目录

[ScoreCard 2](#_Toc32635)

[Classes 2](#_Toc21990)

[CardFlow 2](#_Toc6682)

[Functions 13](#_Toc14868)

[get\_X\_score 13](#_Toc2968)

[Bins 14](#_Toc23841)

[Functions 15](#_Toc18168)

[\_x1OptBin 15](#_Toc7178)

[OptBin 23](#_Toc6973)

[OptBin\_mp 32](#_Toc16347)

[\_x1FreqBin 33](#_Toc21264)

[FreqBin 36](#_Toc7993)

[FreqBin\_mp 39](#_Toc20119)

[get\_bins\_stats 40](#_Toc21856)

[\_x1trans\_woe\_value 41](#_Toc20744)

[trans\_woe\_value 42](#_Toc499)

[trans\_woe\_value\_mp 43](#_Toc18361)

[is\_cate\_bins 44](#_Toc24646)

[bins\_stats\_to\_IV 44](#_Toc16054)

[\_x1MonoSuggest 45](#_Toc14863)

[MonoSuggest 48](#_Toc5939)

[MonoSuggest\_mp 51](#_Toc6314)

[Cutter 51](#_Toc30283)

[Functions 52](#_Toc28999)

[value\_counts\_weight 52](#_Toc22247)

[value\_counts\_weight\_y 53](#_Toc11197)

[\_spec\_del 54](#_Toc16331)

[is\_spec\_value 54](#_Toc8415)

[\_is\_ascending 55](#_Toc26392)

[\_ext\_bins 55](#_Toc21632)

[\_is\_spec\_bin 56](#_Toc5825)

[freq\_cut 57](#_Toc4232)

[cut\_by\_bins 58](#_Toc452)

[freq\_cut\_data 59](#_Toc29131)

[cut\_array 60](#_Toc30768)

[sort\_label 62](#_Toc15491)

[eq\_bin 62](#_Toc6196)

[Category 63](#_Toc15764)

[Functions 63](#_Toc19660)

[cate\_to\_cateBin 63](#_Toc26377)

[cate\_to\_num 64](#_Toc11985)

[numInterval\_to\_cateBin 67](#_Toc2414)

[Reg\_Step\_Wise\_MP 68](#_Toc17258)

[Classes 69](#_Toc22763)

[LinearReg 69](#_Toc11065)

[LogisticReg 75](#_Toc9296)

[Index 83](#_Toc12214)

[Functions 83](#_Toc5494)

[AUC 83](#_Toc18578)

[KS 84](#_Toc19864)

[LIFTn 85](#_Toc16676)

[PSI\_by\_dat 86](#_Toc1433)

[PSI\_by\_dist 88](#_Toc9759)

[VIF 90](#_Toc22863)

[Lan 91](#_Toc9769)

# ScoreCard

## Classes

### CardFlow

Implement a set of fixed processes (see Competitive Product Comparison - Built-in Fixed Workflow). It divides the scorecard development process into 10 stages: data loading, equal frequency binning, feature pre-filtering, monotonic and U-shaped recommendations, SCF optimal binning, WOE conversion, feature filtering, model building, scorecard making, and report generation. The middle results will be automatically saved after each stage is completed. You can resume or update the previous results from any stage, and restarting the computer will not lose the results of the calculated steps.

For example: cardflow.start(start\_step=1,end\_step=10) executes all steps. After execution, if the user changes the configuration file, such as changing the feature screening conditions, which affects the results of step 7 and later, and needs to update 7-10, then just execute: cardflow.start(start\_step=7,end\_step=10), and the previous results do not need to be re-executed , even if the computer is restarted .

For most users, after the configuration file is configured, CardFlow is the only component you need to interact with when using SCF . The interaction method is very simple: cardflow.start(start\_step=a,end\_step=b)

#### \_\_init\_\_

\_\_init\_\_(self, config\_file=None, encoding='utf-8', \*\*cover\_conf)

Build the CardFlow instance. After success, the configuration items will be saved in the specified ${work\_space}/0\_conf.pkl. After the instance is successfully built, SCF will display the welcome message ~-♫♫~.~-♪♪~ and the product design concept, which means to inform the user: the configuration settings comply with the SCF specification. You can go listen to the music, and leave it to me. SCF will add a ♫ for each major version update, and a ♪ for each minor version update. Generally , when the user needs to modify the configuration file, it is a major version update , and when the configuration file does not need to be modified, it is a minor version update . So when there is an extra ♫ in the welcome message, the user needs to check the Change Log to confirm whether the newly added configuration items and their default values are applicable to the configuration files of the old project.

**Parameters**

----------

config\_file : str

Specify the location of the configuration file

None: Do not use a configuration file

Default: None

encoding : str

Configuration file encoding

Default: 'utf-8'

\*\*cover\_conf : dict

The configuration in cover\_conf will supplement and replace the configuration in the config\_file file.

#### do\_load\_datas

do\_load\_datas(self)

The first step of the SCF modeling process. It will divide the data into four uses: model\_data, psi\_data, oot\_data, and performance\_data. They correspond to the file directories model\_data\_file\_path, psi\_data\_file\_path, oot\_data\_file\_path, and performance\_data\_file\_path specified in the configuration. do\_load\_datas reads all the files in these four file directories and forms a nested dict data set structure.

dict<data usage, dict<data file name, DataFrame>> uses the name of the data file as the name of the dataset

The data is used for:

model\_data: data used for modeling, which can be classified into training sets, validation sets, and test sets

psi\_data: used to store data to check the stability of variables, such as when the performance period is insufficient but there is already data for X

oot\_data: used to store data for viewing model indicators on OOT

performance\_data: Data that needs to be used to view the performance of the model on the dataset can be included in this category

After the data is read, the nested dict data set structure will be saved in the specified ${work\_space}/1\_datas.pkl. It can also be obtained through the CardFlow instance scf.datas.

#### do\_freq\_bins

do\_freq\_bins(self)

The second step of the SCF modeling process. It will use the data set with the data purpose of model\_data and the user specified as train to calculate the split point of equal frequency bins, and a separate bin for special values. Then all data in all data purposes will be cut according to the calculated split point, and finally the information of each bin will be counted, including:

Quantity, distribution,For data with target, the number of sample points where the event did not occur, the number of sample points where the event occurred, and the event occurrence rate, woe,IV will also be counted.

For each data set, only the split point is consistent with the data specified as train, and all other information is calculated based on the data set itself.

After do\_freq\_bins runs successfully, two results are generated, which can be obtained through the CardFlow instances scf.train\_freqbins and scf.freqbins\_stat respectively. The first is the split node calculated based on the training set. The second is the statistical information of each box after all data are split according to this node. scf.freqbins\_stat is a two-level dict nested structure: dict<data purpose, dict<data name, equal frequency binning statistics>>. The two results are combined into a tuple and saved in ${work\_space}/2\_freqbins.pkl. For the convenience of user reading, the second result will also be written to ${work\_space}/2\_freqbins.xlsx. Each sheet is the binning statistics of a data set. The naming convention of the sheet is: 'data set file name ( ${ data purpose } )'

See [Bins.FreqBin](#_FreqBin) , [Bins.\_x1FreqBin](#__x1FreqBin)

#### do\_fore\_filter

do\_fore\_filter(self)

The third step of the SCF modeling process. When do\_fore\_filter runs successfully, two results will be generated, which can be called through the CardFlow instances scf.fore\_col\_indices and scf.fore\_filtered\_cols respectively. The first is the index value of the filter calculated for each variable based on the distribution of equal frequency bins, and the second is the filtered variables and the reasons for being filtered out. The format of the filtered out reason record is as follows:

[fore] filter metric > or < filter threshold [dataset name]

... (records deleted by multiple filters, line breaks)

[fore]: Indicates that the variable is deleted in the pre-filtering stage

Filter indicators: iv, homogeneity, miss, user extension

[Dataset Name]: The name of the dataset that produces the maximum or minimum value

These two results will form a tuple and be saved in ${work\_space}/3\_fore\_filter.pkl.

For more detailed information, see the description of the fore\_filters and filters configuration items in the configuration file conf\_doc\_XX.txt

#### do\_mono\_suggest

do\_mono\_suggest(self)

The fourth step of the SCF modeling process. If mono\_suggest=True is configured, SCF will use all data sets under the data purpose model\_data to calculate the monotonicity of variables at the data level and provide users with suggestions for monotonicity settings. The calculated results are:

L+: linear monotonically increasing, the larger the variable value, the higher the event rate

L-: Linear monotonically decreasing, the larger the variable value, the lower the event rate

Uu: U-shaped concave

Un: U-shaped convex (inverted U-shaped)

Unordered categorical variables do not require the recommended monotonicity constraints because the codes for unordered categories are themselves calculated using event rates.

After do\_mono\_suggest runs successfully, two results will be generated. The first is a monotonicity suggestion for all variables (only the first element in the suggestion is useful, and the remaining elements are for backward compatible expansion). The second is the event rate of each equal frequency bin of each data set. Users can use this result to check whether the suggested trend should be adopted. The two results can be obtained through the CardFlow instances scf.mono\_suggests and scf.mono\_suggests\_eventproba respectively. The two results will form a tuple and be saved in ${work\_space}/4\_mono\_suggest.pkl. For the convenience of user reading, the second result will also be written to ${work\_space}/4\_mono\_suggest.xlsx.

See [Bins.MonoSuggest](#_MonoSuggest) , [Bins.\_x1MonoSuggest](#__x1MonoSuggest)

#### do\_optim\_bins

do\_optim\_bins(self)

The fifth step of the SCF modeling process. It will use the data set specified as train in the data usage model\_data to calculate the split point of the SCF optimal bin (for the SCF optimal bin, see the introduction of the [Bins](#_Bins) module). Then all the data in all data usages will be cut according to the calculated split point, and finally the information of each bin will be counted, including:

Quantity, distribution,For data with target, the number of sample points where the event did not occur, the number of sample points where the event occurred, and the event occurrence rate, woe,IV will also be counted.

For each data set, only the split point is consistent with the data specified as train, and all other information is calculated based on the data set itself.

After do\_optim\_bins runs successfully, two results will be generated, which can be obtained through the CardFlow instances scf.train\_optbins and scf.optbins\_stat respectively. The first is the split node calculated based on the training set, and the second is the statistical information of each box after all data are split according to this node. scf.optbins\_stat is a two-level dict nested structure: dict<data purpose, dict<data name, SCF optimal binning statistics>>. These two results will form a tuple and be saved in ${work\_space}/5\_optbins.pkl. For the convenience of user reading, the second result will also be written to ${work\_space}/5\_optbins.xlsx. Each sheet is the binning statistics of a data set. The naming convention of the sheet is: 'data set file name (data purpose)'

See [Bins.OptBin](#_OptBin) , [Bins.\_x1OptBin](#__x1OptBin)

#### do\_woe

do\_woe(self)

The sixth step of the SCF modeling process. According to the mapping relationship between the binning of the training set and WOE specified by the user, all data under all data purposes are converted to WOE. The converted results can be obtained through scf.woes and saved in ${work\_space}/6\_woe.pkl.

scf.woes is a nested dict structure:

dict<data usage, dict<data file name, data set WOE value>>

#### do\_filter

do\_filter(self)

The seventh step of the SCF modeling process. When do\_filter runs successfully, 4 results will be generated. The first is the filter index value calculated based on the distribution of each variable based on the SCF optimal binning or the WOE value. The second is the filtered variables and the reasons for being filtered out. The format of the filtered out reason record is as follows:

filter\_metric>or<filter\_threshold[dataset\_name]

... (records deleted by multiple filters, line breaks)

Filter indicators: iv, homogeneity, miss, ivCoV, corr, psi, user extended

[Dataset Name]: On which dataset does the maximum or minimum value occur?

The third is the intermediate data in the indicator calculation process, which can be output to the model report so that users can deepen their understanding of the data. The fourth records the variables that the user is forced to retain and set. For the difference between retention and setting, refer to the description of the user\_save and user\_set configuration items in the configuration file conf\_doc\_XX.txt.

These four results will form a tuple and be saved in ${work\_space}/7\_filter.pkl.

For more detailed information, see the description of the filters configuration item in the configuration file conf\_doc\_XX.txt.

#### do\_model

do\_model(self)

The eighth step of the SCF modeling process is implemented by calling the built-in two-way stepwise logistic regression of SCF (see [Reg\_Step\_Wise\_MP](#_Reg_Step_Wise_MP) module).

After do\_model runs successfully, 4 results will be generated:

The first one is all the variables entered into the model

The second is the variable deleted by the model

The third one is the constructed model:

The main methods are:

predict(X) Model output prediction probability

The main properties are:

intercept\_ intercept term

coef\_ Coefficient of each variable (excluding the intercept term)

tvalues The t statistic of each model variable. Where const is the t statistic of the intercept term

pvalues Two-tailed P-value of each model variable

resid\_pearson Pearson residual

resid\_deviance residual deviation

The fourth is a summary of the model building information, mainly including: Link Function, No. Observations (sample size), Df Model, Df Residuals, Method (optimization algorithm), AIC, BIC, Log-Likelihood, LL-Null, Deviance, Pearson chi2, Scale

The fifth is the coefficient of each input variable, including the constant term, which contains the following information: Coef, Std.Err (standard error), Wald Chi-Square, P-Values, confidence interval [0.025 ~ 0.975], Standardized Coefficients (standardized coefficients)

The sixth is the reason for deleting each variable

The seventh is a detailed record of each round of modeling process, including: adding or removing variables, model performance indicators

They can be accessed through the CardFlow instances scf.in\_clf\_cols, scf.clf\_del\_cols, scf.clf, scf.clf\_perf, scf.clf\_coef, scf.del\_reason, and scf.step\_proc respectively.

