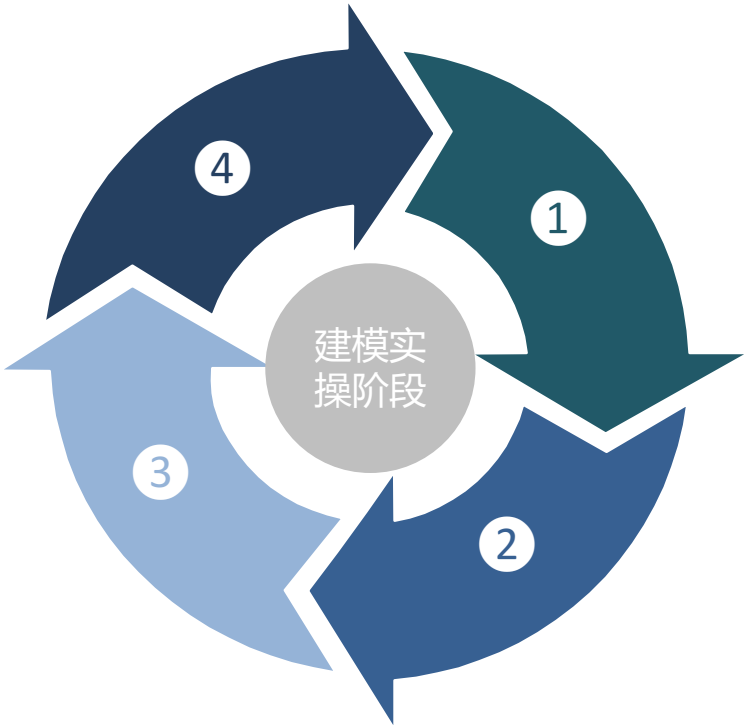
同盾大学
xue.tongdun.cn

让|诚|信|生|活|更|美|好

构建标准化建模代码（上）

C 课程目录 ONTENTS



1 简单实现

可以针对样本进行建模

2 全流程实现

了解建模全流程的理论，并且可以用code实现，那么就可以独立完成一个评分卡项目

3 通用化实现

在实现建模全流程code之后，需要思考怎么把代码通用化，应对到不同场景不同项目都可以复用

4 标准化实现

在实现通用化建模全流程code之后，需要思考如何利用code自动化建模、结果标准化输出
最理想的状态就是：输入样本，调一个函数就可以把建模全流程自动走完，自动整合模型结果并标准输出，然后我们需要做的就是坐在旁边等结果

PYTHON3

针对样本量级巨大，也会涉及到Spark

评分卡全流程代码架构

- standard_model_tool_20180425_V09
  - 01 performance_definition ——— ① 标签定义
    - output_analysis.py ——— 标签定义的分析结果可视化输出
    - vintage_and_roll_rate.py ——— vintage和roll_rate分析
  - 02 reject_inference ——— ② 拒绝推断
    - __init__.py
    - reject_inference_method.py ——— 在标签定义之后使用拒绝推断，可以定义多种拒绝推断方法的函数，然后调用
  - 03 bin_split ——— ③ 变量分箱(分bin)
    - __init__.py
    - best_bin.py ——— 定义变量分bin中的基函数，统计信息函数等功能函数，自动接收切分点输出最终分bin结果
    - ChiMerge_bin.py ——— 卡方分bin方法，输出切分点
    - combine_bin_method.py ——— 合并bin的方法（最大iv、最大gini、最大信息增益等等）
    - cut_bin_method.py ——— 细分bin的方法（最佳ks、最大gini、最大信息增益等等）
    - dynamic_programming_bin.py ——— 动态规划分bin方法，输出切分点
  - 04 feature_analyse_and_woe ——— ④ 变量评价指标计算和woe编码
    - __init__.py
    - base_feature_analyse_and_woe.py ——— 基于变量分箱的code，自动线上spark、线下多线程对所有变量进行分箱、评价信息计算、woe转换样本
    - feature_evaluate_method.py ——— 计算变量评价信息的函数，KS、相关系数、单一阈值、MIC、Iv等等
  - 05 feature_select ——— ⑤ 变量筛选
    - __init__.py
    - feature_select_method.py ——— 变量筛选功能函数，比如剔除高相关系数函数、逐步回归、随机逻辑回归pvalue检验等等
    - feature_select_rule.py ——— 定义单变量筛选和多变量筛选流程，结合变量评价信息和多变量筛选进行自动变量筛选
  - 06 standard_model ——— ⑥ 模型训练
    - __init__.py
    - model_building.py ——— 模型训练code，自动接收筛选后的样本，自动训练模型(切分样本、模型调参、控制训练集测试集稳定性等），得到模型最终结果
  - 07 model_evaluation ——— ⑦ 模型评估
    - __init__.py
    - model_plot.py ——— 模型评估中需要的画图函数，包括ks、roc、PR-F1、lift等等
    - model_result_summary.py ——— 定义模型评估中所有内容的计算函数，并且统一整合
  - 08 result_standard_integration ——— ⑧ 模型结果标准输出
    - __init__.py
    - create_dir.py ——— 定义模型所有结果整合输出的文件夹结构和储存
    - excel_beautify_final.py ——— 整合建模全流程所有结果标准化、美化、自动化输出，可以直接交由评审

