Bins

Calculates the optimal binning split points for rasc . The optimal split points calculated by Bins are mathematically proven global optimal analytical solutions. For categorical variables, including ordered and unordered categories, mathematically proven global optimal analytical solutions can also be calculated. Its main functions are:

1. Find the global optimal solution with or without constraints. Supported constraints: monotonic constraints (automatically determine increasing or decreasing), monotonically decreasing constraints, monotonically 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 prevent the formation of bins with too small differences. This allows users to intuitively understand the size of the differences between bins. This feature is also supported for categorical variables.
4. Do not replace the minimum value of the first bin with negative infinity, nor the maximum value of the last bin with positive infinity. This ensures that outliers are not masked by extending extreme values to infinity. RiskActuarialScoreCard also provides a comprehensive mechanism to handle online values exceeding modeling boundaries. This resolves the common conflict between the need to detect outliers as early as possible during data analysis and the need to mask them in online applications (to ensure that the process is not interrupted but to issue timely alerts).
5. Introduce the concept of wildcards to solve the problem that the online values of categorical variables exceed the modeling value 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. The ScoreCard module automatically calls the Bins module based on the configuration file. However, because RiskActuarialScoreCard is a pluggable component, advanced users can use the Bins module independently, just like any other 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,optBin_prcs=2)

Performs global optimal binning on a variable. Supports multiple features, please refer to the introduction of 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', rasc automatically selects a monotonicity constraint from L+, L-, Uu, and Un. When automatically selecting a constraint, it verifies the correctness of the constraint using different data sets.

2. If the user has already calculated the monotonicity of the suggestion elsewhere, the result can be passed directly to the _x1OptBin function via 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 the monotonic 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 a 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, defining the things users care about most as events is easier to explain. For example, if you want to emphasize the incidence of lung cancer, you can say that smokers have a 50% higher incidence of lung cancer than 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 smokers have a 50% lower incidence of lung cancer than non-smokers. Obviously, the first 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 to pass in Series or dict. 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. All datasets except the one corresponding to train_name 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 globally 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 highest IV value globally under linear monotonic 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 highest IV value globally under U-type constraints (automatically determines Uu, Un)

Default: 'N'.

sgst_mono:tuple(str,**)

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

Recommended constraint: if mono is set to 'A' and sgst_mono is not None, rasc automatically sets mono to the first element of sgst_mono to avoid repeated calculations.

When using rasc automation, this parameter can be used to avoid repeated calculations 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 for 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 very 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 is used here as an intuitive indicator 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 a special value is merged into a bin for a normal value due to a merging rule, the merged bin is a normal bin.
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 the maximum number of bins for the variable
Default: None.

spec_value : list
Special value range
Example. ["{-9997}","{-9999,-9998}"]
"{... , ...}" will not be parsed into a set, but will be processed as a string. {} in special values represents 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 is assigned. For example, for the number of court executions, -9997 might mean that the ID card is not in the citizen database, rather than that it has been executed -9997 times. Through this example, users can clearly see the difference in meaning between -9997 and values like 0, 1, and 2.
"{-9998,-9999}": When a variable takes on the value -9998 or -9999, it has a special meaning. Although these two meanings are different, for the business being modeled, they can be treated as the same and handled according to

the same business logic. For example, when collecting data, data not collected due to Party A's fault is marked as -9998, while data not collected due to Party B's fault is marked as -9999. However, for the business, both values indicate randomly missing data, so they are handled according to the logic of randomly missing data. This preserves the original data's value conventions for retrospective use and saves users from the additional code required to process data.

"{None}" is a special value that means it's a null value. {None} is used instead of words like {miss} because the mechanisms for generating null values and missing values are sometimes different. Missing values represent missing data due to reasons beyond human control during the sampling process, such as a network outage or equipment failure during data transmission. This is a form of missing at random. Null values can also be caused by non-information missingness, such as a lack of loan history, a health checkup not required, or temperatures too low for equipment to collect. Null values themselves contain information. Avoid combining informative null values with missing at random null values into a single special value.

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.

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 distribution 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 proportion 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 merging 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 the lowest 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: The following example explains the meaning of special value writing: ["{-9997}","{-9998,-9999}"] means: there are three special values -9997, -9998, and -9999 in the variable. According to the business meaning, they are divided into two business groups "{-9997}" and "{-9998,-9999}". -9997 itself becomes a business group, and -9998 and -9999 form a business group. Since 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 type of consolidation differs from consolidating -9998 and -9999 into a single business group. The consolidation described in the business group context is at the business level, determined based on business understanding. The consolidation of ["{-9997}","{-9998,-9999}"] is at the data level, determined solely by calculating event rates. It's important to understand the process and underlying meaning behind how the three special values -9997, -9998, and -9999 become the two special value business groups "{-9997}","{-9998,-9999}", and finally become the single special value consolidation bin ["{-9997}","{-9998,-9999}"]. This approach to handling special values adheres to 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 incidence.

Default: False.

no_wild_treat : str

When a categorical 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 (the larger value is taken when the number is 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

Sets 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 within 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, denoted by **. Values not covered in the training set may be seen in other datasets, and these categories are also included in the wildcard category.

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 low frequency of the variable will not be wildcarded.

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

Default: None.

cust_bin : list

User-defined binning.

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 they appear.

Default: None.

optBin_prcs:int

For certain long-tail distribution variables, if you need to calculate the global optimal binning (highest IV), it will take a long time, which may not be worth spending too much time compared to a slightly improved IV.

rascpy provides users with three usage scenarios:

Scenario 1: The bin IV is the highest, but the wait time is long. This scenario is suitable for high-performance CPUs, a large number of cores, a small number of variables, or when you need to achieve the best bin IV or use unattended modeling.

Scenario 2: It is possible to reduce IV by a small amount, but it will reduce the running time.

Scenario 3: Further reduce IV to further reduce runtime. This is suitable for users who need to see binning results quickly or have a low CPU configuration or a large number of variables.

Note: There is no difference in IV between the three scenarios for most variables, and there are only slight differences for variables with certain specific distributions.

Note: There is no significant difference in running time between the three scenarios for most variables, and there are only large differences for variables with certain specific distributions.

Default: 2

Returns

----

optBin : list<str>

Returns the globally optimal bin 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={},optBin_prcs=2)

_x1OptBin calculates the global optimal binning for a single variable, while OptBin calculates the global optimal binning for multiple variables. OptBin is accomplished by calling _x1OptBin.

Parameters

X_dats : DataFrame or dict<dat_name,DataFrame>. Required

Multi-column variables or multiple identical variables belonging to different datasets

There are two points to note:

1. When mono['one var name'] = 'A', rasc automatically selects a constraint for 'one var name' from L+, L-, Uu, and Un. When automatically selecting a constraint, it uses 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 pa
ssed directly to the OptBin function through the parameter sgst_monos. When setting mono['one var na
me'] = 'A', the calculation will not be repeated, 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 N
one, 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. For details on when to pass a Series or a dict, see X_dats.

y_label : dict

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

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, defining the things users care about most as events is easier to explain. For example, if you want to emphasize the incidence of lung cancer, you can say that smokers have a 50% higher incidence of lung cancer than non-smokers. In this case, you can write {'unevent':'No lung cancer','event':'Lung ca
ncer'}. 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 smokers have a 50% lower incidence of lung cancer tha
n non-smokers. Obviously, the first 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. All datasets except the one corresponding to train_name 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 highest IV value globally under linear monotonic 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 highest IV value globally under U-type constraints (automatically determines Uu, Un)

default:{}.

default_mono : str

Variables not listed in mono are assumed to be monotonic.

Default: 'N'.

sgst_monos:tuple

The recommended monotonic trend is: if mono['one var name']='A' and sgst_monos['one var name'] is not None, rasc will automatically set mono['one var name'] to the first element of sgst_monos['one var name']. Avoid repeated operations

When using rasc 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 for 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 very 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

Variables not appearing in distr_min have a default minimum distribution share.

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 impossib le to calculate how small the difference is.

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

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

default:{}.

default_rate_gain_min : float

If a variable does not appear in rate_gain_min, it defaults to the minimum difference in event rates between any two adjacent bins.

Default: 0.001.

bin_cnt_max : dict

The maximum number of bins for each variable. The bins for special values are not counted. If the special value is merged into the bin of the normal value due to the merging rule, the merged bin is the normal bin.

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

Because Bins 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 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 very 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

If a variable does not appear in bin_cnt_max, the default maximum number of bins is used.

Default: None.

spec_value : dict

The range of special values for each variable

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

"{... , ...}" will not be parsed into a set, but will be processed as a string. {} in special values represents 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 is assigned. For example, for the number of court executions, -9997 might mean that the ID card is not in the citizen database, rather than that it has been executed -9997 times. Through this example, users can clearly see the difference in meaning between -9997 and values like 0, 1, and 2.

"{-9998,-9999}": When a variable takes on the value -9998 or -9999, it has a special meaning. Although these two meanings are different, for the business being modeled, they can be treated as the same and handled according to the same business logic. For example, when collecting data, data not collected due to Party A's fault is marked as -9998, while data not collected due to Party B's fault is marked as -9999. However, for the business, both values indicate randomly missing data, so they are handled according to the logic of randomly missing data. This preserves the original data's value conventions for retrospective use and saves users from the additional code required to process data.

"{None}" is a special value that means it's a null value. {None} is used instead of words like {miss} because the mechanisms for generating null values and missing values are sometimes different. Missing values represent missing data due to reasons beyond human control during the sampling process, such as a network outage or equipment failure during data transmission. This is a form of missing at random. Null values can also be caused by non-information missingness, such as a lack of loan history, a health checkup not required, or temperatures too low for equipment to collect. Null values themselves contain information. Avoid combining informative null values with missing at random null values into a single special value.

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

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.

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 spe cial value bin will be merged with other bins. For specific merging rules, see the spec_comb_policy setti ng.

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

If a dict, use the same minimum distribution weight 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 r atio 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 merging 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 the lowest 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 separate 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, all special values of all variables are equivalent to "N" (except variables that can be overwritten by ${default_spec_comb_policy})

Note: This example explains the meaning of special value notation. ["{-9997}","{-9998,-9999}"] means: There are three special values in the variable -9997, -9998, and -9999. Based on the business meaning, they are divided into two business groups "{-9997}" and "{-9998,-9999}". -9997 itself becomes a business group, and -9998 and -9999 form a business group. 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 type of consolidation differs from consolidating -9998 and -9999 into a single business group. The consolidation described in the business group context is at the business level, determined based on business understanding. The consolidation of ["{-9997}","{-9998,-9999}"] is at the data level, determined solely by calculating event rates. It's important to understand the process and underlying meaning behind how the three special values -9997, -9998, and -9999 become the two special value business groups "{-9997}","{-9998,-9999}", and finally become the single special value consolidation bin ["{-9997}","{-9998,-9999}"]. This approach to handling special values adheres to 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 spec ial value merging strategy

If a dict, a default strategy is specified 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 va lue order is set to None, the lexicographic order of the characters is used as the order.

For ordered variables, adjacent nominal orders can only appear in the same bin or at the beginning an d 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 wil dcard categories and are represented by **. When the lexicographical order is used as the order, there are no wildcard categories.

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 t hreshold are merged into the wildcard category. If the threshold of a variable is None, the category wit h 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 data sets, it is possible to see values that are not covered in the training set, and these categ ories 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 categorical variable does not have a wildcard and uncovered categories appear, the specified pr ocessing 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 (the larger value is taken when the numb er is 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

When a variable has no wildcards and is not configured in no_wild_treat, the default treatment for unc overed categories.

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}"]}

optBin_prcs:int

For certain long-tail distribution variables, if you need to calculate the global optimal binning (highest IV), it will take a long time, which may not be worth spending too much time compared to a slightly improved IV.

rascpy provides users with three usage scenarios:

Scenario [1: The given bin IV is the highest, but requires a long wait. Suitable for high CPU performance, a large number of cores, or a small number of variables, or for pursuing the ultimate bin IV, or for unattended modeling.

Scenario 2: It is possible to reduce IV by a small amount, but it will reduce the running time.

Scenario 3: Further reduce IV to further reduce runtime. This is suitable for users who need to see binning results quickly or have a low CPU configuration or a large number of variables.

Note: There is no difference in IV between the three scenarios for most variables, and there are only slight differences for variables with certain specific distributions.

Note: There is no significant difference in running time between the three scenarios for most variables, and there are only large differences for variables with certain specific distributions.

Default: 2

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,optBin_prcs=2)

Multi-process version of OptBin

Parameters

----

cores : int

The number of CPU cores used.

None: Use all cores

Default: None.

Other parameters: See [Bins. OptBin]

Returns

----

optBins : dict<str,list<str>>

Same return value as [Bins. 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)

Performs equal-frequency binning on the series. With the support of the Cutter component, it produces more uniform segmentation points than existing equal-frequency binning software libraries.

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

Supports multiple features, please refer to the introduction of [Bins]module

Parameters

----

x : Series

A list of variables

weight : Series

Sample weight

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 categorical variable does not have a wildcard and uncovered categories appear, the following pr ocessing 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 (the larger value is taken when the numb er is 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

Sets 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 ord er.