These 7 results will form a tuple and be saved in ${work\_space}/8\_model.pkl. For the convenience of users, the results of the 4th, 5th, 6th, and 7th will also be written into ${work\_space}/8\_reg\_mstep.xlsx.

See [Reg\_Step\_Wise\_MP.LinearReg](#_LinearReg) , [Reg\_Step\_Wise\_MP.LogisticReg](#_LogisticReg)

#### do\_card

do\_card(self)

The 9th step of the SCF modeling process. A scorecard is constructed based on the generated model and WOE, which can be accessed through the CardFlow instance scf.card and saved in ${work\_space}/9\_card.pkl.

#### do\_report

do\_report(self)

After the run is completed, a ${work\_space}/{model\_name}\_Report.xlsx model report file will be generated. The file content includes:

1. Sample Y statistics: List the number of good and bad samples, the number of good and bad samples (weighted), the weighted event rate, and other information for all target-labeled data sets under all data uses

2. Suggested monotonicity: Gives the suggested monotonicity and user-set monotonicity for each variable. Lists the event rates of all data sets under model\_data after equal frequency binning, so that users can confirm the suggested monotonicity trend.

3. Variable selection: List the indicator values of each variable (the included indicators are set by the user through the filters configuration item), whether to enter the model, and the reason for deletion

4. Intermediate table of variable selection: When calculating variable selection, some useful intermediate results will be generated. These intermediate results will be output to the model report to facilitate users to deeply understand the details of variable selection

5. Scorecard: comparison table of binning and scoring of output variables, parameters of scorecard conversion, variable weights of standardized coefficient caliber, variable weights of extreme value caliber

6. Model performance: List the model indicators (including KS, AUC, LIFTn) of all target-labeled data sets under each data purpose, and the pivot table of scoring and frequency segmentation (including quantity, cumulative quantity, distribution, cumulative distribution, event incidence, cumulative event incidence, ODDS, cumulative ODDS, etc.)

7. Logistic regression model: Same as the content of ${work\_space}/8\_reg\_mstep.xlsx generated in step 8, see [the API documentation of CardFlow.do\_model](#_do_model)

8. Correlation coefficient: List the correlation coefficients between each input variable

9.VIF: List the VIF of each input variable

10. Equal frequency binning: The content of ${work\_space}/2\_freqbins.xlsx generated in step 2 is the same as that of ${work\_space}/2\_freqbins.xlsx, see [the API documentation of CardFlow.do\_freq\_bins](#_do_freq_bins)

11. SCF optimal binning: Same as the content of ${work\_space}/5\_optbins.xlsx generated in step 5, see [the API documentation of CardFlow.do\_optim\_bins](#_do_optim_bins)

Different contents belong to different sheets.

In addition to generating a model report file, a prediction result is also generated, which saves the (target, predicted value, sample weight) of all data sets under each data purpose, which can be accessed through the CardFlow instance scf.preds and saved in ${work\_space}/10\_preds.pkl.

#### redo\_bins

redo\_bins(self)

After configuring redo\_bins\_cols and redo\_type, call scf.redo\_bins(). It will recalculate the equal frequency and optimal binning for the variables in redo\_bins\_cols according to the current configuration in [BINS CONFIG], and update and save the mono\_suggest, Bins, and WOE of the variables involved. The update principle is:

1. If equal frequency binning is configured in redo\_type, redo\_bins modifies the equal frequency binning node of the variable in redo\_bins\_cols, then incrementally updates [the two results generated in the CardFlow. do\_freq\_bins step , and updates the file saved by](#_do_freq_bins) CardFlow. do\_freq\_bins in ${work\_space}. If the user has run step 4 [CardFlow. do\_mono\_suggest](#_do_mono_suggest) , redo\_bins modifies the monotonicity suggestion of the variable in redo\_bins\_cols, then incrementally updates [the results generated in the CardFlow. do\_mono\_suggest step, and updates the file saved by CardFlow. do\_mono\_suggest](#_do_mono_suggest) in ${work\_space}. Finally, a new ${work\_space}/redo\_freqbins\_Compare.xlsx file is generated to compare the changes before and after the equal frequency binning of the specified variables in all data sets under each data purpose.

2. If the configuration in redo\_type contains SCF optimal binning, redo\_bins modifies the SCF optimal binning node of the variable in redo\_bins\_cols, then incrementally updates [the two results generated in the CardFlow. do\_optim\_bins step, and updates the file saved by CardFlow. do\_optim\_bins](#_do_optim_bins) in ${work\_space}. If the user has already run step 6 [CardFlow. do\_woe](#_do_woe) , then modify the WOE value of the variable in redo\_bins\_cols, then incrementally update [the result generated in the CardFlow. do\_woe step, and update the file saved in ${work\_space} by CardFlow. do\_woe](#_do_woe) . A new ${work\_space}/redo\_optbins\_Compare.xlsx will also be generated to compare the changes before and after the optimal SCF binning of the specified variable in all data sets for each data purpose.

#### dat\_update

dat\_update(self, dat\_uses)

Reread all data sets under the specified data purpose and update the files saved in step [CardFlow.do\_load\_datas under ${work\_space} .](#_do_load_datas)

If you have completed step 2 [CardFlow. do\_freq\_bins](#_do_freq_bins) , then all data sets under the specified data purpose, namely scf.freqbins\_stat, will be updated, and step 2 [CardFlow.do\_freq\_bins](#_do_freq_bins) updates the files saved in ${work\_space}.

If you have completed step 5 [CardFlow. do\_optim\_bins](#_do_optim_bins) , then all data sets under the specified data purpose, namely scf.optbins\_stat, are updated, and step 5 [CardFlow. do\_optim\_bins](#_do_optim_bins) updates the files saved in ${work\_space}.

If you have completed step 6 [CardFlow. do\_woe](#_do_woe) , then all data sets under the specified data purpose, namely scf.woes, will be updated, and step 6 [CardFlow. do\_woe](#_do_woe) updates the files saved in ${work\_space}.

Note: dat\_update only changes the results of scf.freqbins\_stat or scf.optbins\_stat, but does not change scf.train\_freqbins and scf.train\_optbins. If you need to modify the data set, you also need to modify the binning nodes, i.e. scf.train\_freqbins and scf.train\_optbins. You can go through the process again, such as: scf.start(1,5), or use it with redo\_bins.

Parameters

----------

dat\_uses : list

Specifies the purpose of the data to be updated. It is a subset of ['model\_data', 'performance\_data', 'psi\_data', 'oot\_data']

#### start

start(self, start\_step=1, end\_step=10, load\_step=None)

After building CardFlow, call start to start the modeling process

The code corresponding to each step:

LOAD\_DATAS\_STEP=1

FREQ\_BINS\_STEP=2

FORE\_FILTER\_STEP=3

MONO\_SUGGEST\_STEP = 4

OPT\_BINS\_STEP = 5

WOE\_STEP=6

FILTER\_STEP=7

MODEL\_STEP=8

CARD\_STEP=9

REPORT\_STEP=10

example:

scf.start(start\_step=ScoreCard.LOAD\_DATAS\_STEP,end\_step=ScoreCard.REPORT\_STEP)

If you can remember the number of each step, you can also write: scf.start(start\_step=1,end\_step=10)

If you have calculated some steps, even if you restart the computer, start\_step does not need to start from 1. If you run scf.start(start\_step=1,end\_step=4), you can run scf.start(start\_step=5,end\_step=8) after restarting the computer.

Parameters

----------

start\_step : int

From which step to start running

Default: 1

end\_step : int

At which step does the operation end?

Default: 10.

load\_step : float

Load the specified step and all previous steps without recalculating. If some steps have not been calculated before, they will be calculated automatically.

If load\_step is not empty, the settings of start\_step and end\_step will be automatically ignored.

Advanced users may use SCF semi-automatically. This parameter is useful when interacting with SCF.

Default: None.

## Functions

### get\_X\_score

get\_X\_score(card, X, cores=None)

Returns the subscore and total score of X output in the given scorecard.

Parameters

----------

card : dict<var\_name,DataFrame(cols=['Bins','Points'])>

A comparison table of Bins and sub-scores for each variable.

Can be extracted via scf.card.

X : DataFrame

Data to be scored

cores : int

Number of CPU cores used

None: All CPUs

Returns

-------

scores : DataFrame

According to the card, each variable is converted into Bins and then into the corresponding sub-bins

total\_scores : Series

The total score of each sample point. It is also the sum of all sub-scores in each row of scores.

# Bins

Calculate the cut-off point of the optimal binning of SCF. The optimal cut-off point calculated by Bins is a global optimal analytical solution with mathematical proof. For categorical variables, including ordered and unordered categories, the global optimal analytical solution with mathematical proof can also be calculated. Its main functions are:

1. The global optimal solution can be found under unconstrained or constrained conditions. Constraints supported: monotonic constraints (automatically determine increasing or decreasing), monotonic decreasing constraints, monotonic increasing constraints, U-shaped constraints (automatically determine convex or concave), and automatically determined constraints (monotonically increasing, monotonically decreasing, convex U-shaped, concave U-shaped).

2. Find the global optimal solution for ordered categorical variables under unconstrained and constrained conditions.

3. Use "minimum difference in event rates between adjacent bins" instead of "information gain" or "chi-square value" to suppress the formation of bins with too small differences. Users can intuitively feel the size of the difference between bins. Categorical variables also support this function.

4. Do not replace the minimum value of the first bin with negative infinity, and do not replace the maximum value of the last bin with positive infinity. The significance of this is that the ignored abnormal values will not be covered up due to the extension of extreme values to infinity. At the same time, ScoreConflow provides a complete mechanism to deal with the problem of online values exceeding the modeling boundary value. This solves the common contradiction between discovering special values as early as possible during data analysis and covering up special values during online applications (ensuring that the process will not be interrupted but timely alarms are required).

5. Introduce the concept of wildcards to solve the problem that the online values of categorical variables exceed the modeling range.

6.Support multi-process parallel computing.

7. Support binning of weighted samples.

8.Support special value merging.

In most cases, users do not need to interact directly with the Bins module, and the ScoreCard module will automatically call the Bins module based on the configuration file. However, because ScoreConflow is a plug-in component design, advanced users can use the Bins module separately like any python module.

## Functions

### \_x1OptBin

\_x1OptBin(x\_dats,y\_dats,y\_label={'unevent':0,'event':1},weight\_dats=None,train\_name=None,

mono='N',sgst\_mono=None,distr\_min=0.02,rate\_gain\_min=0.001,bin\_cnt\_max=None,

spec\_value=[],spec\_distr\_min=None,default\_spec\_distr\_min=None,

spec\_comb\_policy={},default\_spec\_comb\_policy='N',

is\_cate=False,is\_order=False,no\_wild\_treat='M',

order\_list=None,unorder\_combine\_thv=None,

cust\_bin=None)

Performs global optimal binning on a variable. Supports multiple features, please refer to the introduction of [Bins](#_Bins) module

Parameters

----------

x\_dats : Series or dict<dat\_name,Series>. Required

A column of variables or the same column of variables belonging to different data sets

There are a few points to note:

1. When mono='A', SCF automatically selects a monotonicity constraint from L+, L-, Uu, Un. When automatically selecting a constraint, different data sets will be used to verify whether the constraint is correct.

2. If the user has already calculated the monotonicity of the suggestion elsewhere, the result can be passed directly to the \_x1OptBin function through the parameter sgst\_mono. When mono='A' is set, the calculation will not be repeated, wasting the user's waiting time.

According to the above description: If mono='A' and sgst\_mono is None, the type of x\_dats must be dict for calculating and verifying monotonicity trend. In other cases, the type of x\_dats can be Series or dict

y\_dats : Series or dict<dat\_name,Series>. Required

The actual target. For details on when to pass in Series or dict, see x\_dats. Its type must be consistent with x\_dats.

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Example: {'unevent':'good','event':'bad'}

Generally, the things that users care about most are defined as events, which are easier to explain. For example, when you want to emphasize the occurrence of lung cancer, you can say that the incidence of lung cancer among smokers is 50% higher than that among non-smokers. In this case, you can write {'unevent':'No lung cancer','event':'Lung cancer'}. If you write {'unevent':'Lung cancer','event':'No lung cancer'}, although it does not affect the use of the model, the explanation will become that the incidence of lung cancer among smokers is 50% lower than that among non-smokers. Obviously, the first way of expression is easier to understand.

Default: {'unevent':0,'event':1}.

weight\_dats : Series or dict<dat\_name,Series>

The weight of the sample. Samples with weights can also solve the global optimal binning point.

None: All weights are 1

When it is not None, you can refer to x\_dats to see when Series or dict is passed in. Its type must be consistent with x\_dats

Default: None

train\_name : str or None

When mono='A' and sgst\_mono=None, this parameter specifies which dataset in x\_dats is used to calculate the monotonicity trend. Except for the dataset corresponding to train\_name, all other datasets are used to verify the calculated monotonicity trend.

