Package ‘autoScorecard’

June 13, 2023

**Type** Package  
**Title** Fully Automatic Generation of Scorecards  
**Version** 0.3.0  
**Maintainer** Tai-Sen Zheng <jc3802201@gmail.com>  

**Description** Provides an efficient suite of R tools for scorecard modeling, analysis, and visualization. Including equal frequency binning, equidistant binning, K-means binning, chi-square binning, decision tree binning, data screening, manual parameter modeling, fully automatic generation of scorecards, etc. This package is designed to make scorecard development easier and faster.  

References include:  
2. Dong-feng Li(Peking University), Class PPT.  

**License** AGPL-3  
**Encoding** UTF-8  
**Imports** infotheo, ROCR, rpart, discretization, stats, graphics, grDevices, corrplot, ggplot2  

**RoxygenNote** 7.2.3  
**Depends** R (>= 2.10)  
**Suggests** knitr, rmarkdown  
**VignetteBuilder** knitr  
**NeedsCompilation** no  
**Author** Tai-Sen Zheng [aut, cre]  
**Repository** CRAN  
**Date/Publication** 2023-06-13 08:50:10 UTC

### R topics documented:

- auto_scorecard

---

1

2
auto_scorecard

Functions to Automatically Generate Scorecards

Description

Functions to Automatically Generate Scorecards

Usage

auto_scorecard(
  feature = accepts,
  key_var = "application_id",
  y_var = "bad_ind",
  sample_rate = 0.7,
  base0 = FALSE,
  points0 = 600,
  odds0 = 1/20,
  pdo = 50,
  k = 2,
  max_depth = 3,
  tree_p = 0.1,
  missing_rate = 0,
  single_var_rate = 1,
  iv_set = 0.02,
  char_to_number = TRUE,
  na.omit = TRUE
)
**auto_scorecard**

**Arguments**

- **feature**: A data.frame with independent variables and target variable.
- **key_var**: A name of index variable name.
- **y_var**: A name of target variable.
- **sample_rate**: Training set sampling percentage.
- **base0**: Whether the scorecard base score is 0.
- **points0**: Base point.
- **odds0**: Odds.
- **pdo**: Point-to Double Odds.
- **k**: Each scale doubles the probability of default several times.
- **max_depth**: Set the maximum depth of any node of the final tree, with the root node counted as depth 0. Values greater than 30 rpart will give nonsense results on 32-bit machines.
- **tree_p**: Meet the following conversion formula: minbucket = round( p*nrow( df )).Smallest bucket(rpart): Minimum number of observations in any terminal <leaf> node.
- **missing_rate**: Data missing rate, variables smaller than this setting will be deleted.
- **single_var_rate**: The maximum proportion of a single variable, the variable greater than the setting will be deleted.
- **iv_set**: IV value minimum threshold, variable IV value less than the setting will be deleted.
- **char_to_number**: Whether to convert character variables to numeric.
- **na.omit**: na.omit returns the object with incomplete cases removed.

**Value**

A list containing data, bins, scorecards and models.

**Examples**

```r
accepts <- read.csv(system.file("extdata", "accepts.csv", package = "autoScorecard" ))
auto_scorecard1 <- auto_scorecard( feature = accepts[1:2000,], key_var= "application_id", y_var = "bad_ind", sample_rate = 0.7, points0 = 600, odds0=1/20, pdo = 50, max_depth = 3, tree_p = 0.1, missing_rate = 0, single_var_rate = 1, iv_set = 0.02, char_to_number = TRUE , na.omit = TRUE)
```
**best_iv**  
*Calculate the Best IV Value for the Binned Data*

**Description**
Calculate the Best IV Value for the Binned Data

**Usage**
`best_iv(df, variable, bin, method, label_iv)`

**Arguments**
- `df`: A data.frame with independent variables and target variable.
- `variable`: Name of variable.
- `bin`: Name of bins.
- `method`: Name of method.
- `label_iv`: Name of IV.