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

If the nominal order is inconsistent with the event rate order, you can decide for yourself whether to c onfigure 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 wil dcard categories, denoted by **. Values not covered in the training set may be seen in other datasets, and these categories are also included in the wildcard category.

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 when calculating 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 occu rred.

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 low frequency of the variable will not be wildcarded.

In other data sets, it is possible to see values that are 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 split 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 a variable, while 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 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 when calculating 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 occu rred.

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 va lue order is set to None, the lexicographic order of the characters is used as the order.

For ordered variables, adjacent nominal orders can only appear in the same bin or at the beginning an d 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 wil dcard categories and are represented by **. When the lexicographical order is used as the order, there are no wildcard categories.

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 t hreshold are merged into the wildcard category. If the threshold of a variable is None, the category wit h 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 data sets, it is possible to see values that are not covered in the training set, and these categ ories 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 categorical variable does not have a wildcard and uncovered categories appear, the specified 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 (the larger value is taken when the number is 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

When a variable has no wildcards and is not configured in no_wild_treat, the default treatment for uncovered categories.

Default: None.

Returns

----

freqbins: dict<str,list<str>>

The equal-frequency cutoff 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 used.

None: Use all cores

Default: None.

Other parameters: see Bins.FreqBin

Returns

----

freqbins: dict<str,list<str>>

The equal-frequency cutoff 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 ratio. If y is not None, the event 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

Sample weight

None: All weights are 1

Default: None

sync_bins : bool

True: Update bins when the extreme value of X exceeds the extreme value of the cut point

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

Default: True

y_label : dict

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

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>

Segment-wise 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, then 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 the 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)

of multiple variables, which is done by calling Bins._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

Parameters

----

cores : int

The number of CPU cores used.

None: Use all cores

Default: None.

Other parameters: see Bins.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

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, it returns the IV of each variable.

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 type

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

Target 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 will be 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. Unordered categories 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 categorical 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 (the larger value is taken when the number is 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

Sets 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, denoted by **. Values not covered in the training set may be seen in other datasets, and these categories are also included in the wildcard category.

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

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

Default: None.

When is_cate=False or is_order=False, this parameter will be ignored.

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 give a suggested 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 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

Target 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 va lue 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 wil dcard categories and are represented by **. When the lexicographical order is used as the order, there are no wildcard categories.

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 ele ment 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 categorical variable does not have a wildcard and uncovered categories appear, the specified 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 (the larger value is taken when the number is 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

When a variable has no wildcards and is not configured in no_wild_treat, the default treatment for uncovered categories.

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 recommendation, 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 used.

None: Use all cores

Default: None.

Other parameters: See Bins. MonoSuggest

Returns

----

dict<var_name,tuple(monotonicity_suggestion,**)>

The first element is a monotonicity recommendation, 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 Python segmenter:

1. Mathematically provable analytical solution with minimum global error.
2. All split points come from the original values.

3. More user-friendly 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 derived from the original data, unlike Python's built-in splitter, which modifies the extreme values at both ends.

4. It can also provide the global optimal segmentation solution for extremely tilted data.

5. Support weighted series.

6. Supports user-specified special values. Special values are grouped separately, and users can also c onfigure multiple special values to be combined into one group.

7. If the special value does not contain null values, but the sequence contains null values, 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 b e automatically expanded.

It is recommended to try replacing Python's built-in equal frequency segmentation component with Cutte r.

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

Functions

_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
Fals e: descending bins

_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 that meet the artificially set merging rules 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 the 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 percentage. Due to data skew, it is not guaranteed that all bins will meet threshold_distr. Some bins may exceed threshold_distr, while others may fall below it. However, the minimum percentage will not fall below 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

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 acceptable percentage. Due to data skew, it's not guaranteed that all bins will meet threshold_distr. Some bins may exceed threshold_distr, while others may fall below it. However, the minimum percentage will not fall below 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

Automatically expand the extreme value of cutby corresponding data

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 acceptable percentage. Due to data skew, it's not guaranteed that all bins will meet threshold_distr. Some bins may exceed threshold_distr, while others may fall below it. However, the minimum percentage will not fall below min_distr.

cutby : int ,str, list

int: If datas is ndarray like, then cutby is grouped based on the first column.

str: If datas is a dict or DataFrame, it is grouped according to the corresponding benchmark series of c utby

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 benchmark 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 bin corresponding to the data point 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.

auto_round

auto_round(dats,ex_cols=[],default_spec_value=[],spec_values={},default_trunc_int=True,trunc_ints={})

Automatically determine the precision for each column in the dataframe and convert it into a new dataframe with the calculated precision.
The premise of the precision conversion of each column is not to reduce the IV value of its bin, otherwise the original data remains unchanged.
Parameters
----
dats : DataFrame
Multiple sequences to be converted

ex_cols: list
Variables in ex_cols are not converted to the same precision

default_spec_value : list
Default special value. If a special value of an array is not in spec_values , default_spec_value is used.
If a special value is not in the sequence, it will be automatically ignored
Example. ["{-9997}","{-9999,-9998}"]

spec_values : dict
Set a separate special value for the variable, and no precision conversion will be performed on the special value
Example: {'X1':["{-9997}","{-9999,-9998}"],"X2":["{-9999}"]}
If a special value is not in the sequence, it will be automatically ignored

default_trunc_int : bool
Whether to use the default value for integer precision conversion. If the variable is not in trunc_ints, the default value is used
Integer conversion example: 10212 -> 10210
10212 -> 10200
The premise of automatic adjustment of integer precision is also not to reduce the IV of its binning

trunc_ints : dict(str, bool)
Whether to perform precision conversion on integers.
Example: {'x1':True,'x2':False}
Integer conversion example: 10212 -> 10210

10212 -> 10200
The premise of automatic adjustment of integer precision is also not to reduce the IV of its binning

Returns
----
dataframe
Dataframe after automatic precision conversion

auto_round _one

auto_round_one(data,spec_value,trunc_int = True)

Automatically determine the precision of a column of data and convert each value in the column to that precision. The principle of automatic accuracy determination is not to reduce the IV of its bins, otherwise the original data remains unchanged.

Parameters
----
data : array like
A series of numbers

spec_value : list
The value of a special value. Special values will not be converted to a specific value.
If a special value is not in the sequence, it will be automatically ignored
Example. ["{-9997}","{-9999,-9998}"]

trunc_int: bool
Whether to automatically adjust integer variables
For example: 10212 -> 10210
10212 -> 10200
The premise of automatic adjustment of integer precision is also not to reduce the IV of its binning

Returns
----
array like has the same type as data
Sequence after automatic precision reduction

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 lexicographic 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 of 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 into numerical 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 handled according to the rules of special values.

default:[]

wild : str

Wildcard identifier

default:'**'

no_wild_treat : str

When a categorical variable does not have a wildcard and uncovered categories appear, the specified pr ocessing 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 (the larger value is taken when the numb er is even)

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

Default: M

order_list : tuple

Sets 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 ord er.

If the nominal order is inconsistent with the event rate order, you can decide for yourself whether to c onfigure 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 wil dcard categories, denoted by **. Values not covered in the training set may be seen in other datasets, and these categories are also included in the wildcard category.

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, targ et 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 occu rred.

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 a distribution ratio less th an the threshold into wildcard categories. If is_order=False and unorder_combine_thv=None, categories wi th too low a frequency for 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 numeric 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,**>']

Impute

Classes

BCSpecValImpute

Common missing value imputation methods can only handle missing values, but cannot handle special values, especially when data contains both missing and special values. Special values cannot be simply equated with missing values. Simply treating special values as missing values without considering the business scenario will lead to information loss. Special values transform numeric data into a complex data type that mixes categorical and numerical data. Currently, no model can directly handle this data (although some models can produce results, they are inaccurate and meaningless). The Impute package provided by rascpy can solve this problem. The transformed data can be directly fed into any model and meet practical business requirements.
BCSpecValImpute can be used to handle special values and missing values in the data of binary classification problems.
It can handle special values and missing values for continuous, unordered categorical, and ordered categorical variables.
BCSpecialsImpute not only fills empty values, but also converts special values that the model cannot handle.
It uses the special value merging method of the Bins module in rascpy (an optimal binning algorithm that uses mathematics to ensure that its IV is the highest) to merge special values with normal values, and then sets the special value to the mean of all normal values in the bin it is merged with (for continuous variables) or randomly selects a category (for categorical variables. Since all categories in the bin are considered to be the same category, they can be randomly selected)

__init__

__init__(self,spec_value={},default_spec_value=[],order_cate_vars={},unorder_cate_vars={},impute_None=True,cores=None)

Parameters
----

spec_value : dict
The range of special values for each variable
ex. {"x1":["{-9997}","{-9999,-9998}"],"x2":["{None}"]}
"{... , ...}" will not be parsed into a set, but will be processed as a string. {} in special values represents 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 is assigned. For example, for the number of court executions, -9997 might mean that the ID card is not in the citizen database, rather than that it has been executed -9997 times. Through this example, users can clearly see the difference in meaning between -9997 and values like 0, 1, and 2.
"{-9998,-9999}": When a variable takes on the value -9998 or -9999, it has a special meaning. Although these two meanings are different, for the business being modeled, they can be treated as the same and handled according to the same business logic. For example, when collecting data, data not collected due to Party A's fault is marked as -9998, while data not collected due to Party B's fault is marked as -9999. However, for the business, both values indicate randomly missing data, so they are handled according to the logic of randomly missing data. This preserves the original data's value conventions for retrospective use and saves users from the additional code required to process data.
"{None}" is a null special value or missing value. For the difference between the two, see the section about outliers, missing values, and special values. {None} is used instead of {miss} because the mechanisms for generating null values and missing values are sometimes different. A missing value indicates that due to reasons beyond human control during the sampling process, a sample point was not collected, resulting in missing data information. For example, a network outage or equipment failure during data transmission may have prevented data from being collected. This is a form of missing at random. Null values can also be caused by non-information missingness, such as a lack of loan records, a health check not requiring a specific medical examination, or a temperature too low for the equipment to collect data. Null values themselves contain information. Do not combine informative null values and missing at random null values into a single special value.
ex. {"x1":"{None,-9997}"} This means that after analyzing the business, null values and -9997 can be handled the same way for this modeling.
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.
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 orders 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 and are represented by **. When the lexicographical order is used as the order, there are no wildcard categories.
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 data sets, it is possible to see values that are 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:{}.

impute_None : bool
Whether to fill null values, because some models can automatically handle null values. If you want to use such models later, you don't need to handle null values when filling.
If the nulls in the data are not randomly missing but business missing, representing a state and the subsequent model can automatically handle null values, it is recommended to set this parameter to False
True: fill empty values
False: Do not fill in null values and let the subsequent model handle them (if the model can handle null values)
Default: True

cores : int
The number of CPU cores used.
None: Use all cores
When it is less than 0, it reserves the number of CPU cores, i.e. os.cpu_count() - cores
Default: None.

Returns

None.

fit

fit(self,X,y,weight=None,y_label={'unevent':0,'event':1})
Train each missing value filling method for each variable (if impute_None=True) and each special value conversion method

Parameters

X : DataFrame
Data that needs to be converted
If a variable is not configured with a special value, or does not contain the configured special value, the variable will be automatically ignored.

y : Series
target

weight : Series, optional
Weight
Default: None.
y_label : dict, optional
The meaning of the target tag
Default: {'unevent':0,'event':1}.

Returns

None.

transform

transform(self,X)

Fill missing values and convert special values. Need to be called after fit.

Parameters

X : DataFrame

Data that needs to be populated and transformed

Returns

DataFrame

The padded and transformed data.

fit_transform

fit_transform(self,X,y,weight=None,y_label={'unevent':0,'event':1})
First call fit training, then call transform conversion
See fit, transform

Index

Functions

AUC

AUC(target, score, weight=None, target_label={'unevent':0,'event':1})

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

Sample weight

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={'unevent':0,'event':1})

Calculating the KS indicator

Support weight

Support target value customization

Parameters

----

target : array like

The actual target.

score : array like

Predicted value

sample_weight : array like

Sample weight

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={'unevent':0,'event':1})

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

Sample weight

None: All weights are 1

Default: None.

score_reverse : bool

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

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

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_dis t=0.005)

To calculate the PSI between multiple single-column datasets, first perform a binning node calculation fo r the specified dataset. Then, split all datasets according to that node and calculate the distribution. Fin ally, call the PSI_by_dist method to obtain the PSI values and other related information for each pair of datasets.

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 percentage. Due to data skew, it is not guaranteed that all bins will meet threshold_distr. Some bins may exceed threshold_distr, while others may fall below it. However, the minimum percentage will not fall below min_distr.

cutby : int ,str, list

int: If datas is ndarray like, then cutby is grouped based on the first column.

str: If datas is a dict or DataFrame, it is grouped 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 are no

t 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 the maximum PSI between the two datasets

DataFrame:

PSI between two datasets.