Default: None

If x\_dats is a dict, train\_name cannot be None.

mono : str

Monotonicity constraints for global optimal binning.

Value range:

N: IV value is the highest globally, no constraints

A: Automatically select a constraint from L+, L-, Uu, Un. Under this constraint, the IV value is the highest globally.

L: The IV value is globally the highest under linear monotone constraints (automatically determines L+, L-)

L+: The IV value is globally the highest under the linear monotonically increasing constraint

L-: The IV value is globally the highest under the linear monotonically decreasing constraint

U: The IV value is the highest globally under the U-type constraint (automatically determines Uu, Un)

Default: 'N'.

sgst\_mono:tuple(str,\*\*)

tuple (monotonicity constraint, some backwards-compatible information). The first element of the tuple is the recommended constraint: possible values are L+, L-, Uu, Un. The remaining elements are some built-in extended information.

Suggested constraint: If mono is set to 'A' and sgst\_mono is not None, SCF automatically sets mono to the first element of sgst\_mono to avoid repeated calculations.

When using SCF automation, this parameter can be calculated repeatedly through internal mechanisms. When using the Bins module alone, sgst\_mono can directly use the default value.

Default: None

distr\_min : float

Minimum distribution ratio for each bin

None: Do not set a minimum distribution ratio for each box of the variable

Note: Setting the minimum distribution ratio of a bin is not necessary to find the global optimal split point, but is determined by the user's confidence in the stability of the bin. If there are few sample points in a bin, the random fluctuation of its event rate may be relatively large, resulting in relatively large model fluctuations.

Default: 0.02.

rate\_gain\_min : float

The event rate between any two adjacent bins cannot be less than rate\_gain\_min. Some software packages often use information gain or chi-square value to suppress the formation of bins with too small differences. However, these parameters do not have a specific and intuitive concept for users, and it is impossible to infer how small the difference is. Therefore, the event rate, an intuitive indicator, is used here to suppress the formation of bins with too small differences.

Default: 0.001.

bin\_cnt\_max : float

The maximum number of bins for a variable. Bins for special values are not counted. If special values are merged into normal value bins due to merging rules, the merged bins are normal bins.

Because you can specify parameters such as the monotonicity of the variable, the minimum bin ratio, and the minimum difference in event rates, these parameters can automatically adjust the number of bins (usually the better the variable effect and the more evenly distributed it is, the more bins there are, and vice versa), so usually this parameter can be set to None.

If the variables are evenly distributed and well ordered, the number of bins may be large. Although this is in line with the actual situation, users can also specify the maximum number of bins allowed for certain business considerations.

None: Do not set a maximum number of bins for the variable

Default: None.

spec\_value : list

Special value range

Example. ["{-9997}","{-9999,-9998}"]

"{... , ...}" will not be parsed as a set, but will be processed as a string. {} in special values expresses a discrete value space symbol.

Let's take an example to explain the meaning of the expression:

"{-9997}": When the variable value is -9997, a special meaning occurs. For example, the number of court executions, -9997 may mean that the ID card is not in the citizen database, rather than being executed -9997 times. Through this example, users can clearly feel the difference in meaning between -9997 and the values 0, 1, and 2.

"{-9998,-9999}": When the variable value is -9998 or -9999, a special meaning occurs. Although the two meanings are different, for the business modeled this time, the two meanings can be regarded as the same meaning and processed according to the same business logic. For example, when collecting data, the data that was not collected due to Party A's reasons is marked as -9998, and the data that was not collected due to Party B's reasons is marked as -9999. However, for the business, these two values mean that the data is randomly missing, so they are both processed according to the logic of random missing. In this way, the value convention of the original data can be retained for backtracking, and the user can be exempted from the work of writing additional code when processing data.

"{None}" is a special value of null, which means: a special value with null value. The reason why {None} is used instead of {miss} is that the mechanism for generating null values is sometimes different from that for generating missing values. Missing values represent that the sample points are not collected due to some uncontrollable reasons during the sampling process, resulting in missing data information, such as network disconnection during data transmission and data not being collected due to equipment failure. This is a random missing. In addition to random missing, null values may also be caused by non-information missing, such as no loan record, no need for a certain examination due to health reasons, and the temperature is too low for the equipment to collect, etc. The null value itself is information. Do not mix null values with information and random missing null values into one special value.

If a variable is not configured with a null special value, but contains a null value, a {None} group is automatically generated to contain the null value of the variable.

default:[].

spec\_distr\_min : dict

Specify a minimum percentage for each special value of the variable. If the distribution percentage of a special value bin is less than the value specified by ${spec\_distr\_min}, the special value bin will be merged with other bins. For specific merging rules, see the spec\_comb\_policy setting. For special values that are not covered, default\_spec\_distr\_min is used.

None: Do not set a minimum distribution percentage for any special value of the variable. If default\_spec\_distr\_min is also set to None, no limit is imposed on the minimum percentage of special value bins.

Default: None.

default\_spec\_distr\_min : float

If the special value of a variable is not configured in ${spec\_distr\_min}, the default minimum distribution ratio of the special value

None: Do not set the default value for the minimum percentage of special value distribution.

Default: None.

spec\_comb\_policy : dict

When the proportion of special values of a variable is less than the threshold specified by ${spec\_distr\_min}, the merge strategy to be adopted can be:

A:auto finds the closest eventProb among all values

a:auto only finds the closest eventProb among non-special values

F:first merges with the first bin of non-special values

L:last merges with the last bin of non-special values

M:median merges with the middle bin of non-special values (if there are an even number of bins, merge with the bin with the high event rate)

m:median merges with the middle bin of non-special values (if there are an even number of bins, merge with the bin with a low event rate)

B: max Probability is merged with the bin with the largest eventProb

S: min Probability merges with the bin with the smallest eventProb

N: Do not merge

If there is a special value that is not covered, ${default\_spec\_comb\_policy} is used.

ex. {"{-9997}":L,"{-9998,-9999}":"N"}

Note: Let's use an example to explain the meaning of special value writing. The meaning of ["{-9997}","{-9998,-9999}"] is: there are three special values in the variable -9997, -9998, -9999. According to the business meaning, they are divided into two business groups "{-9997}","{-9998,-9999}". -9997 itself becomes a business group, -9998, -9999 form a business group, and because the two business groups "{-9997}" and "{-9998,-9999}" meet the special value merging rules set during binning, they are forcibly merged together at the data level to form a bin ["{-9997}","{-9998,-9999}"]. This kind of merging is different from merging -9998, -9999 into a business group. The merging mentioned in the business group is at the business level and is determined by understanding the business. The merging of ["{-9997}","{-9998,-9999}"] is at the data level and is determined only by calculating the event rate. The process and meaning behind how the three special values -9997, -9998, -9999 become two special value business groups "{-9997}","{-9998,-9999}" and finally become a special value merged bin ["{-9997}","{-9998,-9999}"] must be clear. This special value processing method is in line with statistical principles.

default:{}.

default\_spec\_comb\_policy : str

When ${spec\_comb\_policy} does not contain a special value, the default merge policy of the special value

For the value range, see ${spec\_comb\_policy}

Default: 'N'.

is\_cate : bool

Indicates whether the variable is categorical.

True: Categorical variable

False: continuous variable

Default: False.

is\_order : bool

Indicates whether the variable is an ordered categorical variable.

If is\_order=True, is\_cate will be automatically set to True

True: ordered categories

False: When is\_cate=True, it is an unordered category, and the unordered category will be globally optimally binned according to the order of event occurrence rates.

Default: False.

no\_wild\_treat : str

When a category variable does not have a wildcard and uncovered categories appear, the processing methods include:

L: Considered equal to the lowest category in the order

H: Considered equal to the highest order category

M: Considered equal to the middle category of the sequence (take the larger value when even)

m: Considered equal to the middle category of the sequence (smaller value if even)

None: No processing is performed on uncovered categories that appear in the variable

Default: 'M'

order\_list : tuple

Set the order of each value in the ordered categorical variable. The order of the tuple is the nominal order. If is\_order=True and order\_list=None, the lexicographic order of the characters is used as the order.

For ordered variables, adjacent nominal sequences can only appear in the same bin or at the beginning and end of adjacent bins.

If the nominal order is inconsistent with the event rate order, you can decide for yourself whether to configure the variable as an ordered variable or an unordered variable based on the business situation.

Supports globally optimal binning for ordered variables.

All categories that do not appear in the configuration (excluding special values) are collectively called wildcard categories, represented by \*\*. Values not covered in the training set may be seen in other datasets, and these categories are also put into wildcard categories.

When lexicographic order is used as the order, there is no wildcard category

Example: ("v1","\*\*","v2")

Default: None.

unorder\_combine\_thv : float

Set a threshold for unordered categorical variables and merge categories with distribution proportions less than the threshold into wildcard categories. If is\_cate=True and unorder\_combine\_thv=None, categories with too small frequency of the variable will not be wildcarded.

In other datasets, it is possible to see values not covered in the training set, and these categories are also put into the wildcard category.

Default: None.

cust\_bin : list

User-defined binning.

For example: ['[1.0,3.0)','[6.0,9.0)','[3.0,6.0)','{-999,-888}','{-997}','[9.0,10.0]','{-1000,None}']. The values do not need to be written in the order in which they appear.

Default: None.

Returns

-------

optBin : list<str>

Returns the globally optimal binning split point.

Example: ['[1.0,3.0)','[3.0,6.0)','[6.0,10.0)','[10.0,10.0]','{-997}','{-999,-888}','{-1000,None}']

### OptBin

OptBin(X\_dats, y\_dats, y\_label={'unevent': 0, 'event': 1}, weight\_dats=None, train\_name=None, mono={}, default\_mono='N', sgst\_monos={}, distr\_min={}, default\_distr\_min=0.02, rate\_gain\_min={}, default\_rate\_gain\_min=0.001, bin\_cnt\_max={}, default\_bin\_cnt\_max=None, spec\_value={}, default\_spec\_value=[], spec\_distr\_min={}, default\_spec\_distr\_min=None, spec\_comb\_policy={}, default\_spec\_comb\_policy='N', order\_cate\_vars={}, unorder\_cate\_vars={}, no\_wild\_treat=None, default\_no\_wild\_treat=None, cust\_bins={})

\_x1OptBin calculates the global optimal binning of a single variable, and OptBin calculates the global optimal binning of multiple variables. OptBin is accomplished by calling \_x1OptBin.

Parameters

----------

X\_dats : DataFrame or dict<dat\_name,DataFrame>. Required

Multiple columns of variables or multiple identical variables belonging to different data sets

There are two points to note:

1. When mono['one var name'] = 'A', SCF automatically selects a constraint for 'one var name' from L+, L-, Uu, Un. When automatically selecting a constraint, it will use different data sets to verify whether the constraint selection is correct.

2. If the user has already calculated the monotonicity of the suggestion elsewhere, the result can be passed directly to the OptBin function through the parameter sgst\_monos. When mono['one var name']='A' is set, it will not be calculated repeatedly, wasting the user's waiting time.

According to the above two principles: if mono['one var name']='A' and sgst\_monos['one var name'] is None, then the type of X\_dats must be dict. In other cases, the type of X\_dats can be Series or dict

y\_dats : Series or dict<dat\_name,Series>. Required

The actual target. See X\_dats for when to pass in a Series or a dict.

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Example: {'unevent':'good','event':'bad'}

Generally, the things that users care about most are defined as events, which are easier to explain. For example, when you want to emphasize the occurrence of lung cancer, you can say that the incidence of lung cancer among smokers is 50% higher than that among non-smokers. In this case, you can write {'unevent':'No lung cancer','event':'Lung cancer'}. If you write {'unevent':'Lung cancer','event':'No lung cancer'}, although it does not affect the use of the model, the explanation will become that the incidence of lung cancer among smokers is 50% lower than that among non-smokers. Obviously, the first way of expression is easier to understand.

Default: {'unevent':0,'event':1}.

weight\_dats : Series or dict<dat\_name,Series> or None

The weight of the sample. Samples with weights can also solve the global optimal binning.

None: All weights are 1

When it is not None, see X\_dats for when to pass in Series or dict.

Default: None

train\_name : str or None

When mono['one var name']='A' and sgst\_mono['one var name']=None, this parameter specifies which dataset in X\_dats is used to calculate the monotonic trend suggestion for one var name. Except for the dataset corresponding to train\_name, all other datasets are used to verify the calculated monotonic trend.

Default: None

If X\_dats is a dict, train\_name cannot be None.

mono : dict

Configure monotonicity constraints for globally optimal binning for each variable.

Example: {"x1":"L","x2":"N"}

Value range:

N: IV value is the highest globally, no constraints

A: Automatically select a constraint from L+, L-, Uu, Un. Under this constraint, the IV value is the highest globally.

L: The IV value is globally the highest under linear monotone constraints (automatically determines L+, L-)

L+: The IV value is globally the highest under the linear monotonically increasing constraint

L-: The IV value is globally the highest under the linear monotonically decreasing constraint

U: The IV value is the highest globally under the U-type constraint (automatically determines Uu, Un)

default:{}.

default\_mono : str

Variables not listed in mono are monotonic by default.