**Value**
A data frame of best IV, including the contents of the bin, the upper bound of the bin, the lower bound of the bin, and all the contents returned by the `get_IV` function.

**Examples**
```r
accepts <- read.csv( system.file( "extdata", "accepts.csv", package = "autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
f_1 <- bins_unsupervised( df = feature , id="application_id" , label="bad_ind" ,
                         methods = c("k_means", "equal_width","equal_freq" ) , bin_nums=10 )
best1 <- best_iv( df=f_1 ,bin="Var_bins" , method = c('method') ,
                  variable= c( "variable" ) ,label_iv='miv' )
```

---

**best_vs**  
*The Combination of Two Bins Produces the Best Binning Result*

**Description**
The Combination of Two Bins Produces the Best Binning Result

**Usage**
`best_vs(df1, df2, variable = "variable", label_iv = "miv")`
binning_eqfreq

Arguments

df1  A binned data.
df2  A binned data.
variable  A name of X variable.
label_iv  A name of target variable.

Value

A data frame of best IV.

Examples

accepts <- read.csv(system.file( "extdata", "accepts.csv", package = "autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
all2 <- bins_tree(df = feature, key_var= "application_id", y_var= "bad_ind" , max_depth = 3, p = 0.1 )
f_1 <-bins_unsupervised( df = feature , id="application_id" , label="bad_ind" , methods = c("k_means", "equal_width","equal_freq" ) , bin_nums=10 )
best1 <- best_iv( df=f_1 ,bin=c('bins') , method = c('method') , variable= c( "variable" ) ,label_iv='miv' )
vs1 <- best_vs( df1 = all2[-c(3)], df2 = best1[-c(1:2)] ,variable="variable" ,label_iv='miv' )

binning_eqfreq  Equal Frequency Binning

Description

Equal Frequency Binning

Usage

binning_eqfreq(df, feat, label, nbins = 3)

Arguments

df  A data.frame with independent variables and target variable.
feat  A name of dependent variable.
label  A name of target variable.
nbins  Number of bins,default:3.

Value

A data frame, including the contents of the bin, the upper bound of the bin, the lower bound of the bin, and all the contents returned by the get_IV function.
### Examples

```r
accepts <- read.csv( system.file( "extdata", "accepts.csv", package ="autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
binning_eqfreq1 <- binning_eqfreq( df= feature, feat = 'tot_derog', label = 'bad_ind', nbins = 3)
```

### Description

Equal Width Binning

### Usage

```r
binning_eqwid(df, feat, label, nbins = 3)
```

### Arguments

- **df**: A data.frame with independent variables and target variable.
- **feat**: A name of dependent variable.
- **label**: A name of target variable.
- **nbins**: Number of bins, default: 3.

### Value

A data frame, including the contents of the bin, the upper bound of the bin, the lower bound of the bin, and all the contents returned by the get_IV function.

### Examples

```r
accepts <- read.csv( system.file( "extdata", "accepts.csv", package ="autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
binning_eqwid1 <- binning_eqwid( df = feature, feat = 'tot_derog', label = 'bad_ind', nbins = 3 )
```
The K-means Binning The k-means binning method first gives the center number, classifies the observation points using the Euclidean distance calculation and the distance from the center point, and then recalculates the center point until the center point no longer changes, and uses the classification result as the binning of the result.

Usage

binning_kmean(df, feat, label, nbins = 3)

Arguments

df  A data.frame with independent variables and target variable.
feat A name of index variable name.
label A name of target variable.
nbins Number of bins, default: 3.

Value

A data frame, including the contents of the bin, the upper bound of the bin, the lower bound of the bin, and all the contents returned by the get_IV function.