PSI_by_dist

PSI_by_dist(Ddist, spec_value=[], min_spec_dist= 0.005)

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 datasets. 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 are no t 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 the maximum PSI between the two datasets

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 of 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 be changed to German

Performance

Functions

perf_summary

perf_summary(datas,target_label={'unevent':0,'event':1},cut_data_name=None,wide=0.05,thin=None,thin_head=10,lift=None,score_reverse=True)

Calculate and summarize the model performance. Including:

1. Divide the output of the model into equal frequency, observe the number, distribution, proportion, cumulative number, cumulative distribution, cumulative proportion, ODDS, Lift and other information of each interval segment and summarize them

2.lift,ks,auc

Parameters

----

datas: dict{str,tuple(y_true,y_hat,weight)}

All datasets for which model performance needs to be summarized.

The key is the name of the dataset, and the value is a tuple structure that stores y_true, y_hat, and weight.

target_label : dict, optional

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}.

cut_data_name : str, optional

According to which dataset the model output is divided into equal frequency groups.

None: Perform equal frequency division according to the distribution of each data set

Whether to use the same dataset or separate datasets to calculate equal-frequency splits depends on the user's focus and business needs. Using the same dataset (usually the train dataset) not only reflects model performance, but also reflects the stability of the model output and the differences in model scores across different datasets. Using separate datasets to calculate equal-frequency splits reflects the model's true performance on each piece of data (usually higher than using the same dataset).

For example, if the model is used to sort applications (e.g., sorting scores and approving applications based on a certain percentage), if the user uses the same threshold for all applications, consider using the same dataset to calculate the split node. If the user customizes different thresholds for different applications, consider using the separate datasets to calculate the split node.

Using the same or different data sets to calculate split nodes requires users to make comprehensive judgments based on their own business application scenarios.

Default: None.

wide : float, optional

The model's output is grouped into equal frequency groups. This parameter is the user's desired proportion of each group. Thanks to the powerful Cutter module, even if the score distribution is skewed, it can still provide the grouping closest to wide.
Default: 0.05.

thin : float, optional
This option has the same meaning as wide, but provides a more detailed breakdown. Some businesses may not only focus on the overall situation, but also on the recognition efficiency (such as recall and precision) of the small subset of events with the highest (or lowest) occurrence rate. This can be achieved by configuring thin. If thin is not None, the function returns two model metric statistics tables for equal-frequency groups: a broader wide model metric table and a narrower thin model metric table.
Default: None.

thin_head : int, optional
The smaller the thin value is, the more equal-frequency groups there are. The narrower the thin model indicator statistics table will be, the longer it will be. It is not very convenient. Usually, the purpose of using thin is just to focus on the head data. Therefore, thin_head can be used to control the length of the thin model indicator statistics table, and only the first thin_head groups are retained.
If thin is None, thin_head is automatically ignored.
If thin_head is None, all thin groups will be retained.
Default: 10.

lift : tuple(int,...), optional
Calculate the corresponding lift value
Example: (1,5,10,20) represents the calculation model lift1, lift5, lift10, lift20
None: Do not calculate the lift of the model
Default: None.

score_reverse: bool , optional
Tell the relationship between the scoring value and the event rate so that the function can give a hum

anized display

True: The higher the probability of an event occurring, the lower the score

False: The higher the probability of an event, the higher the score

Default: True

Returns

----

wide_perfs : dict<str,pd.DataFrame>

Returns the number, distribution, proportion, cumulative number, cumulative distribution, cumulative proportion, ODDS, Lift and other information of each interval segment after each data set is equally grouped according to wide

thin_perfs : dict<str,pd.DataFrame>

Returns the number, distribution, proportion, cumulative number, cumulative distribution, cumulative proportion, ODDS, Lift and other information of each interval segment after each data set is equally grouped according to thin model output.

lifts : dict<str,list>

Returns the lift specified by the user for each dataset

If the user-specified lift is None, this value also returns None.

ks : dict<str,float>

Returns the ks of each data set

auc : dict<str,float>

Returns the auc of each dataset

gen_perf_table_by_pred

gen_perf_table_by_pred(target,pred,pect,target_label=None,weight=None,score_reverse=True)

According to the user-given y_target, y_hat, weight and the specified proportion of each equal-frequency group, the number, distribution, proportion, cumulative number, cumulative distribution, cumulative proportion, ODDS, Lift and other information of each interval segment of the model output are summarized.

Parameters

----

target : pd.Series

y_target

pred : pd.Series

y_true

pect : float

The expected proportion of each group depends on the powerful function of the Cutter module. Even if the score distribution is skewed, it can still give the group closest to the expected proportion.

target_label : dict, optional

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}.

weight : pd.Series

The weight of each sample

None: All samples have the same weight

Default: None.

score_reverse: bool , optional

Tell the relationship between the scoring value and the event rate so that the function can give a hum anized display

True: The higher the probability of an event occurring, the lower the score

False: The higher the probability of an event, the higher the score

Default: True

Returns

----

perf : pd.DataFrame

Returns the number, distribution, proportion, cumulative number, cumulative distribution, cumulative prop ortion, ODDS, Lift and other information of each interval segment after equal frequency grouping of mo del output

label_bins : list

Returns the grouping of model outputs

Reg_Step_Wise_MP

Since Version== 2025.10.2 , Reg_Step_Wise_MP has been replaced by StepwiseRegressionSKLearn. Although Reg_Step_Wise_MP can still be used, it is recommended to use StepwiseRegressionSKLearn because it is more computationally efficient.

It is a linear/logistic two-way stepwise regression implemented in Python, which adds the following featu res to the traditional two-way stepwise regression:

1. When performing stepwise variable selection for logistic regression, AUC, KS, and LIFT metrics can be used instead of AIC and BIC. For some business scenarios, AUC and KS are more relevant. For example, in ranking tasks, a model built using the KS metric uses fewer variables while maintaining the same KS, thereby reducing data costs.
2. When performing stepwise variable selection, use other datasets to calculate model evaluation metrics rather than the modeling dataset. Especially when the data size is large and a validation set is include

d in addition to the training and test sets, it is recommended to use the validation set to calculate evaluation metrics to guide variable selection. This helps reduce overfitting.
3. Supports using partial data to calculate model evaluation metrics to guide variable selection. For example, if a business needs to maintain a certain pass rate of N%, then the bad event rate of the top N% of samples can be minimized, without requiring all samples to be included in the calculation. Actual testing shows that in appropriate scenarios, using partial data as evaluation metrics results in fewer variables than using full data, but the metrics of interest to users remain unchanged. Because the model focuses only on the top, more easily distinguishable sample points, business objectives can be achieved without requiring too many variables.
4. Supports setting multiple conditions. Variables must meet all conditions simultaneously to be included in the model. Built-in conditions include: P-Value, VIF, correlation coefficient, coefficient sign, number of variables in a group, etc.
5. Supports specifying variables that must be entered into the model. If the specified variables conflict with the conditions in 4, a comprehensive mechanism has been designed to resolve the problem.
6. The modeling process is exported to Excel, recording the reasons for deleting each variable and the process information of each round of stepwise regression.
7. Support actuarial calculations, using company profits as a loss function, which simultaneously considers the model's prediction accuracy + the profit level of a single user + the data cost (in the testing phase, there will be significant changes later)

In most cases, users do not need to interact directly with the Reg_Step_Wise_MP component. However, rascpy is designed to be pluggable, so advanced users can use the Reg_Step_Wise_MP module independently, just like any other Python module.

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,exc_group=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, without addition or deletion.

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 modeling weight is set for 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 is used to determine whether the model has improved.

Indicators include: 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 its meaning 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 screening 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 gradually screening 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 w ill 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, the introduction of the non-mandatory variable will not be affected.

None: No constraints are placed 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 included in 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 included in 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, the introduction of the non-mandatory variable will not be affected.

None: No restrictions are placed on the vif of the input variables

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-forced variable with a forced variable exceeds the threshold, the non-forced variable will not be introduced into the model.

Even if the correlation coefficient between two forced 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 has a negative sign

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 entered into the model are not subject to this constraint

If the introduction of a non-mandatory variable causes a mandatory variable that originally satisfied the sign constraint to no longer satisfy the sign constraint, the non-mandatory variable cannot be included i n the module. If the sign of the mandatory variable itself does not satisfy the sign constraint, the intro duction 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 symbol 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 remaining variables that meets the constraints and whose addition will impro ve the model index by the highest amount. Introduce this variable into the model.
2. Find a variable from all the variables in the model that meets the constraints and whose removal wi ll improve the model index more than the current one, and the one with the highest improvement. Re move this variable from the model.

If, at the Nth round (N < iter_num), adding or removing variables does not improve the model's perfor mance further, the iteration terminates 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 recor ds the process of variable selection and elimination in stepwise regression, as well as the reasons for v ariable deletion.

None: Do not record the process

Default: None.

exc_group : str

Exclusive group, only one variable in the group can be entered into the model

ex1. exc_group = _g

ex2. exc_group = g#

Value Description:

The value must have exactly two characters:

A character is any character except g. It is a delimiter for variable names. Group names can be separat ed by this delimiter.

A character can only be g. If g appears in the front, it means the group name is the prefix of the vari able name. If g appears in the back, it means the group name is the suffix of the variable name.

For example, if the variable naming format is x1_gname1, then this should be configured as _g

If the variable naming format is gname1#x1, then this should be configured as g#

If a variable does not contain a separator, it means that the variable is not restricted by the exclusive group constraint.

Default: None. No variable in the data is subject to the constraints of the exclusive group.

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 is 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 residuals 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 interv al, Standardized Coefficients

del_reason : Series

The reason for deletion of each deleted variable

step_proc : DataFrame

Detailed records of each round of modeling process, including: adding or removing variables, and m odel 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,exc_group=None)

Bidirectional stepwise logistic regression. It adds functionality to existing packages that implement bidirect ional stepwise logistic regression. See the introduction of the 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, defining the things users care about most as events is easier to explain. For example, if you want to emphasize the incidence of lung cancer, you can say that smokers have a 50% higher incidence of lung cancer than 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 smokers have a 50% lower incidence of lung cancer than non-smokers. Obviously, the first 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, without addition or deletion.

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 modeling weight i s set for various considerations, such as reducing heteroscedasticity, balancing positive and negative sam ples, and setting different loss costs.

Default: None

measure : str

In bidirectional stepwise regression, an indicator is used 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 fro m fit_weight, and its meaning is usually similar to that of the sample weight.

If measure is aic or bic, the measure_weight configuration is ignored.

Default: None.

measure_frac : float

Sort by the probability of event occurrence from large to small or small to large, and take the first N s ample 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 calcul ate the model evaluation index. Equivalent to measure_frac=1.

If measure_index is aic or bic, the measure_frac configuration is ignored. Only the entire 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 largest to smallest (round down)

-1 <= measure_frac < 0: Take the first N = sample_n*measure_frac*-1 sample points from smallest to la rgest (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 screening variables

None: Use the same dataset as used for modeling

If measure is aic or bic, the measure_X configuration is ignored. Only the same dataset as that used fo r modeling can be used.

Default: None.

measure_y : Series

The y data set used to measure the model performance index when gradually screening variables

None: Use the same dataset as used for modeling

If measure is aic or bic, the measure_y configuration is ignored. You can only use the same dataset as the one 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 o r 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 w hose p-value is originally less than the threshold to exceed the threshold, the non-mandatory variable w ill 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, the introduc tion of the non-mandatory variable will not be affected.

None: No constraints are placed 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 threshol d

Variables forced into the module are not affected by this constraint

If a non-mandatory variable is included in 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 included in 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, the introduction of the non-mandat ory variable will not be affected.

None: No restrictions are placed on the vif of the input variables

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-forced variable with a forced variable exceeds the threshold, the n on-forced variable will not be introduced into the model.

Even if the correlation coefficient between two forced variables is above this threshold, both variables w ill 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 has a negative sign

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 entered into the model are not subject to this constraint

If the introduction of a non-mandatory variable causes a mandatory variable that originally satisfied the sign constraint to no longer satisfy the sign constraint, the non-mandatory variable cannot be included i n the module. If the sign of the mandatory variable itself does not satisfy the sign constraint, the intro duction 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 symbol 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 remaining variables that meets the constraints and whose addition will impro ve the model index by the highest amount. Introduce this variable into the model.
2. Find a variable from all the variables in the model that meets the constraints and whose removal wi ll improve the model index more than the current one, and the one with the highest improvement. Re move this variable from the model.

If, at the Nth round (N < iter_num), adding or removing variables does not improve the model's perfor mance further, the iteration terminates 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.

exc_group : str

Exclusive group, only one variable in the group can be entered into the model

ex1. exc_group = _g

ex2. exc_group = g#

Value Description:

The value must have exactly two characters:

A character is any character except g. It is a delimiter for variable names. Group names can be separated by this delimiter.

A character can only be g. If g appears in the front, it means the group name is the prefix of the variable name. If g appears in the back, it means the group name is the suffix of the variable name.

For example, if the variable naming format is x1_gname1, then this should be configured as _g

If the variable naming format is gname1#x1, then this should be configured as g#

If a variable does not contain a separator, it means that the variable is not restricted by the exclusive group constraint.

Default: None. No variable in the data is subject to the constraints of the exclusive group.

fit

fit(self)

After constructing the LogisticReg object, you need to call the fit method to perform a bidirectional step wise 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 is 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

Model parameter information summary: 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 process, including: adding or removing variables, and model performance indicators .