Default: 'N'.

sgst\_monos:tuple

Suggested monotonic trend. If mono['one var name']='A' and sgst\_monos['one var name'] is not None, SCF automatically sets mono['one var name'] to the first element of sgst\_monos['one var name']. Avoid repeated operations

When using SCF automation, this parameter can be used to avoid repeated calculations through internal mechanisms. When using the Bins module alone, sgst\_monos can be ignored.

The first element of the tuple is the recommended monotonic trend: possible values are L+, L-, Uu, Un

The remaining elements are some built-in extended information

Default: None

distr\_min : dict

Configure the minimum distribution ratio for each variable

ex. {"x1":0.05,"x2":0.01}

Note: Setting the minimum distribution ratio of a bin is not necessary for finding the global optimal split point, but is determined by the user's confidence in the stability of the bin. If there are few sample points in a bin, the random fluctuation of its event rate may be relatively large, resulting in larger fluctuations in Woe and the final model.

default:{}.

default\_distr\_min : float

The default minimum distribution share for variables not appearing in distr\_min.

Default: 0.02.

rate\_gain\_min : dict

The event rate between any two adjacent bins cannot be less than rate\_gain\_min['one var name']

Some software packages often use information gain or chi-square value to suppress the formation of bins with too small differences.

However, these parameters do not provide a concrete and intuitive concept for users, and it is impossible to deduce how small the difference is.

Bins uses the event rate as an intuitive indicator to suppress the formation of bins with too small differences.

ex. {"x1":0.005,"x2":0.001}.

default:{}.

default\_rate\_gain\_min : float

For variables not present in rate\_gain\_min, the default is the minimum difference in event rates between any two adjacent bins.

Default: 0.001.

bin\_cnt\_max : dict

The maximum number of bins per variable. Bins for special values are not counted. If special values are merged into normal value bins due to merging rules, the merged bins are normal bins.

Example: {"x1":5,"x2":8}

Because Bins can specify parameters such as the monotonicity of variables, the minimum bin ratio, and the minimum difference in event rates, these parameters can automatically adjust the number of bins (usually the better the variable effect and the more evenly distributed it is, the more bins there are, and vice versa), so this parameter can usually be set to None.

If a variable is evenly distributed and has a strong order, the number of bins may be large. Although this is in line with the actual situation, users can also specify the maximum number of bins allowed for certain business considerations.

default:{}.

default\_bin\_cnt\_max : int

The default maximum number of bins for variables not appearing in bin\_cnt\_max.

Default: None.

spec\_value : dict

The range of special values for each variable

ex. {"x1":["{-9997}","{-9999,-9998}"],"x2":["{None}"]}

"{... , ...}" will not be parsed as a set, but will be processed as a string. {} in special values expresses a discrete value space symbol.

Let's take an example to explain the meaning of the expression:

"{-9997}": When the variable value is -9997, a special meaning occurs. For example, the number of court executions, -9997 may mean that the ID card is not in the citizen database, rather than being executed -9997 times. Through this example, users can clearly feel the difference in meaning between -9997 and the values 0, 1, and 2.

"{-9998,-9999}": When the variable value is -9998 or -9999, a special meaning occurs. Although the two meanings are different, for the business modeled this time, the two meanings can be regarded as the same meaning and processed according to the same business logic. For example, when collecting data, the data that was not collected due to Party A's reasons is marked as -9998, and the data that was not collected due to Party B's reasons is marked as -9999. However, for the business, these two values mean that the data is randomly missing, so they are both processed according to the logic of random missing. In this way, the value convention of the original data can be retained for backtracking, and the user can be exempted from the work of writing additional code when processing data.

"{None}" is a special value of null, which means: a special value with null value. The reason why {None} is used instead of {miss} is that the mechanism for generating null values is sometimes different from that for generating missing values. Missing values represent that the sample points are not collected due to some uncontrollable reasons during the sampling process, resulting in missing data information, such as network disconnection during data transmission and data not being collected due to equipment failure. This is a random missing. In addition to random missing, null values may also be caused by non-information missing, such as no loan record, no need for a certain examination due to health reasons, and the temperature is too low for the equipment to collect, etc. The null value itself is information. Do not mix null values with information and random missing null values into one special value.

ex. {"x1":"{None,-9997}"} means that after analyzing the business, null values and -9997 can be processed in the same way for this modeling.

If a variable is not configured with a null special value, but contains a null value, a {None} group is automatically generated to contain the null value of the variable.

default:{}.

default\_spec\_value : list

If the variable is not configured in spec\_value, its default special value

This configuration is usually convenient when the data has global public special values.

ex. ["{-9997}","{None}","{-9998,-9996}"].

default:[].

spec\_distr\_min : dict

If the distribution ratio of a special value bin is less than the value specified by spec\_distr\_min, the special value bin will be merged with other bins. For specific merging rules, see the spec\_comb\_policy setting.

If it is a nested dict, then the minimum percentage is specified separately for each special value of each variable.

If a dict, use the same minimum distribution for all unique values of each variable.

ex. {"x1":{"{-9997}":0.01,"{-9999,-9998}":0.05},"x2":0.01}.

default:{}.

default\_spec\_distr\_min : dict or float

If the special value of a variable is not configured in spec\_distr\_min, the default minimum distribution ratio of the special value

If it is a dict, a default minimum distribution ratio is specified for each special value.

If it is float, the default minimum distribution ratio of all special values is this value.

ex1. {"-9999":0.02,"-9998":0.01}

ex2. 0.05.

Default: None.

spec\_comb\_policy : dict

When the proportion of special values of a variable is less than the threshold specified by ${spec\_distr\_min}, the merge strategy to be adopted can be:

A:auto finds the closest eventProb among all values

a:auto only finds the closest eventProb among non-special values

F:first merges with the first bin of non-special values

L:last merges with the last bin of non-special values

M:median merges with the middle bin of non-special values (if there are an even number of bins, merge with the bin with the high event rate)

m:median merges with the middle bin of non-special values (if there are an even number of bins, merge with the bin with a low event rate)

B: max Probability is merged with the bin with the largest eventProb

S: min Probability merges with the bin with the smallest eventProb

N: Do not merge

If it is a nested dict, a merge strategy is specified for each special value of the variable. If there is a special value that is not covered, ${default\_spec\_comb\_policy} is used.

If a dict, all special values of the variable are merged using the strategy corresponding to that character.

ex. spec\_comb\_policy={"x1":{"{-9997}":"F","{-9998,None}":"L"},"x2":"N"}

If None, it is equivalent to all special values of all variables being "N" (except variables that can be overwritten by ${default\_spec\_comb\_policy})

Note: Let's use an example to explain the meaning of special value writing. The meaning of ["{-9997}","{-9998,-9999}"] is: there are three special values -9997, -9998, -9999 in the variable. According to the business meaning, they are divided into two business groups "{-9997}","{-9998,-9999}". -9997 itself becomes a business group, and -9998,-9999 form a business group. Because the two business groups "{-9997}","{-9998,-9999}" meet the special value merging rules set during binning, they are forcibly merged together at the data level to form a bin ["{-9997}","{-9998,-9999}"]. This kind of merging is different from merging -9998, -9999 into a business group. The merging mentioned in the business group is at the business level and is determined by understanding the business. The merging of ["{-9997}","{-9998,-9999}"] is at the data level and is determined only by calculating the event rate. The process and meaning behind how the three special values -9997, -9998, -9999 become two special value business groups "{-9997}","{-9998,-9999}" and finally become a special value merged bin ["{-9997}","{-9998,-9999}"] must be clear. This special value processing method is in line with statistical principles.

default:{}.

default\_spec\_comb\_policy : dict or str

When ${spec\_comb\_policy} does not contain a variable or a special value of a variable, the default special value merging strategy

If a dict, a default strategy is specified individually for each special value.

If it is str, all special values default to this strategy

ex1. {"-9999":"A","-9998":"B"}

ex2. M

For the value range, see ${spec\_comb\_policy}.

Default: 'N'.

order\_cate\_vars : dict

List the ordered categorical variables here and give the order of each category in the variable. If the value order is set to None, the lexicographic order of the characters is used as the order.

For ordered variables, adjacent nominal sequences can only appear in the same bin or at the beginning and end of adjacent bins.

If the nominal order is inconsistent with the event rate order, you can decide whether to configure the variable as an ordered variable based on the business situation.

Bins supports global optimal binning for ordered variables

For example: {"x1":("v1","v2"),"x2":("v3","\*\*","v4"),"x3":None}

All categories that do not appear in the configuration (excluding special values) are collectively called wildcard categories, represented by \*\*. When the lexicographic order is used as the order, there is no wildcard category.

None or {}: variables with no ordered categories in them.

default:{}.

unorder\_cate\_vars : dict

List unordered categorical variables here, where the categories are ordered based on the event rates.

Each variable is configured with a threshold, and the categories with a distribution ratio less than the threshold are merged into the wildcard category. If the threshold of a variable is None, the category with a frequency of too small for the variable will not be wildcarded.

ex1. {'x1':0.01,'x2':None}

Bins supports global optimal binning for unordered variables

In other datasets, it is possible to see values not covered in the training set, and these categories are also put into the wildcard category.

None or {}: There are no unordered categorical variables in the variable.

default:{}.

no\_wild\_treat : dict

When a category variable does not have a wildcard and uncovered categories appear, the processing methods that can be specified include:

L: Considered equal to the lowest category in the order

H: Considered equal to the highest order category

M: Considered equal to the middle category of the sequence (take the larger value when even)

m: Considered equal to the middle category of the sequence (smaller value if even)

ex. {'x1':'H','x2':'m'}

None: No processing is performed on uncovered categories that appear in the variable

Default: None.

default\_no\_wild\_treat : str

Default handling for uncovered categories when a variable has no wildcards and is not configured in no\_wild\_treat.

Default: None.

cust\_bins : dict

User-defined binning takes precedence over other binning settings.

ex. {"x1": ["[1.0,4.0)","[4.0,9.0)","[9.0,9.0]","{-997}","{-999,-888}"," {-1000,None}"]}

Returns

-------

optBins : dict<str,list<str>>

The globally optimal split point for each variable.

ex. {"x1": ["[1.0,4.0)","[4.0,9.0)","[9.0,9.0]","{-997}","{-999,-888}"," {-1000,None}"]}

### OptBin\_mp

OptBin\_mp(X\_dats, y\_dats, y\_label={'unevent': 0, 'event': 1}, weight\_dats=None, train\_name=None, mono={}, default\_mono='N', sgst\_mono={}, distr\_min={} , default\_distr\_min=0.02, rate\_gain\_min={}, default\_rate\_gain\_min=0.001, bin\_cnt\_max={}, default\_bin\_cnt\_max=None, spec\_value={}, default\_spec\_value=[], spec\_distr\_min={}, default\_spec\_distr\_min=None, spec\_comb\_policy={}, default\_spec\_comb\_policy=' N', order\_cate\_vars={}, unorder\_cate\_vars={}, no\_wild\_treat={}, default\_no\_wild\_treat=None, cust\_bins={}, cores=None)

Multi-process version of OptBin

Parameters

----------

cores : int

The number of CPU cores to use.

None: Use all cores

Default: None.

Other parameters: See [Bins. OptBin](#_OptBin)

Returns

-------

optBins : dict<str,list<str>>

Same return value as [Bins. OptBin](#_OptBin)

### \_x1FreqBin

\_x1FreqBin(x,weight=None,freqBin\_cnt=20,spec\_value=[]

,is\_cate=False,is\_order=False,no\_wild\_treat='M'

,order\_list=None

,y=None,y\_label={'unevent':0,'event':1},unorder\_combine\_thv=None)

The series is divided into equal frequency bins. With the support of the Cutter component, it produces more uniform segmentation points than the existing equal frequency binning software library.

Can handle special values. Special values set by users can be grouped separately

Supports multiple features, please refer to the introduction of [Bins](#_Bins) module

Parameters

----------

x : Series

A list of variables

weight : Series

The weight of the sample

None: All weights are 1

Default: None

freqBin\_cnt : int

Equal frequency binning groups

Default: 20

spec\_value : list

Special value

Example. ["{-9997}","{-9999,-9998}"]

default:[]

is\_cate : bool

Mark whether the variable is a categorical variable

True: Categorical variable

False: continuous variable

Default: False

is\_order : bool

Mark whether the variable is an ordered categorical variable

If is\_order=True, is\_cate will be automatically set to True

True: ordered categories

False: When is\_cate=True, it is an unordered category.

Default: False.

no\_wild\_treat : str

When a category variable does not have a wildcard and uncovered categories appear, the following processing methods are used:

L: Considered equal to the lowest category in the order

H: Considered equal to the highest order category

M: Considered equal to the middle category of the sequence (take the larger value when even)

m: Considered equal to the middle category of the sequence (smaller value if even)

None: No processing is performed on uncovered categories that appear in the variable

Default: 'M'

order\_list : tuple

Set the order of each value in the ordered categorical variable. The order of the tuple is the nominal order. If is\_order=True and order\_list=None, the lexicographic order of the characters is used as the order.

For ordered variables, adjacent nominal sequences can only appear in the same bin or at the beginning and end of adjacent bins.

If the nominal order is inconsistent with the event rate order, you can decide for yourself whether to configure the variable as an ordered variable or an unordered variable based on the business situation.