1
code demo

变量分箱

变量分箱

同盾大学 xue.tongdun.cn

①构建变量分箱函数之前，需要考虑清楚参数

参数名	参数含义
flag_name	标签列名
factor_name	指标列名
data	样本集
bad_name	坏样本个数列名
good_name	好样本个数列名
piece	最大分组数
rate	每组样本最小占比
min_bin_size	每组样本最小数
not_in_list	缺失值列表
cnt_method	切分方法
combine_method	聚合方法
bin_thought	分bin思想
if_need_control	是否需要对woe列表进行顺序控制

②然后将样本实现如下初步统计

Index	var	#Obs	#Good	#Bad	%Bad_Rate
0	317	1	0	1	100.00%
1	329	2	1	1	50.00%
2	332	14	0	14	100.00%
3	333	6	2	4	66.67%
4	334	9	0	9	100.00%
5	335	4	0	4	100.00%
6	336	7	0	7	100.00%
7	338	37	0	37	100.00%
8	339	7	0	7	100.00%
9	340	14	1	13	92.86%
10	341	23	3	20	86.96%
11	342	20	0	20	100.00%
12	345	1	0	1	100.00%
13	346	3	0	3	100.00%
14	348	7	4	3	42.86%
15	350	2	0	2	100.00%
16	351	11	2	9	81.82%
17	352	19	1	18	94.74%
18	353	8	3	5	62.50%
19	354	26	1	25	96.15%
20	355	13	5	8	61.54%
21	356	13	5	8	61.54%
22	357	23	17	6	26.09%
23	358	19	4	15	78.95%
24	359	33	4	29	87.88%
25	360	28	9	19	67.86%
26	361	17	4	13	76.47%
27	362	12	4	8	66.67%
28	363	14	1	13	92.86%

变量分箱

同盾大学 xue.tongdun.cn

③根据不同的分bin思想，获得最佳切分点
(左开右闭)

Index	var	#Obs	#Good	#Bad	%Bad_Rate
0	317	1	0	1	100.00%
1	329	2	1	1	50.00%
2	332	14	0	14	100.00%
3	333	6	2	4	66.67%
4	334	9	0	9	100.00%
5	335	4	0	4	100.00%
6	336	7	0	7	100.00%
7	338	37	0	37	100.00%
8	339	7	0	7	100.00%
9	340	14	1	13	92.86%
10	341	23	3	20	86.96%
11	342	20	0	20	100.00%
12	345	1	0	1	100.00%
13	346	3	0	3	100.00%
14	348	7	4	3	42.86%
15	350	2	0	2	100.00%
16	351	11	2	9	81.82%
17	352	19	1	18	94.74%
18	353	8	3	5	62.50%
19	354	26	1	25	96.15%
20	355	13	5	8	61.54%
21	356	13	5	8	61.54%
22	357	23	17	6	26.09%
23	358	19	4	15	78.95%
24	359	33	4	29	87.88%
25	360	28	9	19	67.86%
26	361	17	4	13	76.47%
27	362	12	4	8	66.67%
28	363	14	1	13	92.86%

1
2
3
4

④根据切分点统计分bin结果

Characteristic	Description	Bin	WOE	IV_Bin	IV_Total	KS	#Obs
RH_out	变量含义	(-inf, 57.5]	0.83	0.05	0.06	0.17	844
		(57.5, 63.0]	0.10	0.00	0.06	0.07	459
		(63.0, 71.3]	0.00	0.00	0.06	0.09	994
		(71.3, 88.5]	-0.03	0.00	0.06	0.07	3244
		(88.5, inf)	-0.15	0.01	0.06	0.05	3156