Report

Functions

write_performance

write_performance(datas,target_label=None,cut_data_name=None,wide=0.05,thin=0.01,thin_head=10,lift=None,score_reverse=True,writer=None,sheet_name=lan['Performance of the model'],filePath=None)

Write the performance indicators of the data set into Excel. Including:

1. Divide the output of the model into equal frequency, observe the number, distribution, proportion, cumulative number, cumulative distribution, cumulative proportion, ODDS, Lift and other information of each interval segment and summarize them

2.lift,ks,auc

Parameters

----

datas: dict{str,tuple(y_true,y_hat,weight)}

All datasets for which model performance needs to be summarized.

The key is the name of the dataset, and the value is a tuple structure that stores y_true, y_hat, and weight.

target_label : dict, optional

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}.

cut_data_name : str, optional

According to which dataset the model output is divided into equal frequency groups.

None: Perform equal frequency division according to the distribution of each data set

Whether to use the same dataset or separate datasets to calculate equal-frequency splits depends on the user's focus and business needs. Using the same dataset (usually the train dataset) not only reflects model performance, but also reflects the stability of the model output and the differences in model scores across different datasets. Using separate datasets to calculate equal-frequency splits reflects the model's true performance on each piece of data (usually higher than using the same dataset).

For example, if the model is used to sort applications (e.g., sorting scores and approving applications based on a certain percentage), if the user uses the same threshold for all applications, consider using the same dataset to calculate the split node. If the user customizes different thresholds for different applications, consider using the separate datasets to calculate the split node.

Using the same or different data sets to calculate split nodes requires users to make comprehensive judgments based on their own business application scenarios.

Default: None.

wide : float, optional

The model's output is grouped into equal frequency groups. This parameter is the user's desired proportion of each group. Thanks to the powerful Cutter module, even if the score distribution is skewed, it can still provide the grouping closest to wide.

Default: 0.05.

thin : float, optional

This option has the same meaning as wide, but provides a more detailed breakdown. Some businesses may not only focus on the overall situation, but also on the recognition efficiency (such as recall and precision) of the small subset of events with the highest (or lowest) occurrence rate. This can be achieve

d by configuring thin. If thin is not None, the function returns two model metric statistics tables for equal-frequency groups: a broader wide model metric table and a narrower thin model metric table.
Default: None.

thin_head : int, optional
The smaller the thin value is, the more equal-frequency groups there are. The narrower the thin model indicator statistics table will be, the longer it will be. It is not very convenient. Usually, the purpose of using thin is just to focus on the head data. Therefore, thin_head can be used to control the length of the thin model indicator statistics table, and only the first thin_head groups are retained.
If thin is None, thin_head is automatically ignored.
If thin_head is None, all thin groups will be retained.
Default: 10.

lift : tuple(int,...), optional
Calculate the corresponding lift value
Example: (1,5,10,20) represents the calculation model lift1, lift5, lift10, lift20
None: Do not calculate the lift of the model
Default: None.

score_reverse: bool , optional
Tell the relationship between the scoring value and the event rate so that the function can give a humanized display
True: The higher the probability of an event occurring, the lower the score
False: The higher the probability of an event, the higher the score
Default: True

writer : writer, optional
A writer for Excel. If you want to combine the current information with other information and output them in the same Excel file, it is more convenient to use the writer. Otherwise, it is more convenient to use filePath.

sheet_name : str, optional

Write the performance indicators of the model to the sheet page

filePath : str, optional

Writer is used first. If writer is None, the Excel file location specified by filePath is used.

Both writer and filePath cannot be None at the same time

Returns

----

wide_perfs : dict<str,pd.DataFrame>

Returns the number, distribution, proportion, cumulative number, cumulative distribution, cumulative proportion, ODDS, Lift and other information of each interval segment after each data set is equally grouped according to wide

thin_perfs : dict<str,pd.DataFrame>

Returns the number, distribution, proportion, cumulative number, cumulative distribution, cumulative proportion, ODDS, Lift and other information of each interval segment after each data set is equally grouped according to thin model output.

lifts : dict<str,list>

Returns the lift specified by the user for each dataset

If the user-specified lift is None, this value also returns None.

ks : dict<str,float>

Returns the ks of each data set

auc : dict<str,float>

Returns the auc of each dataset

write_y_stat

write_y_stat(datas,y_stat_group_cols = None,weight_col=None,y_col='y',y_label=None,writer=None,sheet_name=lan['0113'],filePath=None)

Perform statistics on the Y value distribution of all data sets. If y_stat_group_cols is specified, statistics are performed after grouping according to y_stat_group_cols.

Parameters

----

datas: dict{str,dict{str,dataframe}}

datas is a two-level nested dict structure. The first str is the purpose of the dataset, and the second str is the name of the dataset.

Example: {'model_data':{'train':df_train,'test':df_test},'perf_data':{'client1':df1,'client2':df2},'oot_data':{'time1':df_time1,'time2':df_time2}}

y_stat_group_cols : list, optional

One or more user-specified variables for grouping statistics Y

None: No group is specified

Default: None.

weight_col : str, optional

Sample weight column name.

None: All samples have the same weight

Default: None.

y_col : str, optional

Column name for the sample y values

Default: 'y'.

y_label : dict, optional

Define which value of the y_col column in the data indicates that the event has occurred, and which value indicates that 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}.

writer : writer, optional

A writer for Excel. If you want to combine the current information with other information and output them in the same Excel file, it is more convenient to use the writer. Otherwise, it is more convenient to use filePath.

sheet_name : str, optional

Write the y statistical results to the sheet page

filePath : str, optional

Writer is used first. If writer is None, the Excel file location specified by filePath is used.

Both writer and filePath cannot be None at the same time

Returns

None.

write_feature_select

write_feature_select(indices,filtered_cols,used_cols,filters_middle_data,var_describe_file_path=None,writer=None,sheet_name=lan['Selection of variables'],filePath=None)

Write the results of variable selection into Excel

Parameters

----

indices : dict{str,Series}

Display the indicator values of each variable in the report

key is the indicator name

vaue is the value of each variable on the key indicator

filtered_cols : dict{str,str}

The reason why each variable is filtered out. Multiple reasons are separated by carriage returns.

The reasons include not meeting the threshold of the indicator (list several indicators if they are not met) and logistic regression deletion

used_cols : dict<str,str>

The reason for each variable to be included in the model includes:

Logistic regression introduction, user forced introduction, etc.

filters_middle_data : dict{str,dataframe}

Output the intermediate process data of the variable indicators entered by the user to Excel

str will be used as the name of the sheet

var_describe_file_path : str, optional

Variable description file

The first column of the file records the name of the variable. The name must be consistent with the variable in the model data and the capitalization must be the same.

The columns of the file except the first column will be displayed in the Excel sheet page corresponding to the variable selection, which is convenient for users to view

writer : writer, optional

A writer for Excel. If you want to combine the current information with other information and output them in the same Excel file, it is more convenient to use the writer. Otherwise, it is more convenient to use filePath.

sheet_name : str, optional

Write the results to the sheet page

filePath : str, optional

Writer is used first. If writer is None, the Excel file location specified by filePath is used.

Both writer and filePath cannot be None at the same time

Returns

None.

write_reg

write_reg(reg,var_describe_file_path,writer=None,sheet_name=lan['Model Info'],filePath=None)

Output the return result of StepwiseRegressionSKLearn.LogisticReg/LinearReg.fit() to Excel

Parameters

----

reg: StepwiseRegressionSKLearn.LogisticReg/LinearReg

Instance produced after StepwiseRegressionSKLearn.LogisticReg/LinearReg.fit

var_describe_file_path : str, optional

Variable description file

The first column of the file records the name of the variable. The name must be consistent with the v ariable in the model data and the capitalization must be the same.

All columns except the first column will be displayed in the Excel sheet page corresponding to the mod el selection, which is convenient for users to view.

writer : writer, optional

A writer for Excel. If you want to combine the current information with other information and output them in the same Excel file, it is more convenient to use the writer. Otherwise, it is more convenient to use filePath.

sheet_name : str, optional

Write the results to the sheet page

filePath : str, optional

Writer is used first. If writer is None, the Excel file location specified by filePath is used.

Both writer and filePath cannot be None at the same time

Returns

None.

write_card

write_card(Dcard,base_points,base_event_rate,pdo,train_bins_stat,stand_coef,var_describe_file_path,writer,sheet_name=lan['Score card'],filePath=None)

Enter the scorecard into Excel, including:

The score of each variable in each bin, the parameters for calibrating the scorecard score, and the variable weights

Parameters

Dcard:

ScoreCard.CardFlow.card

base_points : float

Benchmark score

base_event_rate : float

Benchmark event rate corresponding to the benchmark score

pdo : float

PDO

stand_coef : TYPE

The return value of Reg_Step_Wise_MP.LogisticReg.fit() is clf_coef['Standardized Estimate']

writer : writer, optional

A writer for Excel. If you want to combine the current information with other information and output them in the same Excel file, it is more convenient to use the writer. Otherwise, it is more convenient to use filePath.

sheet_name : str, optional

Write the y statistical results to the sheet page

filePath : str, optional

Writer is used first. If writer is None, the Excel file location specified by filePath is used.

Both writer and filePath cannot be None at the same time

Returns

None.

bfy_df_like_excel

bfy_df_like_excel(df_grid,writer,sheet_name='Sheet1',default_decimal=None,text_lum=0.5,red_max=True,row_num=0,col_num=0,row_gap=2,col_gap=2)

Export all tables to an Excel sheet

example:

```
df_grid=[]

df_rows1=[]#Put the table output to row 1 into this list

df_rows2=[]#Put the table output to row 2 into this list

df_rows3=[]#Put the table output to row 3 into this list

df_rows4=[]# Output the table in row 4 and put it into this list

#Note: The row here is not the concept of "row" in Excel

df_grid.append(df_rows1)

df_grid.append(df_rows2)

df_grid.append(df_rows3)

df_grid.append(df_rows4)


df1 = pd.DataFrame(np.random.randn(10, 4), columns=['A1', 'B1', 'C1', 'D1'])

df1['SCORE_BIN']=['[0,100)','[100,200)','[200,300)','[300,400)','[40 0,500)','[500,600)','[600,700)','[700,800)','[800,900)','[900,1000]']

# Output df1 to the first table in the first row

df_rows1.append({'df':df1,'title':['DF1','notAnum'],'percent_cols':df1.columns,'color_gradient_sep':True})


df2 = pd.DataFrame(np.random.randn(1, 4), columns=['A2', 'B2', 'C2', 'D2'])

# Output df2 to the second table in row 1

df_rows1.append({'df':df2,'color_gradient_cols':['A2','D2'],'title':['percent_BC','gradient_AD'],'percent_cols':['B2','C2']})
```

```
df3 = pd.DataFrame(np.random.randn(15, 4), columns=['A3', 'B3', 'C3', 'D3'])
# Output df3 to the first table in the second row
df_rows2.append({'df':df3,'color_gradient_cols':['B3'],'title':['long table']})

df4 = pd.DataFrame(np.random.randn(4, 4), columns=['A4', 'B4', 'C4', 'D4'])
# Output df4 to the second table in the second row
df_rows2.append({'df':df4,'color_gradient_sep':False})

df5 = pd.DataFrame(np.random.randn(10, 5), columns=['A5', 'B5', 'C5', 'D5', 'E5'])
# Output df5 to the first table in the third row
df_rows3.append({'df':df5,'color_gradient_sep':True,'not_color_gradient_cols':['C5']})

df6 = pd.DataFrame({'A6':[1,2,3,4],'B6':[0.1,1.2,100.5,7.4]})
# Output df6 to the first table in row 4
df_rows4.append({'df':df6,'color_gradient_sep':True,'percent_cols':['A6']})

#Two output methods, choose one according to actual needs
r,c = bfy_df_like_excel(df_grid,'Demo.xlsx',sheet_name='demo',red_max=False,row_num=4,col_num=4,row_gap
=2,col_gap=2)
or
with pd.ExcelWriter('Demo.xlsx') as writer:
r,c = bfy_df_like_excel(df_grid,writer,sheet_name='demo',red_max=False,row_num=4,col_num=4,row_gap=2,col
_gap=2)
print(r,c)

Parameters
----------
df_grid: a 2-dimensional list
[
[dict,dict,...], #line 1
```

[dict,dict,...],#Line 2

...

]

Each dict represents the settings of a DataFrame (table)

The key meaning of dict:

df: DataFrame to be output

title:See bfy_df_like_excel_one.df

percent_cols: see bfy_df_like_excel_one.percent_cols

color_gradient_cols: see bfy_df_like_excel_one.color_gradient_cols

not_color_gradient_cols: see bfy_df_like_excel_one.not_color_gradient_cols

color_gradient_sep: see bfy_df_like_excel_one.color_gradient_sep

decimal: See bfy_df_like_excel_one.decimal. If decimal is not set for the df, default_decimal is used by d efault

writer : str or pandas.ExcelWriter

Specify the excel path, or an already constructed ExcelWriter.

sheet_name : str, optional

The specified Excel sheet.

Default 'Sheet1'

default_decimal: int, optional

The default number of decimal places. If decimal is not set for a df, default_decimal is used by default.