Supports globally optimal binning for ordered variables.

All categories that do not appear in the configuration (excluding special values) are collectively called wildcard categories, represented by \*\*. Values not covered in the training set may be seen in other datasets, and these categories are also put into wildcard categories.

When lexicographic order is used as the order, there is no wildcard category

Example: ("v1","\*\*","v2")

Default: None.

y : Series

The actual target. Because the order of unordered categorical variables is calculated by event rates, the target is needed to calculate equal frequency bins of unordered categorical variables.

Default: None.

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Example: {'unevent':'good','event':'bad'}.

Default: {'unevent':0,'event':1}.

unorder\_combine\_thv : float

Set a threshold for unordered categorical variables and merge categories with distribution proportions less than the threshold into wildcard categories. If is\_cate=True and unorder\_combine\_thv=None, categories with too small frequency of the variable will not be wildcarded.

In other datasets, it is possible to see values not covered in the training set, and these categories are also put into the wildcard category.

Default: None.

Returns

-------

freqbin : list<str>

Returns the equal frequency binning cut points.

Example: ['[1.0,3.0)','[3.0,6.0)','[6.0,10.0)','[10.0,10.0]','{-997}','{-999,-888} ','{-1000,None}']

### FreqBin

FreqBin(X, y=None, y\_label={'unevent': 0, 'event': 1}, weight=None, freqBin\_cnt=20, spec\_value={}, default\_spec\_value=[], order\_cate\_vars={}, unorder\_cate\_vars={ }, no\_wild\_treat=None, default\_no\_wild\_treat=None)

\_x1FreqBin calculates the equal frequency binning of one variable, and FreqBin calculates the equal frequency binning of multiple variables. FreqBin is accomplished by calling \_x1FreqBin.

Supports multiple features, please refer to the introduction of [Bins](#_Bins) module

Parameters

----------

X : DataFrame

Dataset, multiple columns of variables

y : Series

The actual target. Because the order of unordered categorical variables is calculated by event rates, the target is needed to calculate equal frequency bins of unordered categorical variables.

Default: None.

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Default: {'unevent':0,'event':1}.

weight : Series

The weight of the sample. Samples with weights can also solve the global optimal binning.

None: All weights are 1

Default: None.

freqBin\_cnt : int

Equal frequency binning groups

Default: 20.

spec\_value : dict

The value of each variable's special value

ex. {"x1":["{-9997}","{-9999,-9998}"],"x2":["{None}"]}.

default:{}.

default\_spec\_value : list

If the variable is not configured in spec\_value, its default special value

This configuration is usually convenient when the data has global public special values.

ex. ["{-9997}","{None}","{-9998,-9996}"].

default:[].

order\_cate\_vars : dict

List the ordered categorical variables here and give the order of each category in the variable. If the value order is set to None, the lexicographic order of the characters is used as the order.

For ordered variables, adjacent nominal sequences can only appear in the same bin or at the beginning and end of adjacent bins.

If the nominal order is inconsistent with the event rate order, you can decide whether to configure the variable as an ordered variable based on the business situation.

Example: {"x1":("v1","v2"),"x2":("v3","\*\*","v4"),"x3":None}

All categories that do not appear in the configuration (excluding special values) are collectively called wildcard categories, represented by \*\*. When the lexicographic order is used as the order, there is no wildcard category.

None or {}: variables with no ordered categories in them.

default:{}.

unorder\_cate\_vars : dict

List unordered categorical variables here, where the categories are ordered based on the event rates.

Each variable is configured with a threshold, and the categories with a distribution ratio less than the threshold are merged into the wildcard category. If the threshold of a variable is None, the category with a frequency of too small for this variable will not be wildcarded.

ex1. {'x1':0.01,'x2':None}

Bins supports global optimal binning for unordered variables

In other datasets, it is possible to see values not covered in the training set, and these categories are also put into the wildcard category.

None or {}: There are no unordered categorical variables in the variable.

default:{}.

no\_wild\_treat : dict

When a category variable does not have a wildcard and uncovered categories appear, the processing methods that can be specified include:

L: Considered equal to the lowest category in the order

H: Considered equal to the highest order category

M: Considered equal to the middle category of the sequence (take the larger value when even)

m: Considered equal to the middle category of the sequence (smaller value if even)

ex. {'x1':'H','x2':'m'}

None: No processing is performed on uncovered categories that appear in the variable

Default: None.

default\_no\_wild\_treat : str

Default handling for uncovered categories when a variable has no wildcards and is not configured in no\_wild\_treat.

Default: None.

Returns

-------

freqbins: dict<str,list<str>>

The equal-frequency cut-off points for each variable.

ex. {"x1": ["[1.0,4.0)","[4.0,9.0)","[9.0,9.0]","{-997}","{-999,-888}"," {-1000,None}"]}

### FreqBin\_mp

FreqBin\_mp(X, y=None, y\_label={'unevent': 0, 'event': 1}, weight=None, freqBin\_cnt=20, spec\_value={}, default\_spec\_value=[], order\_cate\_vars={}, unorder\_cate\_vars={ }, no\_wild\_treat=None, default\_no\_wild\_treat=None, cores=None)

Multi-process version of FreqBin

Parameters

----------

cores : int

The number of CPU cores to use.

None: Use all cores

Default: None.

Other parameters: see [Bins.FreqBin](#_FreqBin)

Returns

-------

freqbins: dict<str,list<str>>

The equal-frequency cut-off points for each variable.

ex. {"x1": ["[1.0,4.0)","[4.0,9.0)","[9.0,9.0]","{-997}","{-999,-888}","{-1000,None}"]}

### get\_bins\_stats

get\_bins\_stats(X, y=None, bins={}, weight=None, sync\_bins=True, y\_label={'unevent': 0, 'event': 1})

Given the split point of each variable, transform X according to the split point and then count the information of each interval.

Information includes: the number of samples in the interval, the sample proportion. If y is not None, the event occurrence rate, event non-occurrence rate, woe, IV in the interval will also be counted

Parameters

----------

X : DataFrame

Datasets containing multiple variables

y : Series

The actual target. Can be set to None

Default: None

bins : dict<str,list<str>>

The split point for each variable.

ex. {"x1": ["[1.0,4.0)","[4.0,9.0)","[9.0,9.0]","{-997}","{-999,-888}","{-1000,None}"]}

default:{}

weight : Series

The weight of the sample

None: All weights are 1

Default: None

sync\_bins : bool

True: When the extreme value of X exceeds the extreme value of the cut point, update bins

False: When the extreme value of X exceeds the extreme value of the cut point, bins are not updated

Default: True

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Default: {'unevent':0,'event':1}

Returns

-------

bins\_stats: dict<str,DataFrame>

Statistics for each variable.

Example: {'x1':DataFrame,'x2':DataFrame}

### \_x1trans\_woe\_value

\_x1trans\_woe\_value(x,bins\_stat)

Convert a column of values to woe

Support for categorical variables

When bins\_stat fails to cover the extreme value of x, update bins\_stat

Parameters

----------

x : Series

A list of variables

bins\_stat : DataFrame

The return value of get\_bins\_stats corresponds to the bins\_stat of this variable.

Returns

-------

Series

woe value

DataFrame

Updated bins\_stat. When bins\_stat fails to cover the extreme value of x, bins\_stat is updated

### trans\_woe\_value

trans\_woe\_value(X, bins\_stats, sync\_bins=True)

Returns the woe value of multiple variables, which [is done by calling Bins. \_x1trans\_woe\_value](#__x1trans_woe_value)

Support for categorical variables

When bins\_stat fails to cover the extreme value of the variable, bins\_stat can be updated

Parameters

----------

X : DataFrame

Dataset, containing multiple variables

bins\_stats: dict<str,DataFrame>

The return value of get\_bins\_stats.

sync\_bins : bool

Whether to update bins\_stats when bins\_stat fails to cover the extreme value of the variable

True: Update

False: Do not update

Default: True

Returns

-------

DataFrame

Converted WOE value

### trans\_woe\_value\_mp

trans\_woe\_value\_mp(X, bins\_stats, sync\_bins=True, cores=None)

[Bins. Multi-process version of trans\_woe\_value](#_trans_woe_value)

Parameters

----------

cores : int

The number of CPU cores to use.

None: Use all cores

Default: None.

Other parameters: see [Bins.trans\_woe\_value](#_trans_woe_value)

Returns

-------

DataFrame

Converted WOE value

### is\_cate\_bins

is\_cate\_bins(bins)

Determine whether a bin is a categorical bin

Parameters

----------

bins : list

The bins to be judged.

Returns

-------

bool

True: Category bins

False: Continuous bins

### bins\_stats\_to\_IV

bins\_stats\_to\_IV(bins\_stats, asc=False)

By passing bins\_stats, the IV of each variable is returned.

Parameters

----------

bins\_stats: dict<str,DataFrame>

The return value of get\_bins\_stats.

asc : bool

True: Returns the IV in positive order

False: Returns the IV in reverse order

Default: False.

Returns

-------

Series

IV value for each variable

### \_x1MonoSuggest

\_x1MonoSuggest(x\_dats,y\_dats,w\_dats=None,train\_name='train',spec\_value=[],is\_cate=False,is\_order=False,no\_wild\_treat='M'

,order\_list=None,y\_label={'unevent':0,'event':1})

To give a monotonicity suggestion for a variable, the monotonicity is calculated on the specified dataset and then verified on other datasets.

Suggested values for monotonicity are:

L+: linear monotonically increasing

L-: linear monotonically decreasing

Uu: U-shaped concave

Un: U-shaped convex

Parameters

----------

x\_dats : dict<dat\_name,Series>

The same variable in different datasets

y\_dats : dict<dat\_name,Series>

Actual target

Targets in different datasets

w\_dats : dict<dat\_name,Series>

Sample weights in different datasets.

None: All weights are 1

Default: None

train\_name : str

Which dataset is used for calculating the monotonicity suggestion? The remaining datasets are used to test the calculated trend.

Default: train

spec\_value : list

Special value range

Example. ["{-9997}","{-9999,-9998}"]

If a variable is not configured with an empty special value, but contains an empty value, a {None} group is automatically generated to contain the empty value of the variable.

When calculating monotonicity, special values are removed

default:[]

is\_cate : bool

Indicates whether the variable is categorical.

True: Categorical variable

False: continuous variable

Default: False.

is\_order : bool

Indicates whether the variable is an ordered categorical variable.

If is\_order=True, is\_cate will be automatically set to True

True: ordered categories

False: When is\_cate=True, it is an unordered category. An unordered category will not give a suggested monotonic trend, and the first element of the return value will be marked as an unordered category.

Default: False.

no\_wild\_treat : str

When a category variable does not have a wildcard and uncovered categories appear, the following processing methods are used:

L: Considered equal to the lowest category in the order

H: Considered equal to the highest order category

M: Considered equal to the middle category of the sequence (take the larger value when even)

m: Considered equal to the middle category of the sequence (smaller value if even)

None: No processing is performed on uncovered categories that appear in the variable

Default: 'M'

When is\_cate=False, this parameter will be ignored.

order\_list : tuple

Set the order of each value in the ordered categorical variable. The order of the tuple is the nominal order. If is\_order=True and order\_list=None, the lexicographic order of the characters is used as the order.

If the nominal order is inconsistent with the event rate order, you can decide for yourself whether to configure the variable as an ordered variable or an unordered variable based on the business situation.

All categories that do not appear in the configuration (excluding special values) are collectively called wildcard categories, represented by \*\*. Values not covered in the training set may be seen in other datasets, and these categories are also put into wildcard categories.

When lexicographic order is used as the order, there is no wildcard category

Example: ("v1","\*\*","v2")

Default: None.

This parameter will be ignored when is\_cate=False or is\_order=False

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Default: {'unevent':0,'event':1}

Returns

-------

tuple(monotonicity suggestion, \*\*)

The first element is a monotonicity suggestion, followed by some additional information

Unordered categories will not suggest a monotonic trend, and the first element of the return value will be marked as unordered.

### MonoSuggest

MonoSuggest(X\_dats, y\_dats, w\_dats=None, train\_name='train', spec\_value={}, default\_spec\_value=[], order\_cate\_vars={}, unorder\_cate\_vars={}, no\_wild\_treat=None, default\_no\_wild\_treat=None, y\_label={'unevent': 0, 'event': 1})

Give monotonicity suggestions for multiple variables

This is done by [calling](#__x1MonoSuggest) Bins.\_x1MonoSuggest

Parameters

----------

X\_dats : dict<dat\_name,DataFrame>

Multiple variables that are the same in different datasets

y\_dats : dict<dat\_name,Series>

Actual target

Targets in different datasets

w\_dats : dict<dat\_name,Series>

Sample weights in different datasets.

None: All weights are 1

Default: None

train\_name : str

Which dataset is used for calculating the monotonicity suggestion? The remaining datasets are used to test the calculated trend.

Default: train

spec\_value : dict

The range of special values for each variable

ex. {"x1":["{-9997}","{-9999,-9998}"],"x2":["{None}"]}

default:{}.

default\_spec\_value : list

If the variable is not configured in spec\_value, its default special value

This configuration is usually convenient when the data has global public special values.

ex. ["{-9997}","{None}","{-9998,-9996}"].

default:[].

order\_cate\_vars : dict

List the ordered categorical variables here and give the order of each category in the variable. If the value order is set to None, the lexicographic order of the characters is used as the order.