Examples

accepts <- read.csv(system.file("extdata", "accepts.csv", package = "autoScorecard"))
feature <- stats::na.omit(accepts[, c(1, 3, 7:23)])
ddd <- binning_kmean(df = feature, feat = 'loan_term', label = 'bad_ind', nbins = 3)

Chi-Square Binning Chi-square binning, using the ChiMerge algorithm for bottom-up merging based on the chi-square test.

Description

Chi-Square Binning Chi-square binning, using the ChiMerge algorithm for bottom-up merging based on the chi-square test.
Usage

bins_chim(df, key_var, y_var, alpha)

Arguments

df A data.frame with independent variables and target variable.
key_var A name of index variable name.
y_var A name of target variable.
alpha Significance level (discretization);

Value

A data frame, including the contents of the bin, the upper bound of the bin, the lower bound of the bin, and all the contents returned by the get_IV function.

Examples

accepts <- read.csv( system.file( "extdata", "accepts.csv" , package = "autoScorecard" ))
feature2 <- stats::na.omit( accepts[1:200,c(1,3,7:23)] )
al13 <- bins_chim( df = feature2 , key_var = "application_id", y_var = "bad_ind", alpha=0.1 )

bins_tree

Automatic Binning Based on Decision Tree

Usage

bins_tree(df, key_var, y_var, max_depth = 3, p = 0.1)

Arguments

df A data.frame with independent variables and target variable.
key_var A name of index variable name.
y_var A name of target variable.
max_depth Set the maximum depth of any node of the final tree, with the root node counted as depth 0. Values greater than 30 rpart will give nonsense results on 32-bit machines.
p Meet the following conversion formula: minbucket = round(p*nrow(df)).Smallest bucket(rpart):Minimum number of observations in any terminal <leaf> node.
**Value**

A data frame, including the contents of the bin, the upper bound of the bin, the lower bound of the bin, and all the contents returned by the `get_IV` function.

**Examples**

```r
accepts <- read.csv(system.file( "extdata", "accepts.csv", package = "autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
all2 <- bins_tree(df = feature, key_var= "application_id", y_var= "bad_ind"
, max_depth = 3, p = 0.1 )
```

---

**Description**

Unsupervised Automatic Binning Function By setting `bin_nums`, perform three unsupervised automatic binning

**Usage**

```r
bins_unsupervised(df, id, label, methods = c("k_means", "equal_width", "equal_freq"),
bin_nums)
```

**Arguments**

- `df` A data.frame with independent variables and target variable.
- `id` A name of index.
- `label` A name of target variable.
- `methods` Simultaneously calculate three kinds of unsupervised binning("k_means","equal_width","equal_freq"), the parameters only determine the final output result.
- `bin_nums` Number of bins.

**Value**

A data frame, including the contents of the bin, the upper bound of the bin, the lower bound of the bin, and all the contents returned by the `get_IV` function.
Examples

```r
comparison_two <- read.csv(system.file("extdata", "accepts.csv", package = "autoScorecard" ))
feature <- stats::na.omit(accepts[,c(1,3,7:23)])
f_1 <- bins_unsupervised( df = feature, id="application_id", label="bad_ind", methods = c("k_means", "equal_width","equal_freq" ), bin_nums=10 )
```

---

**comparison_two**

*Compare the Distribution of the Two Variable Draw box plots, cdf plot, QQ plots and histograms for two data.*

Description

Compare the Distribution of the Two Variable Draw box plots, cdf plot, QQ plots and histograms for two data.

Usage

```r
comparison_two(var_A, var_B, name_A, name_B)
```

Arguments

- **var_A**: A variable.
- **var_B**: A variable.
- **name_A**: The name of data A.
- **name_B**: The name of data B.