Default: None

text_lum : float, optional

A value between [0,1] that controls the color difference between the text and the color scale. A larger value for text_lum increases the color difference.

When text_lum=0, the text is always black

Default 0.5

red_max : bool , optional

True: The larger the value, the redder the color, and the smaller the value, the greener the color.

False: The larger the value, the greener the color, and the smaller the value, the redder the color.

Defaults to True.

row_num : int, optional

The starting row number of the table

Default is 0.

col_num : int, optional

The starting column number of the table

Default is 0.

row_gap : int, optional

The spacing between rows.

Default is 2.

col_gap : int, optional

The spacing between columns.

Default is 2.

Returns

----

int

The last row after all tables are output.

int

The last column after all tables are output. (Not the largest column in the last row)

For example, if there are two rows, the maximum column of the first row is 5, and the maximum colu mn of the second row is 4, then 5 is returned.

bfy_df_like_excel_one

bfy_df_like_excel_one(df,writer,sheet_name='Sheet1',title=[],decimal=None,percent_cols=[],color_gradient_cols=None,not_color_gradient_cols=[],color_gradient_sep=True,text_lum=0.5,red_max=True,row_num=0,col_num=0)

Beautify a dataframe into an Excel pivot table-style chart and output it to the specified Excel.

You can do Excel color scale filling and percentage display (not formatted as percentage, but displayed as percentage, the actual value and value type remain unchanged).

example:

df1 = pd.DataFrame(np.random.randn(10, 4), columns=['A1', 'B1', 'C1', 'D1'])

df1['SCORE_BIN']=['[0,100)','[100,200)','[200,300)','[300,400)','[40 0,500)','[500,600)','[600,700)','[700,800)','[800,900)','[900,1000]']

r,c = bfy_df_like_excel_one(df1,'df1.xlsx',title=['DF1','any'],percent_cols=df1.columns,color_gradient_sep=True,text_lum=0,row_num=2,col_num=2)#

print(r,c)

Parameters

----

df : DataFrame

Dataframe that needs to be beautified

writer : str or pandas.ExcelWriter

Specify the excel path, or an already constructed ExcelWriter.

sheet_name : str, optional

The specified Excel sheet.

Default 'Sheet1'

title : list, optional

Each text in the title will be output above the table, and each element will occupy a cell

The default is [].

decimal : int , optional

The number of decimal places retained. Automatically exclude int type columns and non-numeric columns in df

The original data stored in Excel will not be changed, only the display of Excel will be changed. The function is the same as "Format Cells - Keep Decimals" in Excel.

None: Do not adjust the number of decimal places

Default: None

percent_cols: list, optional

Columns that need to be displayed as percentages automatically retain 2 decimal places of percentages

For example: 0.23456 -> 23.46%

The original data stored in Excel will not be changed, only the display of Excel will be changed. The function is the same as "Format Cells - Percentage" in Excel.

Int type columns and non-numeric columns will be automatically excluded

default[]

color_gradient_cols : list, optional

Columns that need to display color scale

None: All numeric columns in df need to display color scales

Non-numeric columns will be automatically excluded

Default is None.

not_color_gradient_cols : list, optional

Columns that do not need to display color levels. not_color_gradient_cols has a higher priority than color_gradient_cols

Non-numeric columns are automatically excluded, so there is no need to mark them here.

default[].

color_gradient_sep: bool , optional

True: The color of each cell is determined by the maximum and minimum values in its column

False: The color of each cell will be determined by the maximum and minimum values of all participating color scale columns.

Default True

text_lum : float, optional

A value between [0,1] that controls the color difference between the text and the color scale. A larger value for text_lum increases the color difference.

When text_lum=0, the text is always black

Default 0.5

red_max : bool , optional

True: The larger the value, the redder the color, and the smaller the value, the greener the color.

False: The larger the value, the greener the color, and the smaller the value, the redder the color.

Defaults to True.

row_num : int, optional

The starting row number of the table

Default is 0.

col_num : int, optional

The starting column number of the table

Default is 0.

Returns

----

int

The row_num at the end (the row number in the lower right corner of the table)

int

col_num at the end (the column number in the lower right corner of the table)

Sampling

Functions

split_cls

split_cls(dat,y='y',test_size=0.3,w=None,groups=[],random_state=0)

High-precision high-dimensional stratified sampling. The stratified sampling built into machine learning is overly simplistic. When stratified sampling is performed solely on the Y label, a phenomenon may occur: after the variables in the training and test sets are segmented with equal frequency according to the same nodes, the event rates of Y differ significantly, making it difficult to narrow the indicator differences between the training and validation sets during modeling. Without high-precision high-dimensional stratified sampling, this problem can only be addressed by reducing model performance to increase model generalization. Another phenomenon is that after binning, the binning effects of the training and validation sets differ significantly, reflected in significantly different IV values. Without high-precision high-dimensional stratified sampling, the only way to increase generalization is to increase the bin width. The stratified sampling method provided by rascpy has been tested and found to significantly alleviate this phenomenon, reducing the difference between the training and validation sets without reducing model performance or binning IV. (Excessive differences between datasets are often caused by inconsistent high-dimensional joint distributions, but due to sampling accuracy limitations, they can only be treated as overfitting.)

One of the most important evaluation criteria for the effectiveness of a high-dimensional stratified sampling algorithm is whether the joint distribution of each x variable and y can remain consistent in each data set after sampling.

If the data itself can be divided into multiple groups from different dimensions, then it is also required that the joint distribution of each x variable and y can be consistent in each group in each data set after sampling.

rascpy provides the rascpy.Sampling.split_cls algorithm, which is designed for high-precision sampling of binary classification problems. Compared with multiple sampling algorithms, it shows good consistency in joint distribution, regardless of whether the data contains groups. This is especially true for x variables, which have good predictive power.

Parameters

----
dat : DataFrame
Data to be sampled

y : str, optional
target, performs stratified sampling based on the column names specified by y. After sampling, the event rates of y remain roughly consistent across all datasets. The joint distribution of each x and y is also kept roughly consistent across all datasets.
Default: 'y'.

test_size : float, optional
The scale of the dataset
Default: 0.3.

w : str, optional
The column name corresponding to the sample weights. If w is not empty, stratified sampling is performed according to the column corresponding to y, so that the weighted event rate of y remains roughly consistent across all datasets. The weighted joint distribution of each x and y is also kept roughly consistent across all datasets.
Default: None, meaning all samples have equal weights.

groups : list, optional
The elements in groups are the column names corresponding to the data groups. If the number of elements in groups is 2 or more, a cross-group is formed. Each sample point belongs to a group (only 1 group) or a cross-group (2 or more groups). When the number of elements in groups is greater than 0, stricter requirements are imposed on stratified sampling, that is, the following requirements must still be met within each group or cross-group:
1. Meet the test_size ratio requirements
2. The event rate or weighted event rate that satisfies y should be roughly consistent
3. The joint distribution of each x and y should be kept roughly consistent across all datasets.

default:[].

random_state: float, optional
Random Seed
Default: 0.

Returns

DataFrame
train dataset.

DataFrame
test dataset.

ScoreCard

Classes

CardFlow

It divides the scorecard development process into 10 phases: data reading, equal-frequency binning, feature pre-screening, monotonic and U-shaped recommendations, rasc optimal binning, Word of Equinox conversion, feature screening, model building, scorecard creation, and report generation. Results are automatically saved after each phase, allowing you to resume work from any breakpoint, and restarting the computer will not lose the results of completed 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 filtering conditions, which affects the results of steps 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.
For most users, after configuring the configuration file, CardFlow is the only component you need to interact with. The interaction method is very simple: cardflow.start(start_step=a,end_step=b)
CardFlow adds an extra stage to the 10 stages: rejection inference. If the user specifies the data location for rejection inference, you can set end_step to 11 to enable training of the rejection inference model.

```
from rascpy.ScoreCard import CardFlow
if __name__ == '__main__': # Windows must write a main function (but not in jupyter), Linux and MacOS do not need to write a main function
# Pass in the command file
scf = CardFlow('./inst.txt')
scf.start(start_step=1,end_step=11)# will automatically give the score card + the score card for rejection inference

# You can stop at any step, as follows:
scf.start(start_step=1,end_step=10)#No scorecard will be developed for rejection inference
scf.start(start_step=1,end_step=9)#No model report will be output

# If the results of the run have not been modified, there is no need to run again. As shown below, steps 1-4 that have been run will be automatically loaded (will not be affected by restarting the computer)
scf.start(start_step=5,end_step=8)

# You can also omit start_step and end_step, abbreviated as:
scf.start(1,10)
```

__init__

__init__(self, config_file=None, encoding='utf-8', **cover_conf)

Build the CardFlow instance. If successful, the configuration items will be saved in the specified ${work_space}/0_conf.pkl.

Parameters

----

inst_file : str

Specify the location of the directive file

None: Do not use the command file

Default: None

encoding : str

Configuration file encoding

Default: 'utf-8'

**cover_inst : dict

The configuration in cover_inst will supplement and replace the configuration in the inst_file file.

do_load_datas

do_load_datas(self)

The first step in the rasc modeling process. It divides the data into five directories: model_data, psi_data, oot_data, performance_data, and reject_model_data. These directories correspond to the file directories specified in the configuration: model_data_file_path, psi_data_file_path, oot_data_file_path, performance_data_file_path, and reject_data_file_path. (Only model_data_file_path is required.) do_load_datas reads all files in these five directories (directories not specified are automatically ignored) and constructs a data set structure with nested dicts.

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

Data uses include:

model_data: data used for modeling, which can be classified into training set, validation set and test set.

psi_data: used to store data for checking variable stability, 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 model's performance on the dataset can be included in this data.

reject_model_data: data used to train the rejection inference model

After the data is read, the nested dict dataset structure is saved in the specified ${work_space}/1_datas.pkl file. It can also be obtained through the CardFlow instance scf.datas.

Data categories are only used to help users remember and are not used to calculate metrics. For example, the model effects of the data in oot_data will also be displayed. The data in model_data can also be used to calculate the PSI metric to filter variables .

do_freq_bins

do_freq_bins(self)

The second step in the rasc modeling process uses a dataset whose data usage is model_data and is set to train by the user to perform node calculations for equal-frequency binning, with special values and null values in separate bins. It then splits all data from all data usages according to the calculated equal-frequency binning nodes, and finally calculates information for each bin, including:

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

The equal-frequency binning nodes of each variable in each data set remain consistent, and other indicators are calculated based on the variables themselves.

After do_freq_bins successfully runs, it generates two results, which can be obtained through the CardFlow instances scf.train_freqbins and scf.freqbins_stat, respectively. The first is the binning nodes calculated based on the training set. The second is the statistics of all data after being segmented by the binning nodes of the first result. scf.freqbins_stat is a two-level nested dict structure: dict<data purpose, dict<data name, equal frequency binning statistics>>.

Combine the two results into a tuple and save it in ${work_space}/2_freqbins.pkl. For user convenience, the second result will also be written to ${work_space}/2_freqbins.xlsx. Each sheet contains the equal frequency bin statistics of a data set. The naming convention of the sheet is: 'Data set file name (data purpose)' See Bins.FreqBin , Bins._x1FreqBin

do_fore_filter

do_fore_filter(self)

The third step in the rasc modeling process is to pre-filter the variables.

Because equal-frequency binning is unconstrained, its IV value must be greater than or equal to the IV value of the optimal binning. Therefore, when the IV value of the equal-frequency binning is below the threshold specified by small_iv, these variables can be filtered out and excluded from the subsequent optimal binning (because the IV value of the optimal binning is always below the threshold). When there are many variables and many of them are below the threshold specified by small_iv, this feature can be enabled to reduce the number of variables and thus shorten the time it takes to run the optimal binning.

Because equal-frequency binning usually produces more bins than optimal binning, the big_homogeneity of equal-frequency binning is smaller than the big_homogeneity of optimal binning. Therefore, when the big_homogeneity of equal-frequency binning is greater than the threshold, these variables can be filtered out first to reduce the running time of subsequent steps.

Missing values will not change whether it is equal-frequency binning or optimal binning, so when the big_miss of equal-frequency binning is greater than the threshold, these variables can be filtered out first to reduce the running time of subsequent steps.

When do_fore_filter runs successfully, it generates two results, which can be called through the CardFlow instances scf.fore_col_indices and scf.fore_filtered_cols respectively. The first is the filter index value calculated for each variable based on the distribution of equal frequency bins, and the second is the variable that was filtered out and the reason for filtering 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, wrapped in newlines)

[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 description.

do_mono_suggest

do_mono_suggest(self)

The fourth step in the rasc modeling process. If mono_suggest=True is configured, rasc will use all datasets 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 (the suggestion is a tuple, only the first element is useful, and the remaining elements are for backward compatibility extensions). 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 be combined into a tuple and saved in ${work_space}/4_mono_suggest.pkl. For user convenience, the second result will also be written to ${work_space}/4_mono_suggest.xlsx.