If the nominal order is inconsistent with the event rate order, you can decide whether to configure the variable as an ordered variable based on the business situation.

For example: {"x1":("v1","v2"),"x2":("v3","\*\*","v4"),"x3":None}

All categories that do not appear in the configuration (excluding special values) are collectively called wildcard categories, represented by \*\*. When the lexicographic order is used as the order, there is no wildcard category.

None or {}: variables with no ordered categories in them.

default:{}.

unorder\_cate\_vars : list or dict

Listing unordered categorical variables here will not give a suggested monotonic trend, and the first element of the return value will be marked as an unordered category.

ex1. ['x1','x2']

ex2. {'x1':0.01,'x2':None}

None or {}: There are no unordered categorical variables in the variable.

default:{}.

no\_wild\_treat : dict

When a category variable does not have a wildcard and uncovered categories appear, the processing methods that can be specified include:

L: Considered equal to the lowest category in the order

H: Considered equal to the highest order category

M: Considered equal to the middle category of the sequence (take the larger value when even)

m: Considered equal to the middle category of the sequence (smaller value if even)

ex. {'x1':'H','x2':'m'}

None: No processing is performed on uncovered categories that appear in the variable

Default: None.

default\_no\_wild\_treat : str

Default handling for uncovered categories when a variable has no wildcards and is not configured in no\_wild\_treat.

Default: None.

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Default: {'unevent':0,'event':1}

Returns

-------

dict<var\_name,tuple(monotonicity suggestion,\*\*)>

The first element is a monotonicity suggestion, followed by some additional information (for backward compatibility)

### MonoSuggest\_mp

MonoSuggest\_mp(X\_dats, y\_dats, w\_dats=None, train\_name='train'

, spec\_value={}, default\_spec\_value=[]

, order\_cate\_vars={}, unorder\_cate\_vars={}, no\_wild\_treat=None, default\_no\_wild\_treat=None

, y\_label={'unevent': 0, 'event': 1}, cores=None)

Multi-process version of MonoSuggest

Parameters

----------

cores : int

The number of CPU cores to use.

None: Use all cores

Default: None.

Other parameters: See [Bins. MonoSuggest](#_MonoSuggest)

Returns

-------

dict<var\_name,tuple(monotonicity suggestion,\*\*)>

The first element is a monotonicity suggestion, followed by some additional information (for backward compatibility)

# Cutter

Perform equal frequency segmentation or segmentation according to specified split points, which has the following enhancements over the built-in segmenter in Python:

1. Mathematically provable analytical solution with minimum global error.

2. All split points come from the original values.

3. More humane support for left closed and right open: a. The last group is right closed. b. The extreme values at both ends of the minimum and maximum groups are from the original data, which is different from the built-in splitter in Python, which will change the extreme values at both ends.

4. The globally optimal segmentation solution can also be given for extremely tilted data.

5.Support weighted series.

6. Supports special values specified by users. Special values are grouped separately, and users can also configure multiple special values to be combined into one group.

7. Special values do not include null values, but if there are null values in the sequence, the null values will be automatically processed into a group.

8. Use the specified split point to cut the sequence. When the maximum or minimum value of the sequence exceeds the split point boundary, the maximum and minimum values of the split point will be automatically extended.

It is recommended to try to replace Python's built-in equal frequency segmentation component with Cutter.

Note: Cutter can only be used for numeric sequences. Character sequences must first be converted to numeric using Category.

## Functions

### value\_counts\_weight

value\_counts\_weight(data, weight=None)

Same functionality as pandas.Series.value\_counts(), but value\_counts\_weight supports weights

Parameters

----------

data : array like

The sequence to be counted.

weight : array like

The weight of the data point

None: Each data point has a weight of 1

Default: None.

Returns

-------

Series

The number of each data point (with weight)

Series

The proportion of each data point (with weight)

### value\_counts\_weight\_y

value\_counts\_weight\_y(dat, y, y\_label={'event': 1, 'unevent': 0}, weight=None)

Count the occurrence and non-occurrence of events for each value

Parameters

----------

dat : array like

A series of numbers

y : array like

The actual target.

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Default: {'unevent':0,'event':1}.

weight : Series

The weight of the data point

None: All weights are 1

Default: None.

Returns

-------

DataFrame

columns = ['Number of each value', 'Proportion', 'Number of events that occurred', 'Number of events that did not occur', 'Event incidence rate']

### \_spec\_del

\_spec\_del(data,spec\_value)

Remove special values from a sequence

Parameters

----------

data : array like

A column of numbers.

spec\_value : list

Definitions of special values.

Example. ["{-9997}","{-9999,-9998}"]

Returns

-------

array like is the same type as data

The new array after removing special values.

### is\_spec\_value

is\_spec\_value(value, spec\_value)

Determine whether the value is a special value

Parameters

----------

value : float or str

The value to be judged

spec\_value : list

Special values. Example: ['{-999,-888}','{-1000}']

Returns

-------

bool

Whether the value is a special value.

### \_is\_ascending

\_is\_ascending(bins)

Determine whether the bins are in ascending or descending order

Parameters

----------

bins : list

Split Point

ex1. ["[1.0,4.0)","[4.0,9.0)","[9.0,9.0]","{-997}","{-999,-888}","{-1000,None}"]

Returns

-------

bool

True: for ascending bins

False: Descending bins

### \_ext\_bins

\_ext\_bins(data,ser\_bin)

Expand the extreme values of bins

Parameters

----------

data : array like

A column of numbers, you need to remove special values first

ser\_bin : Series

bins

Returns

-------

bins

Expanded bins.

bool

Whether bins are expanded

True: ser\_bin is expanded

False: ser\_bin does not need to be expanded

### \_is\_spec\_bin

\_is\_spec\_bin(one\_bin)

Determine whether a bin is a special value

Parameters

----------

one\_bin : list or str

A single bin.

Supports combine bin. Combine bin means that two bins meet the manually set merging rules and are merged together.

['[1,2)',['[2,4]','{-1000}']], where ['[2,4]','{-1000}'] is a merged bin

Returns

-------

bool

True: This bin is a special value bin

False: This bin is not a special value bin

### freq\_cut

freq\_cut(data, threshold\_distr, min\_distr, weight=None, spec\_value=[], ascending=True)

Equal frequency segmentation tool. Supports weighted and special valued sequences. For more functions, please refer to the introduction of [Cutter](#_Cutter) module.

Parameters

----------

data : array like

A sequence of numbers to be split

threshold\_distr : int or float

Greater than 1: Divide into several parts

Less than 1: What is the proportion of each portion?

min\_distr : float

The minimum acceptable proportion. Due to data skew, it is not guaranteed that all bins can meet threshold\_distr. Some bins may be higher than threshold\_distr, and some may be lower than threshold\_distr. However, the lowest will not be lower than min\_distr.

weight : array like

The weight of the data point

None: Each data point has a weight of 1

Default: None.

spec\_value : list

Special value range

Example. ["{-9997}","{-9999,-9998}"]

default:[].

ascending : bool

True: binning in ascending order. Example ['[1,4)','[4,10)','[10,10]']

False: Binning in descending order. Example ['[10,9)','[9,4)','[4,1]']

Default: True.

Returns

-------

list

Return to bins

Example: ['[1.0,4.0)','[4.0,6.0)','[6.0,9.0)','[9.0,10.0]','{-997}','{-1000,None}' ]

### cut\_by\_bins

cut\_by\_bins(data, bins)

Split the data into the specified bins and return the labels for each data point in order.

Parameters

----------

data : array like

A sequence of numbers to be split

bins : list

Specified bins

Example: ['[1.0,4.0)','[4.0,6.0)','[6.0,9.0)','[9.0,10.0]','{-997}','{-1000,None}' ]

It also supports merging bins, such as ['[1.0,4.0)','[4.0,6.0)','[6.0,9.0)',['[9.0,10.0]','{-997}'],'{ -1000,None}']

Returns

-------

array like

Same data type as data

The bin corresponding to the data point in data

bool

Whether the extreme values of bins are updated

list

Updated extreme value bins

### freq\_cut\_data

freq\_cut\_data(data, threshold\_distr, min\_distr, weight=None, spec\_value=[], ascending=True)

First call freq\_cut, then call cut\_by\_bins with the return value

Parameters

----------

data : array like

The number sequence to be split

threshold\_distr : int or float

Greater than 1: Divide into several parts

Less than 1: What is the proportion of each portion?

min\_distr : float

The minimum proportion you can accept. Because of data skew, it is not guaranteed that all bins can meet threshold\_distr. Some bins will be higher than threshold\_distr, and some will be lower than threshold\_distr. However, the lowest will not be lower than min\_distr.

weight : array like

The weight of the data point

None: Each data point has a weight of 1

Default: None.

spec\_value : list

Special value range

Example. ["{-9997}","{-9999,-9998}"]

default:[].

ascending : bool

True: binning in ascending order. Example ['[1,4)','[4,10)','[10,10]']

False: Binning in descending order. Example ['[10,9)','[9,4)','[4,1]']

Default: True.

Returns

-------

array like

Same data type as data

The bin corresponding to the data point in data

list

Split Point

### cut\_array

cut\_array(datas, threshold\_distr, min\_distr, cutby=0, weight=None, spec\_value=[], ascending=True)

Cut multiple sets of sequences in a unified way

It will automatically expand the extreme value of the corresponding data of cutby

Parameters

----------

datas: dict,DataFrame,ndarray like

Multiple sets of sequences to be split

threshold\_distr : int or float

Greater than 1: Divide into several parts

Less than 1: What is the proportion of each portion?

min\_distr : float

The minimum proportion you can accept. Because of data skew, it is not guaranteed that all bins can meet threshold\_distr. Some bins will be higher than threshold\_distr, and some will be lower than threshold\_distr. However, the lowest will not be lower than min\_distr.

cutby : int ,str, list

int: If datas is ndarray like, then cutby is based on the number of columns to group.

str: If datas is dict or DataFrame, group them according to the corresponding benchmark series of cutby

list:cutby is bins, all series are grouped according to cutby

Default: 0.

weight : array like

The weight of each data point in the base series only needs to pass the weight of the base series.

None: The weight of each data point in the base series is 1

Default: None.

spec\_value : list

Special value range

Example. ["{-9997}","{-9999,-9998}"]

default:[].

ascending : bool

True: binning in ascending order. Example ['[1,4)','[4,10)','[10,10]']

False: Binning in descending order. Example ['[10,9)','[9,4)','[4,1]']

Default: True.

Returns

-------

dict,DataFrame,ndarray like

The same type as the array in datas

The bins corresponding to the data points in the sequence in datas

bins : list

Split Point

### sort\_label

sort\_label(one\_bin)

The lambda function used to sort the bins.

Usage: sorted(bins,key=sort\_label)

Parameters

----------

one\_bin : str or list

str: a bin

list: a combination bin

Returns

-------

float or str

The order of the bins.

### eq\_bin

eq\_bin(one\_bin, compare)

Are two bins equal?

Parameters

----------

one\_bin : str or list

Original bin

compare : str or list

Comparison bin

Returns

-------

bool

Whether two bins are equal.

# Category

The Category module is used to convert categorical variables into continuous variables.

1. Can handle ordered and unordered categories

2. Ordered categories can specify the category order or convert it using dictionary order

3. Unordered categories are transformed using event rates

4.Support sequences with special values

5. Support weighted series

6. You can set wildcards to handle categories that do not appear in the training set

7.Support merging and converting small categories

## Functions

### cate\_to\_cateBin

cate\_to\_cateBin(x, cate\_bins, wild='\*\*')

Convert categories into category bins.

Example: 'A' -> '<A,B,C>'

Parameters

----------

x : str

category.

cate\_bins : list

Category bins.

ex. ['<A,B,C>','<a,b,c,\*\*>']

wild : str

Wildcard identifier

default:'\*\*'.

Returns

-------

str

The category bin corresponding to x.

ex. '<a,b,c,\*\*>'

### cate\_to\_num

cate\_to\_num(dat, is\_order=False, spec\_value=[], wild='\*\*', no\_wild\_treat='M', order\_list=None, letter\_asc=True, y=None, y\_label={'event': 1, 'unevent': 0}, weight=None, unorder\_combine\_thv=None, prob\_asc=True)

Convert categorical variables to numeric values

Parameters

----------

dat : array like

A list of categorical variables to be converted

is\_order : bool

True: The categorical variable to be converted is an ordered category

False: The categorical variable to be converted is an unordered category

Default: False

spec\_value : list

List all special values. Example: ['{-999,-888}','{-1000}']

spec\_value will not be converted to a number. In the new array, this value is still treated as a special value and is handled according to the rules for special values.

default:[]

wild : str

Wildcard identifier

default:'\*\*'

no\_wild\_treat : str

When a category variable does not have a wildcard and uncovered categories appear, the processing methods that can be specified include:

L: Considered equal to the lowest category in the order

H: Considered equal to the highest order category

M: Considered equal to the middle category of the sequence (take the larger value when even)

m: Considered equal to the middle category of the sequence (smaller value if even)

Default: M

order\_list : tuple

Set the order of each value in the ordered categorical variable. The order of the tuple is the nominal order. If is\_order=True and order\_list=None, the lexicographic order of the characters is used as the order.

