard[sub,]
dat = re_name(dat, "default.payment.next.month", "target")
dat = data_cleansing(dat, target = "target", obs_id = "ID", occur_time = "apply_date",
    miss_values = list("", -1))

train_test = train_test_split(dat, split_type = "OOT", prop = 0.7,
    occur_time = "apply_date")

dat_train = train_test$train
dat_test = train_test$test

# get breaks of all predictive variables
x_list = c("PAY_0", "LIMIT_BAL", "PAY_AMT5", "EDUCATION", "PAY_3", "PAY_2")
breaks_list = get_breaks_all(dat = dat_train, target = "target",
    x_list = x_list, occur_time = "apply_date", ex_cols = "ID",
    save_data = FALSE, note = FALSE)

# woe transform
train_woe = woe_trans_all(dat = dat_train,
    target = "target",
    breaks_list = breaks_list,
    woe_name = FALSE)
test_woe = woe_trans_all(dat = dat_test,
    target = "target",
    breaks_list = breaks_list,
    note = FALSE)
```
**xgb_data**

**XGboost data**

**Description**

xgb_data is for prepare data using in training_model.

**Usage**

```r
xgb_data(
  dat_train,
  target,
  dat_test = NULL,
  x_list = NULL,
  prop = 0.7,
  occur_time = NULL
)
```

**Arguments**

- **dat_train**: data.frame of train data. Default is NULL.
- **target**: name of target variable.
- **dat_test**: data.frame of test data. Default is NULL.
- **x_list**: names of independent variables of raw data. Default is NULL.
- **prop**: Percentage of train-data after the partition. Default: 0.7.
- **occur_time**: The name of the variable that represents the time at which each observation takes place. Default is NULL.

**xgb_filter**

**Select Features using XGB**

**Description**

xgb_filter is for selecting important features using xgboost.

**Usage**