See Bins.MonoSuggest , Bins._x1MonoSuggest

do_optim_bins

do_optim_bins(self)

The fifth step in the rasc modeling process uses the dataset specified as train in the data usage model_data to calculate the optimal binning nodes for rasc (see the introduction to the Bins module for more information on optimal binning for rasc). It then divides all data in all data usages according to the calculated optimal binning nodes, and finally summarizes the information for each bin, 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.

The rasc optimal binning node for each variable in each data set remains consistent, and other indicators are calculated based on the variable 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 optimal binning node calculated based on the training set, and the second is the statistical information of each box after all data are divided according to the optimal binning node. scf.optbins_stat is a two-level dict nested structure: dict<data purpose, dict<data name, 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](), [Bins._x1OptBin]()

do_woe

do_woe(self)

Step 6 of the rasc modeling process. Perform a WOE conversion on all data for all data purposes, based on the user-specified mapping between training set bins and WOE. The converted results can be retrieved via 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)

Step 7 of the rasc module process. Variables are filtered according to the user's settings. When do_filter runs successfully, it generates four results: the first is the filter index value calculated for each variable in each dataset, and the second is the filtered variable and the reason for filtering. The format of the filtered reason record is as follows:

filter_metric > or < filter_threshold [dataset_name]

...(records deleted by multiple filters, wrapped in newlines)

Filter metrics: iv, homogeneity, miss, ivCoV, corr, psi, user-extended

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

The third column contains intermediate data from the indicator calculation process. This intermediate data can be output to the model report to help users better understand the data. The fourth column records variables that the user has forced to retain and set. For the difference between retaining and setting, see the description of the user_save and user_set configuration items in the command file.

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 command file.

do_model

do_model(self)

The eighth step of the rasc modeling process is implemented by calling the built-in two-way stepwise logistic regression function of rasc (see the Reg_Step_Wise_MP module).

After do_model runs successfully, 7 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 outputs the predicted probability

The main properties are:

intercept_ intercept term

coef_[0] coefficients of each variable (excluding the intercept term)

tvalues is 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 seven results are combined into a tuple and saved in ${work_space}/8_model.pkl. For easier reading, the results of steps 4, 5, 6, and 7 are also written to ${work_space}/8_reg_mstep.xlsx.

See Reg_Step_Wise_MP.LinearReg , Reg_Step_Wise_MP.LogisticReg

do_card

do_card(self)

The 9th step of the rasc modeling process. A scorecard is constructed based on the generated model and WOE. It can be accessed through the CardFlow instance rasc.card and saved in ${work_space}/9_card.pkl.

do_report

do_report(self)

10th step in the rasc modeling process . After the run is complete, a model report file ${work_space}/{model_name}_Report.xlsx will be generated. The file contents are:

1. Sample Y statistics: Lists 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, making it easier for users to 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. Variable selection intermediate table: When calculating variable selection, some useful intermediate res

ults 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 scores of output variables, scorecard conversion parameters, variable weights of standardized coefficient caliber, and variable weights of range caliber

6. Model Performance: List the model metrics (including KS, AUC, LIFTn) of all target-labeled datasets for each data usage, and a pivot table of scores and frequency segments (including number, cumulative number, distribution, cumulative distribution, event rate, cumulative event rate, ODDS, cumulative ODDS, etc.)

7. Logistic regression model: The content is the same as ${work_space}/8_reg_mstep.xlsx generated in step 8. See the API documentation of [CardFlow.do_model]

8. Correlation coefficient: List the correlation coefficients between each pair of model variables

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]

11. rasc 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]

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 datasets under each data use. It can be accessed through the CardFlow instance rasc.preds and saved in ${work_space}/10_preds.pkl.

do_reject

The 11th step of the rasc modeling process. If the user sets reject_data_file_path and end_step=ScoreCard.REJECT_STEP/11, CardFlow will generate a rejection inference scorecard model.
After the rejection inference scorecard is generated, it will produce the same information as the normal scorecard.
The calling method is similar to the normal scorecard, except that the prefix rejInfer is added.
scf.rejInfer.train_freqbins,scf.rejInfer.freqbins_stat,
scf.rejInfer.fore_col_indices,scf.rejInfer.fore_filtered_cols,
scf.rejInfer.mono_suggests,scf.rejInfer.mono_suggests_eventproba,
scf.rejInfer.train_optbins,scf.rejInfer.optbins_stat,scf.rejInfer.woes,
scf.rejInfer.col_indices,scf.rejInfer.filtered_cols,scf.rejInfer.filters_middle_data,scf.rejInfer.used_cols
scf.rejInfer.in_clf_cols,scf.rejInfer.clf_del_cols,scf.rejInfer.clf,scf.rejInfer.clf_perf,scf.rejInfer.clf_coef,scf.rejInfer.del_r

eason,scf.rejInfer.step_proc
scf.rejInfer.card,scf.rejInfer.preds
And store the newly synthesized dataset for rejection inference in scf.datas['rejData']['__synData']

redo_bins

redo_bins(self)

If the user needs to update the equal frequency binning nodes or optimal binning nodes of individual v ariables after running do_freq_bins or do_optim_bins, they need to call scf.redo_bins(). This method will recalculate the equal frequency or optimal binning for the variables in [MODIFY BINS INST]:redo_bins_c ols in the instruction file according to the current instructions in [BINS INST], and update and save the mono_suggest, Bins, and WOE of the variables involved. The update principle is:

1. If equal-frequency binning is set in redo_type, redo_bins modifies the equal-frequency binning nodes of the variables in redo_bins_cols, incrementally updates the two results generated in step ${do_freq_bin s}, and updates the files saved by ${do_freq_bins} in ${work_space}. If the user has already run step 4, ${do_mono_suggest}, redo_bins modifies the monotonicity suggestions for the variables in redo_bins_col s, incrementally updates the results generated in step ${do_mono_suggest}, and updates the files saved by ${do_mono_suggest} in ${work_space}. Finally, a new file, ${work_space}/redo_freqbins_Compare.xlsx, i s generated to compare the changes before and after the equal-frequency binning of the specified varia bles in all datasets for each data use.
2. If optimal binning is set in redo_type, redo_bins modifies the optimal binning nodes of the variables in redo_bins_cols, incrementally updates the two results generated in step ${do_optim_bins}, and update s the file saved by ${do_optim_bins} in ${work_space}. If the user has already run step 6 ${do_woe}, th e Word of Equivalence (WOE) values of the variables in redo_bins_cols are modified, the results generat ed in step ${do_woe} are incrementally updated, and the file saved by ${do_woe} in ${work_space} is u pdated. A new file ${work_space}/redo_optbins_Compare.xlsx is also generated to compare the optimal b inning changes before and after the change for the specified variable rasc in all datasets for each data use.

dat_update

dat_update(self, dat_uses,only_dat=False)

Update all data sets under the specified data usage, update the result scf.datas generated by step 1${do _load_datas}, and update '1_datas.pkl' under ${work_space}.

If step 2${do_freq_bins} has been completed, all data sets under the specified data purpose in scf.freqbins_stat will be updated, and the '2_freqbins.pkl' and '2_freqbins.xlsx' saved in step 2${do_freq_bins} under ${work_space} will be updated.

If step 5${do_optim_bins} has been completed before, all data sets under the specified data purpose, namely scf.optbins_stat, will be updated, and step 5${do_optim_bins} will be saved in '5_optbins.pkl' and '5_optbins.xlsx' under ${work_space} for update.

If step 6${do_woe} has been completed before, all datasets under the specified data purpose, namely scf.woes, will be updated, and step 6${do_woe} will be saved in '6_woe.pkl' under ${work_space} for update.

Note: dat_update only changes the statistics 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 dataset and also modify the binning nodes, namely scf.train_freqbins and scf.train_optbins, you need to re-execute the already run process, 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','REJECT_MODEL_DATA']

only_dat:boolean

True: only update data

False: In addition to updating the data, the statistics of the equal frequency and optimal binning of the dataset will also be updated (but the binning nodes and woe

If you need to update the data node, please use it in conjunction with redo_bins

start

start(self, start_step=1, end_step=10, load_step=None)

Call start to start the modeling process. The code corresponding to each step is:

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

REJECT_STEP=11 (reject inference step. Reject inference will continue to build a rejection inference model based on the normal model. If the user sets reject_data_file_path, after the first 10 steps are complet ed, running step 11 will generate a rejection inference model for the user)

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 f rom 1. For example, if you run scf.start(start_step=1,end_step=4), you can continue to run scf.start(start_ step=5,end_step=8) even after restarting the computer.

Parameters

----

start_step : int

Which step to start running

Default: 1

end_step : int

At which step does the operation end?

Default: 10.

load_step : float

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

If load_step is not empty, the settings of start_step and end_step will be automatically ignored.

Advanced users may use rasc semi-automatically, in which case the load_step parameter is useful.

Default: None.

Functions

get_row_score

get_row_score(card,row,err_handle=np.min)
Given a piece of data, calculate the score of the data based on the card
Parameters
----
card : dict
Comparison table of bin labels and scores

row : dict
The value of each variable
ex. {'x1':1,'x2':2}

err_handle : TYPE, optional
This method is used to calculate scores when the values of the variable cannot match the bins.
The default is np.min.

Returns
----
score : float
The score of the row .

get_X_score

get_X_score(card, X, cores=None)

Returns the subscores and total score of X 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 rasc .card.

X : DataFrame

Data to be scored

cores : int

Number of CPU cores used

None: All CPUs

err_handle : TYPE, optional

This method is used to calculate scores when the values of the variable cannot match the bins.

The default is np.min.

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

make_card

make_card(base_points,base_event_rate,pdo,clf,bins_stat)

do not use rasc automated modeling. After users use Reg_Step_Wise_MP to build a model and use the Bins module to build bins, they can use this method to generate a scorecard.

Parameters

----

base_points : float

Benchmark score corresponding to the benchmark probability

base_event_rate : float

Baseline probability

pdo : float

PDO

clf : Reg_Step_Wise_MP.LogisticReg

rasc built- in logistic regression model

bins_stat : dict

rasc built- in binning module and binning information

Returns

----

card : dict

Comparison table of bins and scores for each variable

StepwiseRegressionSKLearn

New since version 2025.10.2

from version== 2025.10.2 , it replaces Reg_Step_Wise_MP as the built-in two-way stepwise regression model in rascpy.

Compared to Reg_Step_Wise_MP, this significantly reduces computation time. In Reg_Step_Wise_MP, a single algorithmic step might slightly improve the model with a very small probability, but this would require a significant amount of computation time. This step is omitted in StepwiseRegressionSKLearn, thus reducing computation time.

It is a Python implementation of linear bidirectional stepwise regression and logistic bidirectional stepwis e regression, which adds the following features to traditional bidirectional stepwise regression:

Functions retained from Reg_Step_Wise_MP are:

1. When performing stepwise variable selection for logistic regression, AUC, KS, or LIFT can be used instead of AIC and BIC. For some business scenarios, AUC and KS are more relevant. For example, in ranking tasks, models built using the KS metric have the advantage of using fewer variables while maintaining a consistent KS performance across multiple test sets.

2. During stepwise variable selection, you can use other datasets to calculate model evaluation metrics instead of the modeling dataset. Especially when the data volume is large and a validation set is included in addition to the training and test sets, it is recommended to use the validation set to calculate evaluation metrics to guide variable selection. This helps reduce overfitting.

3. Supports using partial data to calculate model evaluation metrics to guide variable selection. For example, if the business needs to maintain a certain pass rate of N%, then the bad event rate of the top N% of samples can be minimized, without requiring all samples to be included in the calculation. Based on past experience, in appropriate scenarios, using partial data as evaluation metrics results in fewer variables than using full data, but the metrics of interest to users do not decrease across multiple test sets. Because the model focuses only on the top, more easily distinguishable sample points, it can achieve business goals without requiring too many variables.

4. Supports setting multiple conditions. Variables must meet all conditions simultaneously before they can be included in the model. This allows variable selection and model diagnosis to be performed simultaneously, avoiding repeated modeling due to model diagnosis failure. Built-in conditions include: P-Value, VIF, correlation coefficient, and coefficient sign.

5. Supports specifying variables that must be entered into the model. If the specified variables conflict with the conditions in point 4, a comprehensive mechanism has been designed to resolve the problem.

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

New features added in version== 2025.10.2 are:

7. Support users to specify the number of variables that can be included in each variable group

8. Added sklearn interface, which can be used in sklearn's pipline

In most cases, users do not need to interact directly with the Reg_Step_Wise_MP component. However, rascpy is designed to be pluggable, so advanced users can use the Reg_Step_Wise_MP module indepen dently, just like any other Python module.

Classes

LinearReg

from version== 2025.10.2 , Reg_Step_Wise_MP.LinearReg has been replaced as the built-in bidirectional stepwise linear regression model in rascpy. Reg_Step_Wise_MP is still retained and can still be used, but it is recommended to use StepwiseRegressionSKLearn.LinearReg because the calculation time is greatly reduced.
Functions retained from Reg_Step_Wise_MP:
1. During stepwise variable selection, you can use other datasets to calculate model evaluation metrics instead of the modeling dataset. Especially when the data volume is large and a validation set is included in addition to the training and test sets, it is recommended to use the validation set to calculate evaluation metrics to guide variable selection. This helps reduce overfitting.
2. Supports setting multiple conditions. Variables must meet all conditions simultaneously before they can be included in the model. This allows for simultaneous variable selection and model diagnosis, avoiding repeated modeling due to model diagnosis failure. Built-in conditions include: P-Value, VIF, correlation coefficient, and coefficient sign.
3. Supports specifying variables that must be entered into the model. If the specified variables conflict with the conditions in point 4, a comprehensive mechanism has been designed to resolve the problem.
4. The modeling process is exported to EXCEL, recording the reasons for deleting each variable and the process information of each round of stepwise regression.