If the nominal order is inconsistent with the event rate order, you can decide for yourself whether to configure the variable as an ordered variable or an unordered variable based on the business situation.

Supports global optimal binning for ordered variables. The converted sequence can be directly processed by Bins.

All categories that do not appear in the configuration (excluding special values) are collectively called wildcard categories, represented by \*\*. Values not covered in the training set may be seen in other datasets, and these categories are also put into wildcard categories.

When lexicographic order is used as the order, there is no wildcard category

Example: ("v1","\*\*","v2")

Default: None.

letter\_asc : bool

When ordered categories use lexicographic order:

True: The larger the lexicographic order, the larger the converted value

False: The smaller the lexicographic order, the smaller the converted value

Default: True.

y : Series

The actual target. Because the order of unordered categorical variables is calculated by event rates, target is needed when is\_order=False.

Default: None.

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Example: {'unevent':'good','event':'bad'}.

Default: {'unevent':0,'event':1}.

weight : array like

The weight of the sample. When is\_order=False,

None: All weights are 1

Default: None.

unorder\_combine\_thv : float

Set a threshold for unordered categorical variables and merge categories with distribution proportions less than the threshold into wildcard categories. If is\_order=False and unorder\_combine\_thv=None, categories with too small frequency of the variable will not be wildcarded.

Default: None.

prob\_asc : bool

When the categories are unordered:

True: The higher the event rate, the larger the converted value

False: The higher the event rate, the smaller the converted value

Default: True.

Returns

-------

trans\_data : array like

The converted numeric sequence

encoder : Series

Converted code table

The key of the encoder is the category, and the value is the corresponding value.

### numInterval\_to\_cateBin

numInterval\_to\_cateBin(encoder, numer\_interval)

Convert the numeric interval back to a category set. The expression of cateBin is '<c1,c2,...>'

Parameters

----------

encoder : Series

The code table returned by cate\_to\_num. The encoder key is the category and the value is the corresponding value.

numer\_interval : str or list

A numeric range or a combined numeric range to be converted

Example: '[4,7)' or ['[4,7)','{-999}']

Returns

-------

str

cateBin. If numer\_interval contains the number corresponding to the wildcard, the converted cateBin contains the wildcard

Example: When numer\_interval is a numeric interval, '<A,B,C,\*\*>'

When numer\_interval is a combined numeric interval, ['<A,B,C>','<a,b,c,\*\*>']

# Reg\_Step\_Wise\_MP

It is a linear two-way stepwise regression and logistic two-way stepwise regression implemented in Python, which adds the following features compared to traditional two-way stepwise regression:

1. When performing stepwise variable selection for logistic regression, AUC, KS, and LIFT indicators can be used instead of AIC and BIC indicators. For some businesses, AUC and KS are indicators that are more suitable for business scenarios. For example, in the sorting business, the model built using the KS indicator has the advantage of using fewer variables but the KS of the model does not decrease on multiple test sets according to past experience.

2. When performing stepwise variable selection, use other data sets to calculate model evaluation indicators instead of using the modeling data set. Especially when the amount of data is large and there is a validation set in addition to the training set and test set, it is recommended to use the validation set to calculate evaluation indicators to guide variable selection. This helps reduce overfitting.

3. Support the use of partial data to calculate model evaluation indicators to guide variable selection. Scenario example: If the business needs to maintain a certain pass rate of N%, then the bad event rate of the first N% samples can be minimized, and all samples do not need to participate in the calculation. According to past experience: In appropriate scenarios, using partial data as evaluation indicators to select fewer variables than using all data, but the indicator that users are concerned about does not decrease in multiple test sets. Because the model only focuses on sample points that are easier to distinguish at the head, business goals can be achieved without too many variables.

4. Supports setting multiple conditions. Variables must meet all conditions at the same time before they can be included in the model. Variable selection and model diagnosis are performed simultaneously to avoid repeated modeling due to model diagnosis failure. The built-in conditions are: P-Value, VIF, correlation coefficient, coefficient sign.

5. Supports specifying variables that must be entered into the model. If the specified variables to be entered into the model conflict with the conditions in 4, a complete mechanism is designed to solve the problem.

6. The modeling process is output to EXCEL, recording the reasons for deleting each variable and the process information of each round of stepwise regression.

7. When the number of configured CPU cores is greater than 1, multi-process parallel computing stepwise regression is automatically started.

## Classes

### LinearReg

#### \_\_init\_\_

\_\_init\_\_(self, X, y, user\_save\_cols=[], user\_set\_cols=[], fit\_weight=None

, measure='r2', measure\_weight=None, measure\_X=None, measure\_y=None, kw\_measure\_args=None

, pvalue\_max=0.05, vif\_max=3, corr\_max=0.8, coef\_sign={}, default\_coef\_sign=None

, iter\_num=20, kw\_algorithm\_class\_args=None, n\_core=None, results\_save=None)

Bidirectional stepwise linear regression

Parameters

----------

X : DataFrame

X dataset

y : Series

Actual target

user\_save\_cols : array like

Forced variables into the module

default:[].

user\_set\_cols : array like

Only these variables can be entered into the model, no additions or deletions.

If user\_set\_cols is not empty and has length greater than 0, stepwise regression degenerates to ordinary regression.

default:[].

fit\_weight : array like

Modeling weights

None: The weight of each modeling sample point is 1

This weight is not the sample weight, and the two have different meanings. Generally speaking, the sample weight mainly records the sampling ratio during the sampling process, while the setting of the modeling weight is based on various considerations, such as reducing heteroscedasticity, balancing positive and negative samples, and setting different loss costs.

Default: None.

measure : str

In bidirectional stepwise regression, an indicator to determine whether the model has improved.

Classification indicators are: r2, adj\_r2 (under development)

Default: 'r2'.

measure\_weight : array like

The sample weight used when calculating the measure indicator. This weight has a different meaning from fit\_weight, and the meaning of this weight is usually similar to that of the sample weight.

Default: None.

measure\_X : DataFrame

X data set used to measure model performance indicators when gradually selecting variables

None: Use the same dataset as used for modeling

Default: None.

measure\_y : Series

The y data set used to measure the model performance index when stepping through variables

None: Use the same dataset as used for modeling

Default: None.

kw\_measure\_args : dict

Additional parameters passed to the metric calculation method

Default: None.

pvalue\_max : float

The p-value of the coefficients of all model variables (excluding the intercept term) must be less than or equal to the threshold

Variables that the user requires to be entered into the model are not subject to this restriction

If a non-mandatory variable is included in the model and causes the p-value of a mandatory variable whose p-value is originally less than the threshold to exceed the threshold, the non-mandatory variable will not be included in the model. However, if the p-value of a mandatory variable originally exceeds the threshold, that is, the p-value caused by other mandatory variables exceeds the threshold, it will not affect the introduction of the non-mandatory variable.

None: No constraints are imposed on the p-value of the model variable

Default: 0.05

vif\_max : float

The vif of all model variables (excluding the intercept term) must be less than or equal to the threshold

Variables forced into the module are not affected by this constraint.

If a non-mandatory variable is introduced into the model and causes the vif of a mandatory variable whose vif is originally less than the threshold to exceed the threshold, the non-mandatory variable will not be introduced into the model. However, if the vif of a mandatory variable itself exceeds the threshold, that is, the vif caused by other mandatory variables exceeds the threshold, it will not affect the introduction of the non-mandatory variable.

None: No restrictions are imposed on the vif of the input model variable

Default: 3

corr\_max : float

The correlation coefficients between all input variables must be less than or equal to the threshold

If the correlation coefficient of a non-mandatory variable with a mandatory variable exceeds the threshold, the non-mandatory variable will not be introduced into the model.

Even if the correlation coefficient between two mandatory variables is above this threshold, both variables will be included.

None: No restriction on the correlation coefficient of the model variables

Default: 0.8

coef\_sign : dict

Sign constraints on variable coefficients

ex. {"x1":"+","x2":"-"} or file://xx/xx.json read from the file

Value Description:

+ The coefficient of this variable is positive

- The coefficient of this variable is negative

None This variable does not constrain the coefficient sign

coef\_sign = None: No constraints are placed on the coefficient signs of all variables

Variables that the user forces to be included in the model are not subject to this constraint

If the introduction of a non-mandatory variable causes a mandatory variable that originally satisfied the symbol constraint to no longer satisfy the symbol constraint, the non-mandatory variable cannot be included in the model. If the symbol of the mandatory variable itself does not satisfy the constraint, the introduction of the non-mandatory variable will not be affected.

Default: None

default\_coef\_sign : str

When a variable is not in coef\_sign, the default value of the variable sign constraint is

None: The default value of all variables is None

Default: None

iter\_num : int

Maximum number of iterations

Each iteration has two operations:

1. Find a variable from all the remaining variables that meets the constraints and whose addition will improve the model index by the highest amount compared to the current one. Introduce this variable into the model

2. Find a variable from all the variables in the model that meets the constraints and whose removal will improve the model index by the highest amount. Remove this variable from the model.

If in round N (N < iter\_num), adding or removing variables cannot further improve the performance of the model, the iteration is terminated early.

Default: 20

kw\_algorithm\_class\_args : dict

Additional parameters passed to the underlying regression algorithm.

Default: None.

n\_core : int or float

>1: specifies the number of CPU cores

=1: Do not use multi-process

<1: Actual number of cores = total number of CPU cores \* n\_core rounded down

None: Actual number of cores = total number of CPU cores - 1

Default: None.

results\_save : str

The file name that records the modeling process. In addition to common information, the file also records the process of variable selection and elimination in stepwise regression, as well as the reasons for variable deletion.

None: Do not record the process

Default: None.

#### fit

fit(self)

After constructing the LinearReg object, you need to call the fit method to perform linear bidirectional stepwise regression.

Returns

----------------------------------------------------------

in\_vars : list

Model variables

clf\_final : statsmodels.regression.linear\_model.RegressionResults like

The main method of clf\_final:

predict(X) outputs the model prediction value

The main attributes of clf\_final are:

intercept\_ intercept term

coef\_ Coefficient of each variable (excluding the intercept term)

tvalues The t statistic of each model variable. Where const is the t statistic of the intercept term

pvalues Two-tailed P-value of each model variable

rsquared R2 goodness of fit of the model

rsquared\_adj The adjusted R2 goodness of fit of the model

aic aic

bic bic

resid residual of the model

clf\_perf : DataFrame

Model building information: R-squared, adjusted R-squared, AIC, BIC, Log-Likelihood, F-statistic, Prob (F-statistic), etc.

clf\_coef : DataFrame

Model parameter information: Coef, Std.Err, coefficient test t statistic, t statistic Pvalue, confidence interval, Standardized Coefficients

del\_reason : Series

The reason for deletion of each deleted variable

step\_proc : DataFrame

Detailed records of each round of modeling, including: adding or removing variables, model performance indicators

### LogisticReg

#### \_\_init\_\_

\_\_init\_\_(self, X, y, y\_label={'unevent': 0, 'event': 1}, user\_save\_cols=[], user\_set\_cols=[]

, fit\_weight=None, measure='aic', measure\_weight=None

, measure\_frac=None, measure\_X=None, measure\_y=None, kw\_measure\_args=None

, pvalue\_max=0.05, vif\_max=3, corr\_max=0.8, coef\_sign={}, default\_coef\_sign=None

, iter\_num=20, kw\_algorithm\_class\_args=None, n\_core=None, results\_save=None)

Bidirectional stepwise logistic regression. It adds more functions than the existing bidirectional stepwise logistic regression package. Please refer to the introduction of the [Reg\_Step\_Wise\_MP](#_Reg_Step_Wise_MP) module.

Parameters

----------

X : DataFrame

X dataset

y : Series

Actual target

y\_label : dict

Define which value in y means the event has occurred and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Example: {'unevent':'good','event':'bad'}

Generally, the things that users care about most are defined as events, which are easier to explain. For example, when you want to emphasize the occurrence of lung cancer, you can say that the incidence of lung cancer among smokers is 50% higher than that among non-smokers. In this case, you can write {'unevent':'No lung cancer','event':'Lung cancer'}. If you write {'unevent':'Lung cancer','event':'No lung cancer'}, although it does not affect the use of the model, the explanation will become that the incidence of lung cancer among smokers is 50% lower than that among non-smokers. Obviously, the first way of expression is easier to understand.

Default: {'unevent':0,'event':1}.

user\_save\_cols : array like

Forced variables into the module

default:[]

user\_set\_cols : array like

Only these variables can be entered into the model, no additions or deletions.

If user\_set\_cols is not empty and has length greater than 0, stepwise regression degenerates to ordinary regression.

default:[]

fit\_weight : array like

Modeling weights

None: The weight of each modeling sample point is 1

This weight is not the sample weight, and the two have different meanings. Generally speaking, the sample weight mainly records the sampling ratio during the sampling process, while the setting of the modeling weight is based on various considerations, such as reducing heteroscedasticity, balancing positive and negative samples, and setting different loss costs.

Default: None

measure : str

In bidirectional stepwise regression, an indicator to determine whether the model has improved.