```r
xgb_filter(
  dat_train,
  dat_test = NULL,
  target = NULL,
  pos_flag = NULL,
  x_list = NULL,
  occur_time = NULL,
  prop = 0.7,
  occur_time = NULL
)
```
ex_cols = NULL,
xgb_params = list(nrounds = 100, max_depth = 6, eta = 0.1, min_child_weight = 1,
    subsample = 1, colsample_bytree = 1, gamma = 0, scale_pos_weight = 1,
    early_stopping_rounds = 10, objective = "binary:logistic"),
f_eval = "auc",
cv_folds = 1,
cp = NULL,
seed = 46,
vars_name = TRUE,
note = TRUE,
save_data = FALSE,
file_name = NULL,
dir_path = tempdir(),
...)

Arguments

dat_train A data.frame with independent variables and target variable.
dat_test A data.frame of test data. Default is NULL.
target The name of target variable.
pos_flag The value of positive class of target variable, default: "1".
x_list Names of independent variables.
occur_time The name of the variable that represents the time at which each observation takes place.
ex_cols A list of excluded variables. Regular expressions can also be used to match variable names. Default is NULL.
f_eval Customized evaluation function, "ks" & "auc" are available.
cv_folds Number of cross-validations. Default: 5.
cp Threshold of XGB feature’s Gain. Default is 1/number of independent variables.
seed Random number seed. Default is 46.
vars_name Logical, output a list of filtered variables or table with detailed IV and PSI value of each variable. Default is FALSE.
note Logical, outputs info. Default is TRUE.
save_data Logical, save results results in locally specified folder. Default is FALSE.
file_name The name for periodically saved results files. Default is "Feature_importance_XGB".
dir_path The path for periodically saved results files. Default is "/variable".
... Other parameters to pass to xgb_params.

Value

Selected variables.
\textbf{xgb\_params}  

\textbf{Description}  

\texttt{xgb\_params} is the list of parameters to train a XGB model using in \texttt{training\_model}. \texttt{xgb\_params\_search} is for searching the optimal parameters of xgboost, if any parameters of params in \texttt{xgb\_params} is more than one.

\textbf{Usage}  

\begin{verbatim}
xgb_params(
  nrounds = 1000,
  params = list(max_depth = 6, eta = 0.01, gamma = 0, min_child_weight = 1, subsample =
                1, colsample_bytree = 1, scale_pos_weight = 1),
  early_stopping_rounds = 100,
  method = "random_search",
  iters = 10,
  f_eval = "auc",
  nfold = 1,
  nthread = 2,
  ...
)

xgb_params_search(
  dat_train,
  target,
  dat_test = NULL,
  x_list = NULL,
  ...)
\end{verbatim}
prop = 0.7,
occur_time = NULL,
method = "random_search",
iters = 10,
nrounds = 100,
early_stopping_rounds = 10,
params = list(max_depth = 6, eta = 0.01, gamma = 0, min_child_weight = 1, subsample =
1, colsample_bytree = 1, scale_pos_weight = 1),
f_eval = "auc",
nfold = 1,
nthread = 2,
...)

Arguments

nrounds Max number of boosting iterations.
params List of contains parameters of xgboost. The complete list of parameters is available at: http://xgboost.readthedocs.io/en/latest/parameter.html
early_stopping_rounds If NULL, the early stopping function is not triggered. If set to an integer k, training with a validation set will stop if the performance doesn’t improve for k rounds.
method Method of searching optimal parameters."random_search","grid_search","local_search" are available.
iters Number of iterations of "random_search" optimal parameters.
f_eval Customized evaluation function,"ks" & "auc" are available.
nfold Number of the cross validation of xgboost
nthread Number of threads
... Other parameters
dat_train A data.frame of train data. Default is NULL.
target Name of target variable.
dat_test A data.frame of test data. Default is NULL.
x_list Names of independent variables. Default is NULL.
prop Percentage of train-data after the partition. Default: 0.7.
occur_time The name of the variable that represents the time at which each observation takes place.Default is NULL.

Value

A list of parameters.

See Also

training_model, lr_params, gbm_params, rf_params
Fuzzy String matching

**Description**
Fuzzy String matching

**Usage**
x %alike% y

**Arguments**
x A string.
y A string.

**Value**
Logical.

**Examples**
"xyz" %alike% "xy"

---

Fuzzy String matching

**Description**
Fuzzy String matching

**Usage**
x %islike% y

**Arguments**
x A string.
y A string.

**Value**
Logical.

**Examples**
"xyz" %islike% "yz$"
Index

* datasets
  ewm_data, 28
  lendingclub, 65
  UCICreditCard, 128
%alike%, 137
%islike%, 137

add_variable_process, 6
address_variable, 5
analysis_nas, 6, 91
analysis_outliers, 7
as_percent, 8
auc_value, 8
avg_x (rowAny), 105
char_cor (char_cor_vars), 9
char_cor_vars, 9, 30
char_to_num, 10
check_data_format (read_data), 99
checking_data, 11
city_variable, 11
city_variable_process, 12
cnt_x (rowAny), 105
cohort_plot (cohort_table_plot), 13
cohort_table_plot, 13
colAllnas (rowAny), 105
colAllzeros (rowAny), 105
colMaxMins (rowAny), 105
color_ramp_palette (plot_colors), 85
colSds (rowAny), 105
cor_heat_plot, 13
cor_plot, 14
cos_sim, 15
creditmodel (creditmodel-package), 5
creditmodel-package, 5
customer_segmentation, 15
cut_equal, 17, 42, 110, 113, 132
cv_split, 17
data_cleaning, 18, 124
data_exploration, 20
date_cut, 21
de_one_hot_encoding, 24, 82
de_percent, 25
derived_interval, 21
derived_partial_acf, 22
derived_pct, 22
derived_ts (derived_ts_vars), 23
derived_ts_vars, 23
digits_num, 25
e_ij (p_ij), 95
entropy_weight, 26
entry_rate_na, 19, 27
euclid_dist, 27
eval_auc, 28
eval_ks (eval_auc), 28
eval_lift (eval_auc), 28
eval_tnr (eval_auc), 28
ewm_data, 28
fast_high_cor_filter, 29, 124
feature_selector, 30, 35, 95, 123, 124, 135
fuzzy_cluster (fuzzy_cluster_means), 32
fuzzy_cluster_means, 32
gather_data, 33
gbm_filter, 32, 34, 95, 135
gbm_params, 36, 72, 104, 123, 124, 136
get_auc_ks_lambda, 37
get_bins_table, 54, 90, 131
get_bins_table (get_bins_table_all), 38
get_bins_table_all, 38, 131
get_breaks, 17, 57, 110, 113, 131
groupbreaks (get_breaks_all), 39
get_breaks_all, 17, 39, 57, 113, 124, 131
groupcorrelation_group, 30, 42
group_iv, 39, 45, 50, 52
group_iv (group_iv_all), 43
group_iv_all, 39, 43, 45, 50, 52
replace_value_x (replace_value), 101
require_packages, 102
rf_params, 37, 72, 90, 104, 124, 136
roc_plot (model_result_plot), 75
rowAll (rowAny), 105
rowAllnas (rowAny), 105
rowAny, 105
rowCVs (rowAny), 105
rowMaxMins (rowAny), 105
rowMaxs (rowAny), 105
rowMins (rowAny), 105
rowSds (rowAny), 105

save_data, 106
score_distribution_plot
  (model_result_plot), 75
score_transfer, 107, 124
select_best_breaks, 42, 44, 52, 71, 113,
  125, 132
select_best_breaks (select_best_class), 108
select_best_class, 42, 44, 52, 71, 108, 113,
  125, 132
select_cor_group
  (get_correlation_group), 42
select_cor_list
  (get_correlation_group), 42
sim_str, 111
split_bins, 111
split_bins_all, 112
sql_hive_text_parse, 114
start_parallel_computing, 115
stop_parallel_computing, 116
str_match, 117
sum_table, 117
sum_x (rowAny), 105

term_filter (term_tfidf), 118
term_idf (term_tfidf), 118
term_tfidf, 118
time_series_proc, 119
time_transfer, 119
time_variable, 120
time_vars_process, 121
tnr_value, 121
train_lr, 125
train_test_split, 123, 124, 126
train_xgb, 127

testing_model, 36, 37, 70, 72, 90, 96, 104,
  122, 125, 127, 133, 135, 136
UCICreditCard, 66, 128
var_group_proc, 130
variable_process, 130
woe_trans (woe_trans_all), 131
woe_trans_all, 124, 131
xgb_data, 128, 133
xgb_filter, 32, 35, 95, 133
xgb_params, 37, 72, 90, 104, 123, 124, 135,
  135
xgb_params_search (xgb_params), 135