#Good	#Bad	%Obs	%Bad	%Cumulative_Bad	%Bad_Rate	%Cumulative_Bad_Rate
772	72	9.70%	4.68%	4.68%	8.53%	8.53%
384	75	5.28%	4.88%	9.56%	16.34%	11.28%
818	176	11.43%	11.44%	21.00%	17.71%	14.06%
2657	587	37.30%	38.17%	59.17%	18.09%	16.42%
2528	628	36.29%	40.83%	100.00%	19.90%	17.68%

②样本初步统计


```
152     @staticmethod
153     def group_by_df(data, flag_name, factor_name, bad_name, good_name,discrete_list=[]):
154         """method"s help string.
155 
156         Explanation
157         ----------
158         转换数据格式，变为指标每一种取值、每一种指标取值下好样本的个数、坏样本的个数
159 
160         Parameters
161         ----------
162         data: dataframe
163             原始样本
164 
165         flag_name: string
166             标签名称
167 
168         factor_name: string
169             指标名称
170 
171         bad_name: string
172             坏样本个数列名
173 
174         good_name: string
175             好样本个数列名
176 
177         discrete_list: list,default=[]
178             指定离散指标的名称list
179         """
180         if len(data) == 0:
181             return pandas.DataFrame()
182         regroup1 = data.groupby([factor_name])[flag_name].count()
183         regroup2 = data.groupby([factor_name])[flag_name].sum()
184         data1 = pandas.DataFrame({good_name: regroup1-regroup2, bad_name: regroup2}).reset_index()
185         data1['%Bad_Rate'] = data1[bad_name]/(data1[bad_name]+data1[good_name])
186         if factor_name not in discrete_list:
187             #data1[factor_name] = data1[factor_name].astype(float)
188             data1 = data1.sort_values(by=[factor_name], ascending=True)
189             #data1[factor_name] = list(map(lambda x:Best_Bin_Cut.get_str(x),data1[factor_name]))
190             data1['Char_Type']='numeric'
191         else:
192             data1 = data1.sort_values(by=['%Bad_Rate'], ascending=True)
```

```
193         data1['Char_Type'] = 'non-numeric'
194     data1 = data1.reset_index(drop=True)
195     return data1
```