New since version 2025.10.2
5. Support users to specify the number of variables that can be entered into the model in each variable group
6. Added sklearn interface, which can be used in sklearn's pipline

__init__

```
__init__(self,user_save_cols=[],user_set_cols=[],measure='r2',measure_frac=1
        ,pvalue_max=0.05,vif_max=3,corr_max=0.7,coef_sign={},default_coef_sign=None
              ,X_group_format=None,cnt_in_group={},default_cnt_in_group=None,fea_cnt=15,results_save=None)
```

Parameters

----

user_save_cols : array like, optional

Forced variables into the module

default:[]

user_set_cols : array like, optional

Only these variables can be entered into the model, without addition or deletion.

If user_set_cols is not empty and has length greater than 0, stepwise regression degenerates to ordinary regression.

default:[]

measure : str, optional

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

Indicators include: r2

Default: 'r2'.

measure_frac : float, optional

Sort by predicted value from large to small or small to large, and take the first N sample points from ${MODEL CONFIG:measure_data_name} as the evaluation index of the model

None: Take all sample points from ${MODEL CONFIG:measure_data_name} to calculate the model evaluation index. Equivalent to measure_frac=1.

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 largest to smallest (round down)

-1 <= measure_frac < 0: Take the first N = sample_n*measure_frac*-1 sample points from smallest to largest (round down)

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

Default: 1

pvalue_max : float, optional

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 w ill 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, the introduction of the non-mandatory variable will not be affected.

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

Default: 0.05

vif_max : float, optional

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 included in 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 included in 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, the introduction of the non-mandatory variable will not be affected.

None: No restrictions are placed on the vif of the input variables

Default: 3

corr_max : float, optional

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

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

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

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

Default: 0.7

coef_sign : dict, optional

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 has a negative sign

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 entered into the model are not subject to this constraint

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

default: {}

default_coef_sign : str, optional

When a variable is not in coef_sign, the default value of the variable symbol constraint is

None: The default value of all variables is None

Default: None

X_group_format : str, optional

New in version 2025.10.2

How to express groups in X variables

In business, X can sometimes be categorized and managed by groups. For example, all data provided by data service company A can be divided into one group, and all variables provided by data service company B can be divided into another group.

ex1. X_group_format = _g

ex2. X_group_format = g$$

Value Description:

A character must be g and can only appear at the beginning or end. If g appears at the beginning, it means the group name is the prefix of the variable name. If g appears at the end, it means the group name is the suffix of the variable name. The situation where the group name is in the middle of the variable name cannot be handled.

The remaining characters are group separators, which are strings that do not contain the letter g. This separator is used to separate the group name from the variable name.

For example: If the variable name format is x1_group1, then this should be configured as _g

If the variable naming format is group1##x1, then this should be configured as g##

If a variable name does not contain the configured separator, it means that the variable is not in any group, and subsequent instructions for operating variables by group will not be applied to this variable.

None: All X do not need to be grouped

Default: None

cnt_in_group : dict, optional

Set the maximum number of variables allowed in each variable group

Example: {"g1":1,"g2":2}

Default: {} means no group will be restricted by the number of variables entered into the model

default_cnt_in_group : int, optional

If a variable group is not set in cnt_group, the maximum number of variables allowed to be entered into the module is

None: There is no default maximum limit on the number of variables that can be entered into the model. If the variable group does not appear in cnt_in_group, there is no limit on the number of variables that can be entered into the model within the group.

Default: None

fea_cnt : int, optional

The number of variables entered into the model

Each stepwise regression has two operations:

1. Find a variable from all remaining variables that meets the constraints and whose addition will improve the model index by the highest amount. 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 more than the current one, and the one with the highest improvement. Remove this variable from the model.

If N variables have been entered into the model (N < fea_cnt), and adding or removing variables canno t further improve the performance of the model, the stepwise regression is terminated early.

Default: 15

results_save : str, optional

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

None: Do not record the process

Default: None.

Returns

None.

fit

fit (self,X,y,sample_weight=None,fit_args={},X_val=None,y_val=None,sample_weight_val=None,val_args={})

Training the model

Parameters

X : pandas.DataFrame

Feature dataset used for model training

y : pandas.Series

y labels for model training

sample_weight : pandas.Series, optional

Sample weights used for training

None: Each sample has the same weight

Default: None.

fit_args : dict, optional

Other parameters used for model training

default:{}

X_val : pandas.DataFrame, optional

Feature dataset for validation

Default: None.

y_val : pandas.Series, optional

y labels for validation

Default: None.

sample_weight_val : pandas.Series, optional

Sample weights for validation

None: Each sample has the same weight

Default: None.

val_args : dict, optional

Other parameters used for model evaluation

default:{}

Returns

The trained StepwiseRegressionSKLearn.LinearReg instance (self)

Attributes

After calling fit, the StepwiseRegressionSKLearn.LinearReg instance generates the following Attributes

----

in_vars : list

Model variables

intercept_ : float

Intercept term

coef_ : pandas.Series

Coefficients of each variable (excluding the intercept term)

estimator_ : statsmodels.regression.linear_model.RegressionResultsWrapper

When deploying online applications, if the rascpy library is not installed in Python, but statsmodels is installed , users can directly use estimator_ to deploy the model.

perf : DataFrame

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

coef : DataFrame

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

del_reason s : pandas. Series

The reason for deletion of each deleted variable

step_proc : DataFrame

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

predict

predict(self,X)

Returns the predicted value for each sample point

Parameters

X : pd.DataFrame

Feature dataset to be predicted

Returns

----

hat: array, shape (n_samples,)

Predicted value

LogisticReg

from version== 2025.10.2 , Reg_Step_Wise_MP.LogisticReg has been replaced as the built-in two-way stepwise logistic regression model in rascpy. Reg_Step_Wise_MP is still retained and can still be used, but it is recommended to use StepwiseRegressionSKLearn.LogisticReg because the calculation time is greatly reduced.
Functions retained from Reg_Step_Wise_MP:
1. When performing stepwise variable selection for logistic regression, AUC, KS, or LIFT can be used instead of AIC and BIC. For some business scenarios, AUC and KS are more relevant. For example, in ranking tasks, models built using the KS metric have the advantage of using fewer variables while maintaining a consistent KS performance across multiple test sets.
2. During stepwise variable selection, you can use other datasets to calculate model evaluation metrics instead of the modeling dataset. Especially when the data volume is large and a validation set is included in addition to the training and test sets, it is recommended to use the validation set to calculate evaluation metrics to guide variable selection. This helps reduce overfitting.
3. Supports using partial data to calculate model evaluation metrics to guide variable selection. For example, if the business needs to maintain a certain pass rate of N%, then the bad event rate of the top N% of samples can be minimized, without requiring all samples to be included in the calculation. Based on past experience, in appropriate scenarios, using partial data as evaluation metrics results in fewer variables than using full data, but the metrics of interest to users do not decrease across multiple test sets. Because the model focuses only on the top, more easily distinguishable sample points, it can achieve business goals without requiring too many variables.
4. Supports setting multiple conditions. Variables must meet all conditions simultaneously before they can be

included in the model. This allows variable selection and model diagnosis to be performed simultaneously, avoiding repeated modeling due to model diagnosis failure. Built-in conditions include: P-Value, VIF, correlation coefficient, and coefficient sign.

5. Supports specifying variables that must be entered into the model. If the specified variables conflict with the conditions in point 4, a comprehensive mechanism has been designed to resolve the problem.

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

New since version 2025.10.2

7. Support users to specify the number of variables that can be entered into the model in each variable group

8. Added sklearn interface, which can be used in sklearn's pipline

__init__

```
__init__(self,y_label={'unevent':0,'event':1},user_save_cols=[],user_set_cols=[]
        ,measure='roc_auc',measure_frac=1,pvalue_max=0.05,vif_max=3,corr_max=0.7
        ,coef_sign={},default_coef_sign=None,X_group_format=None
        ,cnt_in_group={},default_cnt_in_group=None,fea_cnt=15,results_save=None)
```

Parameters

----

y_label : dict, optional

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, defining the things users care about most as events makes explanation easier. For example, if you want to emphasize the incidence of lung cancer, you can say that smokers have a 50% higher incidence of lung cancer than 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 "Smokers have a 50% lower incidence of lung cancer than non-smokers." Obviously, the first expression emphasizes the event you are concerned about.

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

user_save_cols : array like, optional

Forced variables into the module

default:[]

user_set_cols : array like, optional

Only these variables can be entered into the model, without addition or deletion.

If user_set_cols is not empty and has length greater than 0, stepwise regression degenerates to ordinary regression.

default:[]

measure : str, optional

In bidirectional stepwise regression, an indicator is used 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: 'roc_auc'.

measure_frac : float, optional
Sort the predicted event probability from large to small or small to large, and take the first N sample points from ${MODEL CONFIG:measure_data_name} as the model evaluation index
None: Take all sample points from ${MODEL CONFIG:measure_data_name} to calculate the model evaluation index. Equivalent to measure_frac=1.
If measure_index is aic or bic, the measure_frac configuration is ignored. Only the entire 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 largest to smallest (round down)
-1 <= measure_frac < 0: Take the first N = sample_n*measure_frac*-1 sample points from smallest to largest (round down)
measure_frac < -1: Take the first N = measure_frac*-1 sample points from small to large
Default: 1

pvalue_max : float, optional
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, the introduction of the non-mandatory variable will not be affected.
None: No constraints are placed on the p-value of the model variable.
Default: 0.05

vif_max : float, optional
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 included in 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 included in 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, the introduction of the non-mandatory variable will not be affected.
None: No restrictions are placed on the vif of the input variables
Default: 3

corr_max : float, optional
The correlation coefficients between all input variables must be less than or equal to the threshold
If the correlation coefficient of a non-forced variable with a forced variable exceeds the threshold, the non-forced

variable will not be introduced into the model.
Even if the correlation coefficient between two forced variables is above this threshold, both variables will be included.
None: No restriction on the correlation coefficient of the model variables
Default: 0.7

coef_sign : dict, optional
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 has a negative sign
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 entered into the model are not subject to this constraint
If the introduction of a non-mandatory variable causes a mandatory variable that originally satisfied the sign constraint to no longer satisfy the sign constraint, the non-mandatory variable cannot be included in the module. If the sign of the mandatory variable itself does not satisfy the sign constraint, the introduction of the non-mandatory variable will not be affected.
default: {}

default_coef_sign : str, optional
When a variable is not in coef_sign, the default value of the variable symbol constraint is
None: The default value of all variables is None
Default: None

X_group_format : str, optional
New in version 2025.10.2
How to express groups in X variables
In business, X can sometimes be categorized and managed by groups. For example, all data provided by data service company A can be divided into one group, and all variables provided by data service company B can be divided into another group.
ex1. X_group_format = _g
ex2. X_group_format = g$$
Value Description:
A character must be g and can only appear at the beginning or end. If g appears at the beginning, it means the group name is the prefix of the variable name. If g appears at the end, it means the group name is the suffix of the variable name. The situation where the group name is in the middle of the variable name cannot be handled.
The remaining characters are group separators, which are strings that do not contain the letter g. This separator is used to separate the group name from the variable name.
For example: If the variable name format is x1_group1, then this should be configured as _g
If the variable naming format is group1##x1, then this should be configured as g##
If a variable name does not contain the configured separator, it means that the variable is not in any group, and subsequent instructions for operating variables by group will not be applied to this variable.

None: All X do not need to be grouped
Default: None

cnt_in_group : dict, optional
Set the maximum number of variables allowed in each variable group
Example: {"g1":1,"g2":2}
Default: {} means no group will be restricted by the number of variables entered into the model

default_cnt_in_group : int, optional
If a variable group is not set in cnt_group, the maximum number of variables allowed to be entered into the module is
None: There is no default maximum limit on the number of variables that can be entered into the model. If the variable group does not appear in cnt_in_group, there is no limit on the number of variables that can be entered into the model within the group.
Default: None

fea_cnt : int, optional
The number of variables entered into the model
Each stepwise regression has two operations:
1. Find a variable from all remaining variables that meets the constraints and whose addition will improve the model index by the highest amount. 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 more than the current one, and the one with the highest improvement. Remove this variable from the model.
If N variables have been entered into the model (N < fea_cnt), and adding or removing variables cannot further improve the performance of the model, the stepwise regression is terminated early.
Default: 15

results_save : str, optional
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.