Value

No return value, called for side effects

Examples

```r
accepts <- read.csv(system.file("extdata", "accepts.csv", package = "autoScorecard" ))
comparison_two(var_A = accepts$purch_price , var_B = accepts$tot_rev_line ,
name_A = 'purch_price', name_B = "tot_rev_line"
)```
**comparison_two_data**  
_Compare the Distribution of the Two Data_

**Description**

Compare the Distribution of the Two Data

**Usage**

```r
comparison_two_data(df1, df2, key_var, y_var)
```

**Arguments**

- `df1`: A data.
- `df2`: A data.
- `key_var`: A name of index variable name.
- `y_var`: A name of target variable.

**Value**

No return value, called for side effects

**Examples**

```r
accepts <- read.csv( system.file( "extdata", "accepts.csv" , package = "autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
d = sort( sample( nrow( feature ), nrow( feature )*0.7))
train <- feature[d,]
test <- feature[-d,]
comparison_two_data( df1 = train , df2 = test ,
key_var = c("application_id","account_number"), y_var="bad_ind"  )
```

---

**data_detect**  
_Data Description Function_

**Description**

Data Description Function

**Usage**

```r
data_detect(df, key_var, y_var)
```
Arguments

- `df`: A data.
- `key_var`: A name of index variable name.
- `y_var`: A name of target variable.

Value

A data frame of data description.

Examples

```r
accepts <- read.csv(system.file("extdata", "accepts.csv", package = "autoScorecard" ))
aaa <- data_detect( df = accepts, key_var = c("application_id","account_number"), y_var = "bad_ind" )
```

Description

Data Filtering

Usage

```r
filter_var(
  df,
  key_var,
  y_var,
  missing_rate,
  single_var_rate,
  iv_set,
  char_to_number = TRUE,
  na.omit = TRUE
)
```

Arguments

- `df`: A data.frame with independent variables and target variable.
- `key_var`: A name of index variable name.
- `y_var`: A name of target variable.
- `missing_rate`: Data missing rate, variables smaller than this setting will be deleted.
- `single_var_rate`: The maximum proportion of a single variable, the variable greater than the setting will be deleted.
- `iv_set`: IV value minimum threshold, variable IV value less than the setting will be deleted.
**Description**

Function to Calculate IV Value

**Usage**

```r
get_IV(df, feat, label, E = 0, woeInf.rep = 1e-04)
```

**Arguments**

- `df`: A data.frame with independent variables and target variable.
- `feat`: A name of dependent variable.
- `label`: A name of target variable.
- `E`: Constant, should be set to [0,1], used to prevent calculation overflow due to no data in binning.
- `woeInf.rep`: Woe replaces the constant, and when woe is positive or negative infinity, it is replaced by a constant.

**Value**

A data frame including counts, proportions, odds, woe, and IV values for each stratum.

**Examples**