Indicators include: aic, bic, roc\_auc, ks, lift\_n (under development), ks\_price (under development)

Cannot be None

Default: 'aic'.

measure\_weight : array like

The sample weight used when calculating the measure indicator. This weight has a different meaning from fit\_weight, and the meaning of this weight is usually similar to that of the sample weight.

If measure is aic or bic, measure\_weight configuration is ignored.

Default: None.

measure\_frac : float

Sort the events by probability of occurrence from large to small or from small to large, and take the first N sample points from the configuration file ${MODEL CONFIG:measure\_data\_name} as the evaluation index of the model

None: Take all sample points from the configuration file ${MODEL CONFIG:measure\_data\_name} to calculate the model's evaluation index. Equivalent to measure\_frac=1.

If measure\_index is aic or bic, measure\_frac configuration is ignored. Only all modeling data can be used.

measure\_frac > 1: Take the first N = measure\_frac sample points from large to small

0 <measure\_frac <= 1: Take the first N = sample\_n\*measure\_frac sample points from large to small (round down)

-1 <= measure\_frac < 0: Take the first N = sample\_n\*measure\_frac\*-1 sample points from small to large (round down)

measure\_frac < -1: Take the first N = measure\_frac\*-1 sample points from small to large

Default: None

measure\_X : DataFrame

X data set used to measure model performance indicators when gradually selecting variables

None: Use the same dataset as used for modeling

If measure is aic or bic, measure\_X configuration is ignored. You can only use the same dataset as the one used for modeling.

Default: None.

measure\_y : Series

The y data set used to measure the model performance index when stepping through variables

None: Use the same dataset as used for modeling

If measure is aic or bic, measure\_y configuration is ignored. You can only use the same dataset as when modeling.

Default: None.

kw\_measure\_args : dict

Additional parameters passed to the metric calculation method

Default: None.

pvalue\_max : float

The p-value of the coefficients of all model variables (excluding the intercept term) must be less than or equal to the threshold

Variables that the user requires to be entered into the model are not subject to this restriction

If a non-mandatory variable is included in the model and causes the p-value of a mandatory variable whose p-value is originally less than the threshold to exceed the threshold, the non-mandatory variable will not be included in the model. However, if the p-value of a mandatory variable originally exceeds the threshold, that is, the p-value caused by other mandatory variables exceeds the threshold, it will not affect the introduction of the non-mandatory variable.

None: No constraints are imposed on the p-value of the model variable

Default: 0.05

vif\_max : float

The vif of all model variables (excluding the intercept term) must be less than or equal to the threshold

Variables forced into the module are not affected by this constraint.

If a non-mandatory variable is introduced into the model and causes the vif of a mandatory variable whose vif is originally less than the threshold to exceed the threshold, the non-mandatory variable will not be introduced into the model. However, if the vif of a mandatory variable itself exceeds the threshold, that is, the vif caused by other mandatory variables exceeds the threshold, it will not affect the introduction of the non-mandatory variable.

None: No restrictions are imposed on the vif of the input model variable

Default: 3

corr\_max : float

The correlation coefficients between all input variables must be less than or equal to the threshold

If the correlation coefficient of a non-mandatory variable with a mandatory variable exceeds the threshold, the non-mandatory variable will not be introduced into the model.

Even if the correlation coefficient between two mandatory variables is above this threshold, both variables will be included.

None: No restriction on the correlation coefficient of the model variables

Default: 0.8

coef\_sign : dict

Sign constraints on variable coefficients

ex. {"x1":"+","x2":"-"} or file://xx/xx.json read from the file

Value Description:

+ The coefficient of this variable is positive

- The coefficient of this variable is negative

None This variable does not constrain the coefficient sign

coef\_sign = None: No constraints are placed on the coefficient signs of all variables

Variables that the user forces to be included in the model are not subject to this constraint

If the introduction of a non-mandatory variable causes a mandatory variable that originally satisfied the symbol constraint to no longer satisfy the symbol constraint, the non-mandatory variable cannot be included in the model. If the symbol of the mandatory variable itself does not satisfy the constraint, the introduction of the non-mandatory variable will not be affected.

Default: None

default\_coef\_sign : str

When a variable is not in coef\_sign, the default value of the variable sign constraint is

None: The default value of all variables is None

Default: None

iter\_num : int

Maximum number of iterations

Each iteration has two operations:

1. Find a variable from all the remaining variables that meets the constraints and whose addition will improve the model index by the highest amount compared to the current one. Introduce this variable into the model

2. Find a variable from all the variables in the model that meets the constraints and whose removal will improve the model index by the highest amount. Remove this variable from the model.

If in round N (N < iter\_num), adding or removing variables cannot further improve the performance of the model, the iteration is terminated early.

Default: 20

kw\_algorithm\_class\_args : dict

Additional parameters passed to the underlying regression algorithm.

Default: None.

n\_core : int or float

>1: specifies the number of CPU cores

=1: Do not use multi-process

<1: Actual number of cores = total number of CPU cores \* n\_core rounded down

None: Actual number of cores = total number of CPU cores - 1

Default: None.

results\_save : str

The file name that records the modeling process. In addition to common information, the file also records the process of variable selection and elimination in stepwise regression, as well as the reasons for variable deletion.

None: Do not record the process

Default: None.

#### fit

fit(self)

After constructing the LogisticReg object, you need to call the fit method to perform a bidirectional stepwise regression.

Returns

-------------------------------------------------------------------------------

in\_vars : list

Model variables

clf\_final : statsmodels.genmod.generalized\_linear\_model.GLMResults like

The main method of clf\_final:

predict(X) The regression value predicted by the model

The main attributes of clf\_final are:

intercept\_ intercept term

coef\_ Coefficient of each variable (excluding the intercept term)

tvalues The t statistic of each model variable. Where const is the t statistic of the intercept term

pvalues Two-tailed P-value of each model variable

resid\_pearson Pearson residual

resid\_deviance residual deviation

clf\_perf : DataFrame

Model building information: Link Function, Df Residuals, Method (optimization algorithm), AIC, BIC, Log-Likelihood, LL-Null, Deviance, Pearson chi2, Scale, etc.

clf\_coef : DataFrame

Information summary of model parameters: Coef, Std.Err, coefficient test Wald statistic, Wald statistic Pvalue, confidence interval, Standardized Coefficients

del\_reason : Series

The reason for deletion of each deleted variable

step\_proc : DataFrame

Detailed records of each round of modeling, including: adding or removing variables, model performance indicators

# Index

## Functions

### AUC

AUC(target, score, weight=None, target\_label=None)

Calculating the AUC metric

Support weight

Support target value customization

Parameters

----------

target : array like

The actual target.

score : array like

Predicted value

sample\_weight : array like

The weight of the sample

None: All weights are 1

Default: None.

target\_label : dict

Define which value in target means the event has occurred, and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Default: {'unevent':0,'event':1}.

Returns

-------

float

Prediction AUC value

### KS

KS(target, score, sample\_weight=None, target\_label=None)

Calculating the KS indicator

Support weight

Supports custom target values

Parameters

----------

target : array like

The actual target.

score : array like

Predicted value

sample\_weight : array like

The weight of the sample

None: All weights are 1

Default: None.

target\_label : dict

Define which value in target means the event has occurred, and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Default: {'unevent':0,'event':1}.

Returns

-------

float

Predicted KS value

### LIFTn

LIFTn(target, pred, n=10, weight=None, score\_reverse=True, target\_label=None)

Calculating the LIFT metric

Support weight

Support target value customization

Parameters

----------

target : array like

The actual target.

pred : array like

Predicted value

n : int

Specify the percentile of LIFT

weight : array like

The weight of the sample

None: All weights are 1

Default: None.

score\_reverse : bool

True: The larger the pred, the lower the event rate

False: The smaller the pred, the lower the event rate

target\_label : dict

Define which value in target means the event has occurred, and which value means the event has not occurred.

The value of keys can only be unevent or event

The value of values should be filled in according to the value of y

Default: {'unevent':0,'event':1}.

Returns

-------

float

The value of LIFTn

### PSI\_by\_dat

PSI\_by\_dat(Ddat, threshold\_distr=0.05, min\_distr=0.02, cutby=0, Dweight=None, spec\_value=[], min\_spec\_dist=0)

To calculate the PSI between multiple single-column data sets, first perform bin node calculations according to the specified data set. Then split all data sets according to the node and calculate the distribution. Finally, call the PSI\_by\_dist method to obtain the PSI values and other related information of each pair of data sets.

Parameters

----------

Ddat : dict<str,Series>

Multiple single-column datasets

threshold\_distr : int or float

Greater than 1: Divide into several parts

Less than 1: What is the proportion of each portion?

Default: 0.05

min\_distr : float

The minimum acceptable proportion. Due to data skew, it is not guaranteed that all bins can meet threshold\_distr. Some bins may be higher than threshold\_distr, and some may be lower than threshold\_distr. However, the lowest will not be lower than min\_distr.

cutby : int ,str, list

int: If datas is ndarray like, then cutby is based on the number of columns to group.

str: If datas is dict or DataFrame, group them according to the corresponding benchmark series of cutby

list:cutby is bins, all series are grouped according to cutby

Default: 0.

Dweight : dict<str,Series>

The weight of each series

None: Each data point has a weight of 1

Default: None

spec\_value : list

Special value range

Example. ["{-9997}","{-9999,-9998}"]

default:[].

min\_spec\_dist : float

If the proportion of special values in each data set is less than min\_spec\_dist, the special values will not be included in the calculation of PSI.

Default: 0

Returns

-------

float:

The maximum PSI between two datasets

DataFrame:

Calculate the intermediate data of PSI between two data sets

DataFrame:

Summary of the results, including which two datasets produced the maximum PSI

DataFrame:

PSI between two datasets.

### PSI\_by\_dist

PSI\_by\_dist(Ddist, spec\_value=[], min\_spec\_dist=0)

Given the distribution of each single column data set, calculate the PSI between the data sets

Example:

from Index import PSI\_by\_dist

import pandas as pd

label = ['[0.0,2.0)','[2.0,3.0)','[3.0,4.0)',['[4.0,22.0]','{-9993,-9994}'], '{-9996,-9997,-9998,-9999}']

d1 = pd.Series(index=label,data=[0.6497,0.0943,0.0422,0.0346,0.1792])

d2 = pd.Series(index=label,data=[0.6286,0.0960,0.0428,0.0410,0.1916])

d3 = pd.Series(index=label,data=[0.6417,0.0844,0.0478,0.0445,0.1816])

dists = {'d1':d1,'d2':d2,'d3':d3}

psi\_max,psi\_df,dist\_df,psi\_values = PSI\_by\_dist(dists)

psi\_max: 0.0044

psi\_df:

d1 d2 PSI SUM\_PSI d1 d3 PSI SUM\_PSI d2 d3 PSI SUM\_PSI

[0.0,2.0) 0.6497 0.6286 0.000697 0.002651 0.6497 0.6417 0.000099 0.004418 0.6286 0.6417 0.000270 0.003139

[2.0,3.0) 0.0943 0.0960 0.000030 0.002651 0.0943 0.0844 0.001098 0.004418 0.0960 0.0844 0.001494 0.003139

[3.0,4.0) 0.0422 0.0428 0.000008 0.002651 0.0422 0.0478 0.000698 0.004418 0.0428 0.0478 0.000552 0.003139

[[4.0,22.0],{-9993,-9994}] 0.0346 0.0410 0.001086 0.002651 0.0346 0.0445 0.002491 0.004418 0.0410 0.0445 0.000287 0.003139

{-9996,-9997,-9998,-9999} 0.1792 0.1916 0.000830 0.002651 0.1792 0.1816 0.000032 0.004418 0.1916 0.1816 0.000536 0.003139

dist\_df:

d1 d2 d3 PSI\_MAX MAX\_LOC

[0.0,2.0) 0.6497 0.6286 0.6417 0.0044 d1 , d3

[2.0,3.0) 0.0943 0.0960 0.0844 0.0044 d1 , d3

[3.0,4.0) 0.0422 0.0428 0.0478 0.0044 d1 , d3

[[4.0,22.0],{-9993,-9994}] 0.0346 0.0410 0.0445 0.0044 d1 , d3

{-9996,-9997,-9998,-9999} 0.1792 0.1916 0.1816 0.0044 d1 , d3

psi\_values:

data\_name\_1 data\_name\_2 PSI

0 d1 d2 0.002651

1 d1 d3 0.004418

2 d2 d3 0.003139

Parameters

----------

Ddist : dict<str,Series>

Distribution information of multiple single-column data sets. The distribution nodes between multiple data sets need to be consistent

spec\_value : list

Special value range

Example. ["{-9997}","{-9999,-9998}"]

min\_spec\_dist : float

If the proportion of special values in each data set is less than min\_spec\_dist, the special values will not be included in the calculation of PSI.

Default: 0

Returns

-------

float:

The maximum PSI between two datasets

DataFrame:

Intermediate data for calculating PSI between two datasets

DataFrame:

Summary of the results, including which two datasets produced the maximum PSI

DataFrame:

PSI between two datasets.

### VIF

VIF(df)

Calculate the VIF of a variable

Parameters

----------

df : DataFrame

Multi-column variables

Returns

-------

Series

The VIF value for each variable.

# Lan

By changing the value of lan, you can switch the language

Example: How to add German

from Lan\_GER import GER

lan = GER

After the change, the language of all output results will become German