Returns

None.

fit

fit(self,X,y,sample_weight=None,fit_args={},X_val=None,y_val=None,sample_weight_val=None,val_args={})

Training the model

Parameters

X : pandas.DataFrame

Feature dataset used for model training

y : pandas.Series

y labels for model training

sample_weight : pandas.Series, optional

Sample weights used for training

None: Each sample has the same weight

Default: None.

fit_args : dict, optional

Other parameters used for model training

default:{}

X_val : pandas.DataFrame, optional

Feature dataset for validation

Default: None.

y_val : pandas.Series, optional

y labels for validation

Default: None.

sample_weight_val : pandas.Series, optional

Sample weights for validation

None: Each sample has the same weight

Default: None.

val_args : dict, optional

Other parameters used for model evaluation

default:{}

Returns

The trained StepwiseRegressionSKLearn.LogisticReg instance (self)

Attributes

After calling fit, the StepwiseRegressionSKLearn.LogisticReg instance generates the following Attributes

----

in_vars : list

Model variables

intercept_ : float

Intercept term

coef_ : pandas.Series

Coefficients of each variable (excluding the intercept term)

estimator_ : statsmodels.discrete.discrete_model.BinaryResultsWrapper

When deploying online applications, if the online Python does not have the rascpy library installed, but statsmodels is installed, users can directly use estimator_ to deploy the model.

perf : DataFrame

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

coef : DataFrame

],

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

del_reasons : pandas.Series
The reason for deletion of each deleted variable

step_proc : DataFrame
Detailed records of each round of modeling process, including: adding or removing variables, model performance indicators, etc.

predict_proba

predict_proba(self,X)

Returns the probability of each label category prediction. The order is the lexicographic order of the labels.

Parameters

X : pd.DataFrame
Feature dataset to be predicted

Returns
----
proba_hat : array-like of shape (n_samples, n_classes)
The probability of each label category prediction. The order is the lexicographic order of the labels.
For example:
[
[0.2,0.8],
[0.6,0.4],
...

]

Tool

Functions

value_counts_weight

value_counts_weight(data,weight=None)

The functionality is the same 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 occu rred.

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 t hat did not occur', 'Event rate']

profit

profit(y,pred,weight=None

,score_reverse=True,fea_count=None,avg_fea_cost=None

,avg_quota=10000,day_call=10000

,pass_rate=0.4,use_rate=0.7

,avg_profit_rate=0.2,avg_loss_rate=0.8,y_label={'unevent':0,'event':1})

Based on the actual performance and the predicted value given by the model , a rough profit estimate is given

Parameters

----

y : Series

Real performance

pred : Series

Predicted value

weight : Series, optional

Sample weight

Default: None

score_reverse: bool , optional

Relationship between pred and event rate

True: Inverse relationship. The higher the pred value, the lower the event rate.

False: Positive relationship. The higher the pred value, the higher the event rate.

Default: True

fea_count : int

The number of features of the variables included in the model

avg_fea_cost : float

Average unit price of a feature

avg_quota : float, optional

Sample amount (average)

Default: 10000

day_call : int, optional

Average number of model calls per day

Default: 10000

pass_rate : float, optional

Model pass rate

Default: 0.4

use_rate : float, optional

User withdrawal rate

Default: 0.7

avg_gain_rate : float, optional

Average profit per transaction for normal customers

Default: 0.2

avg_loss_rate : float, optional

Average loss per defaulting customer

Default: 0.8

Returns

----

dict

day_bad: The number of default samples generated daily based on the provided y and pred, converted i nto day_call visits

day_good: According to the provided y and pred, it is converted into the normal sample generated daily under the day_call visit volume

bad_rate: default rate calculated according to the provided y, pred and pass_rate

day_total：day_bad + day_good

year_gain: annual income (100 million yuan)

year_loss: annual bad debt amount (100 million yuan),

year_fea_cost: annual credit investigation cost (100 million yuan),

year_profit: annual profit (100 million yuan), year_gain - year_loss - year_fea_cost

prob2score

prob2score(p,base_points=500,base_event_rate=0.05,pdo=50)

Convert a probability into an integer fraction

Parameters

----

p : float

The probability of being converted

base_points : float, optional

Benchmark scores

The default value is 500.

base_event_rate:float, optional

Probability corresponding to the benchmark score (note: not a ratio)

Default is 0.05.

pdo : float, optional

PDO

Default is 50.

Returns

----

int

The returned score value.

load_all_files

load_all_files(path)

Read all data under path, but does not support nested folders

The supported data formats are csv, excel, and pkl. The corresponding suffixes must be csv, xlsx, or pkl.

Parameters

----

path : str

Folder Address

Returns

----

datas : dict{str,dataframe}

All datasets in the folder

key is the name of the file without the suffix as the name of the dataset

get_decimals

get_decimals(i)

Get the number of decimal places in a number

Support scientific notation

Numbers such as 1.0, 2.00, ... are considered to have no decimal places.

Parameters

----

i : numeric

Any value

Returns

----

int

Number of decimal places.

_spec_None

_spec_None(data,spec_value)

If the set empty value does not include 'None', but the data contains empty values, a '{None}' is automatically appended to spec_value

Parameters

----

data : array like

Raw data

spec_value : list

Special Values

Returns

----

new_spec_value : list

Special value after adding '{None}'. If there is 'None' in the original spec_value, it will not be added.

For example, if the original spec_value=['{-1,-2}','{None}'] or spec_value=['{-1,-2}','{-997,None}'] or spec_value=['{-1,-2}',['{-997,None}','{-998}']], the original spec_value will not be changed.

_spec_del

_spec_del(data,spec_value)

Remove special values from a sequence

Parameters

----

data : array like

A column of numbers.

spec_value : list

The value of a special value.

If a special value is not in the sequence, it will be automatically ignored

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

Returns

----

array like has 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

Value to be judged

spec_value : list

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

Returns

----

bool

Whether the value is a special value.

predict_proba

Calculate the predicted probability of the model. It provides the following convenient functions:
1. If clf is statsmodels.generalized_linear_model, a constant term is automatically added, that is, sm.add_constant is automatically called .
2. If clf contains columns with feature_importances_==0 (not actually needed), and these columns do not exist in X, the model can continue to run without errors or inconsistent results .
3. If there are columns in X that are not needed in clf, the model can continue to run without errors or inconsistent results.
4. It will directly return the probability of the event occurring, instead of returning a two-dimensional array consisting of the two probabilities of the event not occurring and the event occurring to other predict_proba methods (for binary classification)

Parameters
----
clf : any model
Model.

X : DataFrame
data

decimals : int, optional
Number of decimal places for probability

Default: 4.

Raises

Exception
If the columns actually required by clf are not in X, an exception is thrown.

Returns
----
pd.Series
    Probability of an event occurring

mean_weight

Same functionality as pandas.Series.mean(), but mean_weight supports weights

Parameters
----
dat : pandas.Series
Number series.

weight : pandas.Series, optional
Sample weight. None: Each sample has the same weight.
Default: None.

decimals: int
Number of decimal places
Default: 4

Returns
----
float
    Weighted mean.

Tree

Functions

auto_xgb

auto_xgb(train_X,train_y,val_X,val_y,train_w=None,val_w=None,metric='ks',cost_time=60*5,cands_num=10,varia
nce_level = 1)

Provides automatic parameter tuning for xgboost. According to tests, models created with other parameter tuning frameworks often have large differences between the training set and validation set metrics. However, the xgboost automatic parameter tuning framework provided by rascpy minimizes the difference between the training set and validation set metrics.

Parameters
----
train_X : DataFrame
Training set X

train_y : Series
Training set label

val_X : DataFrame
Validation set X

val_y : Series
Validation set label

train_w : Series, optional
Training set weights.
Default: None, each sample has the same weight

val_w : Series, optional
Validation set weights.
Default: None, each sample has the same weight

metric : str, optional
Model evaluation indicator, currently supports 'ks' or 'auc'
Default: 'auc'

cost_time : int, optional

The time it takes for auto_xgb to run. Because parameter search is essentially a combinatorial explosion, the goal of any algorithm is to find the most likely optimal set of hyperparameters within a limited time. Therefore, a longer cost_time is, the more likely it is to find the optimal set of hyperparameters.
However, in practice, setting cost_time to 3-5 minutes has yielded the optimal model for most cases. Setting it longer rarely yields a higher-scoring model. If you're not satisfied with the model, you can try increasing cost_time, but increasing it to more than 8 minutes is not recommended and will likely be ineffective.
If the user is not satisfied with the bias or variance of the model, the best approach is not to increase cost_time, but to try using a more accurate sampling method, such as rascpy.Impute.BCSpecValImpute

cands_num : int, optional
When auto_xgb automatically searches for hyperparameters, it gives a score to each hyperparameter it tries. The higher the score, the more recommended the model trained with that hyperparameter is. It then sorts the scores from high to low and returns the models with the top cands_num scores.
In most cases, the model with the highest score (clf_cands[0]) is the best model. However, users can still select their favorite model from the candidate models according to their preferences.
Default: 5

variance_level : int, optional
In actual use, the author generally sets this value to 1, which is sufficient in most cases.
If the difference between the training set and the validation set is small, the user can try to increase this value to increase the model variance and reduce the bias (further improving the model performance on the validation set).
In actual use, the maximum value is set to 2.
Default: 1

Raises

Exception
Metric currently only supports ks and auc. If set to other metrics, an exception will be thrown.
If either val_X or val_y is None, an exception is thrown.

Returns
----
tpule(perf_cands,params_cands,clf_cands,vars_cands)
perf_cands:list<dict>
Metrics for all candidate models. Each set of metrics contains three pieces of information: train_ks(train_auc), val_ks(val_auc), |train - val| (the absolute value of the difference between the training set and the validation set)
params_cands:list<dict>
Hyperparameters for each candidate model
clf_cands:list
Candidate models can be used directly for prediction
vars_cands:list<list>:
The input variables of each candidate model
Note: The indexes of these 4 return values are relative. If the user decides to use the clf_cands[0] model, he can view the model's metrics through perf_cands[0], the model's hyperparameters through params_cands[0], and

the model's input variables through vars_cands[0].

param_plot

param_plot(cv_results_,figsize=(10,10),stats='pi')

By using the cv_results_ generated by various grid searches of sklearn to make a graph, users can easily understand the impact of parameters on model performance by observing the graph.

Because we want to compare the difference between train and test in terms of indicators, we must set return_train_score to True when performing grid search.

For example: GridSearchCV(clf, distributions, return_train_score=True)

The graph is a symmetric matrix-like graph.

The diagonal line shows the impact of changing a single parameter on model performance. The horizontal axis is the parameter's range of values, the left vertical axis is the metrics scaled for the training and test sets, and the right vertical axis is the difference between the metrics on the training and test sets. There are three lines: the first is the average of the metrics on the training set obtained when the parameter is fixed and other parameters are exhausted (i.e., the partial effect; this approach ignores the influence of other parameters and only observes the impact of the current parameter on the model metric), with a confidence interval. The second line is similar, but for the average of the metrics on the test set, with a confidence interval. The third line is the average of the difference between the metrics on the training and test sets.

The lower triangle is a heat map between two parameters. The heat map value is the mean value of the training set obtained by fixing these two parameters and exhaustively enumerating other parameters. The larger the value, the darker the color.

The upper triangle is a heat map of the relationship between two parameters. The heat map value is the mean difference between the training set and the test set obtained by fixing the two parameters and exhaustively enumerating other parameters. The larger the value, the darker the color.

Because the image area is limited, it is recommended to adjust 2-3 parameters at a time, which also helps to reduce the number of combinations and speed up training.

Parameters

----

cv_results_ : GridSearchCV(HalvingGridSearchCV,HalvingRandomSearchCV,RandomizedSearchCV).cv_results_

cv_results_ generated after adjusting the hyperparameters with grid

figsize : tuple

The canvas size passed into the plot

The default is (10,10).

stats : str, optional

Value range: 'ci', 'pi'

Confidence interval calculation method. In most cases, you can choose pi, which marks the range of qu

antiles from 2.5% to 97.5%. If you get an error when entering pi, you can enter ci

The default is 'pi'.

Returns

Generate and display parameter graphs

TreeRej

Functions

auto_rej_xgb

auto_rej_xgb(train_X,train_y,val_X,val_y,rej_train_X,rej_val_X,train_w=None,val_w=None,rej_train_w=None,rej_val_w=None,metric='ks',iter_cost_time=60*5)

xgb refuses to infer the model

Parameters
----
train_X : DataFrame
training set

train_y : Series
Training target

val_X : DataFrame
Validation set

val_y : Series
Verify target

rej_train_X : DataFrame
Refuse to infer the training set

rej_val_X : DataFrame
Reject inference on validation set

train_w : Series, optional
Training set weights
Default: None.

val_w : Series, optional
Validation set weight
Default: None.

rej_train_w : Series, optional
Refuse to infer training set weights
Default: None.

rej_val_w : Series, optional
Refuse to infer validation set weights
Default: None.

metric : TYPE, optional
Model evaluation indicator, currently supports 'ks' or 'auc'
Default: 'auc'

iter_cost_time : int, optional
The time taken for each iteration
Default: 60*5.

Returns
----
not_rej_clf : xgboost
Non-rejection inference of the xgb model

rej_clf : xgboost
Rejecting the inferred xgb model

syn_train : DataFrame
Synthetic data used to train the final round of rejection inference model

syn_val : DataFrame
Synthetic data used to validate the final round of rejection inference models