```r
accepts <- read.csv( system.file( "extdata", "accepts.csv",package = "autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
iv1 = get_IV( df= feature ,feat ="tot_derog" , label ="bad_ind" )
```
noauto_scorecard  

Manually Input Parameters to Generate Scorecards

Description
Manually Input Parameters to Generate Scorecards

Usage

noauto_scorecard(
  bins_card,
  fit,
  bins_woe,
  points0 = 600,
  odds0 = 1/19,
  pdo = 50,
  k = 2
)

Arguments

bins_card  Binning template.
fit  See glm stats.
bins_woe  A data frame of woe with independent variables and target variable.
points0  Base point.
odds0  odds.
pdo  Point-to Double Odds.
k  Each scale doubles the probability of default several times.

Value

A data frame with score ratings.

Examples

accepts <- read.csv( system.file( "extdata", "accepts.csv", package = "autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
d = sort( sample( nrow( feature ), nrow( feature )*0.7))
train <- feature[d,]
test <- feature[-d,]
treebins_train <- bins_tree( df = train, key_var = "application_id", y_var="bad_ind", max_depth=3, p=0.1)
woe_train <- rep_woe( df= train , key_var = "application_id", y_var = "bad_ind", tool = treebins_train ,var_label = "variable",col_woe = 'woe', lower = 'lower', upper = 'upper')
woe_test <- rep_woe( df = test , key_var ="application_id", y_var = "bad_ind", tool = treebins_train ,var_label= "variable",col_woe = 'woe', lower = 'lower', upper = 'upper')
**noauto_scorecard2**

    col_woe = 'woe', lower = 'lower', upper = 'upper' 
lg <- stats::glm( bad_ind ~ . , family = stats::binomial( link = 'logit' ) , data = woe_train )
lg_both <- stats::step( lg , direction = "both")
Score1 <- noauto_scorecard( bins_card= woe_test , fit =lg_both , bins_woe = treebins_train ,
points0 = 600 , odds0 = 1/20 , pdo = 50 )

---

**noauto_scorecard2**  
Manually Input Parameters to Generate Scorecards The basic score is dispersed into each feature score

**Description**

Manually Input Parameters to Generate Scorecards The basic score is dispersed into each feature score

**Usage**

    noauto_scorecard2(
      bins_card,
      fit,
      bins_woe,
      points0 = 600,
      odds0 = 1/19,
      pdo = 50,
      k = 3
    )

**Arguments**

- **bins_card**  
  Binning template.
- **fit**  
  See glm stats.
- **bins_woe**  
  Base point.
- **points0**  
  Odds.
- **odds0**  
  Point-to Double Odds.
- **pdo**  
  A data frame of woe with independent variables and target variable.
- **k**  
  Each scale doubles the probability of default several times.

**Value**

A data frame with score ratings.
Examples

accepts <- read.csv( system.file( "extdata", "accepts.csv", package = "autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
d = sort( sample( nrow(feature), nrow(feature)*0.7))
train <- feature[d,]
test <- feature[-d,]
treebins_train <- bins_tree( df = train, key_var = "application_id", y_var="bad_ind", max_depth=3, p=0.1)
woe_train <- rep_woe( df= train , key_var = "application_id", y_var = "bad_ind", 
tool = treebins_train , var_label = "variable",col_woo = 'woe', lower = 'lower', upper = 'upper')
woe_test <- rep_woe( df = test , key_var = "application_id", y_var = "bad_ind", 
tool = treebins_train , var_label = "variable", 
        col_woo = 'woe', lower = 'lower', upper = 'upper' )
lg <- stats::glm(bad_ind~., family = stats::binomial( link = 'logit' ), data = woe_train)
lg_both <- stats::step( lg , direction = "both")
Score2 <- noauto_scorecard2( bins_card= woe_test , fit =lg_both , bins_woo = treebins_train ,
                                points0 = 600 , odds0 = 1/20 , pdo = 50 )

plot_board <- read.csv( system.file( "extdata", "accepts.csv", package = "autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
d = sort( sample( nrow(feature), nrow(feature)*0.7))
train <- feature[d,]
test <- feature[-d,]
treebins_train <- bins_tree( df = train, key_var = "application_id", y_var="bad_ind", max_depth=3, p=0.1)
woe_train <- rep_woe( df= train , key_var = "application_id", y_var = "bad_ind", 
tool = treebins_train , var_label = "variable", 
        col_woo = 'woe', lower = 'lower', upper = 'upper' )
lg <- stats::glm(bad_ind~., family = stats::binomial( link = 'logit' ), data = woe_train)
lg_both <- stats::step( lg , direction = "both")
Score2 <- noauto_scorecard2( bins_card= woe_test , fit =lg_both , bins_woo = treebins_train ,
                                points0 = 600 , odds0 = 1/20 , pdo = 50 )

plot_board(label, pred)

Arguments

label A target variable.
pred A predictor variable.

Value

No return value, called for side effects.

Examples

accepts <- read.csv( system.file( "extdata", "accepts.csv", package = "autoScorecard" ))
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
d = sort( sample( nrow(feature), nrow(feature)*0.7))
train <- feature[d,]
test <- feature[-d,]
treebins_train <- bins_tree( df = train, key_var = "application_id", y_var="bad_ind", max_depth=3, p=0.1)
woe_train <- rep_woe( df= train , key_var = "application_id", y_var = "bad_ind", 

psi_cal

psi_cal  

PSI Calculation Function

Description

PSI Calculation Function

Usage

psi_cal(df_train, df_test, feat, label, nbins = 10)

Arguments

df_train  Train data.
df_test  Test data.
feat  A name of index variable name.
label  A name of target variable.

Value

A data frame of PSI.

Examples

accepts <- read.csv( system.file( "extdata", "accepts.csv", package = "autoScorecard" ) )
feature <- stats::na.omit( accepts[,c(1,3,7:23)] )
d = sort( sample( nrow( feature ), nrow( feature )*0.7))
train <- feature[d,]
test <- feature[-d,]
treebins_train <- bins_tree( df = train, key_var = "application_id", y_var="bad_ind", max_depth=3, p=0.1)
woe_train <- rep_woe( df= train , key_var = "application_id", y_var = "bad_ind", tool = treebins_train ,var_label = "variable",col_woo = 'woe', lower = 'lower' , upper = 'upper')
woe_test <- rep_woe( df = test , key_var = "application_id", y_var = "bad_ind", tool = treebins_train ,var_label = "variable",col_woo = 'woe', lower = 'lower', upper = 'upper' )
lg<-stats::glm(bad_ind~.,family=stats::binomial(link='logit'),data= woe_train)
lg_both<-stats::step(lg,direction = "both")
logit<-stats::predict(lg_both,woe_test)
woe_test$lg_both_p<-exp(logit)/(1+exp(logit))
plot_board( label= woe_test$bad_ind, pred = woe_test$lg_both_p )
rep_woe <- stats::step(lg, direction = "both")
Score_2 <- noauto_scorecard(bins_card = woe_test, fit = lg_both, bins_woe = treebins_train, points0 = 600, odds0 = 1/20, pdo = 50)
Score_1 <- noauto_scorecard(bins_card = woe_train, fit = lg_both, bins_woe = treebins_train, points0 = 600, odds0 = 1/20, pdo = 50)
psi_1 <- psi_cal(df_train = Score_1$data_score, df_test = Score_2$data_score, feat = "Var", label = "bad_ind", nbins = 10)

---

**rep_woe**

Replace Feature Data by Binning Template

**Description**

Replace Feature Data by Binning Template

**Usage**

rep_woe(df, key_var, y_var, tool, var_label, col_woe, lower, upper)

**Arguments**

- **df**: A data.frame with independent variables and target variable.
- **key_var**: A name of index variable name.
- **y_var**: A name of target variable.
- **tool**: Binning template.
- **var_label**: The name of the characteristic variable.
- **col_woe**: The name of the woe variable
- **lower**: The name of the binning lower bound.
- **upper**: The name of the binning upper bound.

**Value**

A data frame of woe

**Examples**

accepts <- read.csv(system.file("extdata", "accepts.csv", package = "autoScorecard" ))
feature <- stats::na.omit(accepts[,c(1,3,7:23)])
all2 <- bins_tree(df = feature, key_var = "application_id", y_var = "bad_ind", max_depth = 3, p = 0.1)
re2 <- rep_woe(df = feature, key_var = "application_id", y_var = "bad_ind", tool = all2, var_label = "variable", col_woe = 'woe', lower = 'lower', upper = 'upper')
Index

auto_scorecard, 2
best_iv, 4
best_vs, 4
binning_eqfreq, 5
binning_eqwid, 6
binning_kmean, 7
bins_chim, 7
bins_tree, 8
bins_unsupervised, 9
comparison_two, 10
comparison_two_data, 11
data_detect, 11
filter_var, 12
get_IV, 13
noauto_scorecard, 14
noauto_scorecard2, 15
plot_board, 16
psi_cal, 17
rep_woe, 18