❸计算最佳分组切分点


```python
# -*- coding:utf-8 -*-
# creater:
import pandas
import numpy
import math
def ChiMerge_Bin(df,good_name='#Good',bad_name='#Bad', confidenceVal=3.841, maxbin=10,minbin=3,minBinPcnt=0):
    '''变量分箱
    Parameters
    ----------
    df:dataframe,传入一个数据框仅包含一个需要卡方分箱的变量与正负样本标识(响应样本为1，非响应为0)
    var:变量名。var需在df中
    flag:y_train
    confidenceVal:P(K^2≥k)=0.05 表示组间相近的概率为5%，k越大，p越小
    maxbin:最多箱子数量
    minbin:最少箱子数量
    minBinPcnt:组内样本最低占比
    Returns
    ---------
    result_data:dataframe
          包含该变量分箱点,woe,iv等信息
    '''
    var=df.columns[0]
    regroup = df[[var,bad_name,good_name]]
    np_regroup = numpy.array(regroup)  # 把数据框转化为numpy（提高运行效率）

    # 处理连续没有正样本或负样本的区间，并进行区间的合并（以免卡方值计算报错）
    i = 0
    while (i <= np_regroup.shape[0] - 2):
        if ((np_regroup[i, 1] == 0 and np_regroup[i + 1, 1] == 0) or (
                np_regroup[i, 2] == 0 and np_regroup[i + 1, 2] == 0)):
            np_regroup[i, 1] = np_regroup[i, 1] + np_regroup[i + 1, 1]  # 正样本
            np_regroup[i, 2] = np_regroup[i, 2] + np_regroup[i + 1, 2]  # 负样本
            np_regroup[i, 0] = np_regroup[i + 1, 0]
            np_regroup = numpy.delete(np_regroup, i + 1, 0)
            i = i - 1
        i = i + 1
    #先保证最后一组大于5%
    i = np_regroup.shape[0] -1
    total_len=float(sum(np_regroup[:,1]+np_regroup[:,2]))
    while (i >0):
        if ((np_regroup[i, 1] + np_regroup[i, 2]) / total_len >= minBinPcnt):
            break
```

```
            break
        if ((np_regroup[i, 1] + np_regroup[i,2])/total_len<minBinPcnt):
            np_regroup[i, 1] = np_regroup[i, 1] + np_regroup[i - 1, 1]  # 响应样本
            np_regroup[i, 2] = np_regroup[i, 2] + np_regroup[i - 1, 2]  # 非响应样本
            np_regroup = numpy.delete(np_regroup, i - 1, 0)
        i = i - 1
    #计算分组样本占比，把低于5%的分组合并
    i = 0
    total_len=float(sum(np_regroup[:,1]+np_regroup[:,2]))
    while (i <= np_regroup.shape[0] - 2):
        if ((np_regroup[i, 1] + np_regroup[i,2])/total_len<minBinPcnt):
            np_regroup[i, 1] = np_regroup[i, 1] + np_regroup[i + 1, 1]  # 响应样本
            np_regroup[i, 2] = np_regroup[i, 2] + np_regroup[i + 1, 2]  # 非响应样本
            np_regroup[i, 0] = np_regroup[i + 1, 0]
            np_regroup = numpy.delete(np_regroup, i + 1, 0)
            i = i - 1
        i = i + 1
    # 对相邻两个区间进行卡方值计算
    chi_table = numpy.array([])  # 创建一个数组保存相邻两个区间的卡方值  n(ad-bc)^2/(a+b)(c+d)(a+c)(b+d)
    for i in numpy.arange(np_regroup.shape[0] - 1):
        a = np_regroup[i, 1]
        b = np_regroup[i, 2]
        d = np_regroup[i + 1, 2]
        c = np_regroup[i + 1, 1]
        chi = (a + b + c + d) * (a*d - b*c) ** 2 /((a + b) * (c + d) * (a + c) * (b + d))
        chi_table = numpy.append(chi_table, chi)
    print('已完成数据初处理，正在进行卡方分箱核心操作')
    # 把卡方值最小的两个区间进行合并（卡方分箱核心）
    while (1):
        if (len(chi_table)>=(minbin - 1)):
            if (len(chi_table) <= (maxbin - 1) and min(chi_table) >= confidenceVal):
                break
            else:
                chi_min_index = numpy.argwhere(chi_table == min(chi_table))[0]  # 找出卡方值最小的位置索引
                np_regroup[chi_min_index, 1] = np_regroup[chi_min_index, 1] + np_regroup[chi_min_index + 1, 1]
                np_regroup[chi_min_index, 2] = np_regroup[chi_min_index, 2] + np_regroup[chi_min_index + 1, 2]
                np_regroup[chi_min_index, 0] = np_regroup[chi_min_index + 1, 0]
                np_regroup = numpy.delete(np_regroup, chi_min_index + 1, 0) #删除指定行

                '''
                计算合并后分组相关联的卡方值并替换
                1、当chi_min_index位于chi_table最后一个时：计算合并后分组与前一个分组的卡方值
                2、当chi_min_index位于chi_table第一个时：计算合并后分组与后一个分组的卡方值
                3、否则：计算合并后分组与前一个分组、后一个分组的卡方值
                '''
                if (chi_min_index == np_regroup.shape[0] - 1):
```

```
                a = np_regroup[chi_min_index - 1, 1]
                b = np_regroup[chi_min_index - 1, 2]
                c = np_regroup[chi_min_index, 1]
                d = np_regroup[chi_min_index, 2]
                chi_table[chi_min_index - 1] = (a + b + c + d) * (a*d - b*c) ** 2 /((a + b) * (c + d) * (a + c) * (b + d)) #计算与前一组的卡方值
                chi_table = numpy.delete(chi_table, chi_min_index, axis=0) # 删除替换前的卡方值
            elif chi_min_index ==0:
                a1 = np_regroup[chi_min_index, 1]
                b1 = np_regroup[chi_min_index, 2]
                c1 = np_regroup[chi_min_index + 1, 1]
                d1 = np_regroup[chi_min_index + 1, 2]
                chi_table[chi_min_index] = (a1 + b1 + c1 + d1) * (a1*d1 - b1*c1) ** 2 /((a1 + b1) * (c1 + d1) * (a1 + c1) * (b1 + d1)) #计算与后一组的卡方值
                chi_table = numpy.delete(chi_table, chi_min_index + 1, axis=0)  #删除被替换组的卡方值
            else:
                a = np_regroup[chi_min_index - 1, 1]
                b = np_regroup[chi_min_index - 1, 2]
                c = np_regroup[chi_min_index, 1]
                d = np_regroup[chi_min_index, 2]
                chi_table[chi_min_index - 1] = (a + b + c + d) * (a*d - b*c) ** 2 /((a + b) * (c + d) * (a + c) * (b + d)) #计算与前一组的卡方值

                a1 = np_regroup[chi_min_index, 1]
                b1 = np_regroup[chi_min_index, 2]
                c1 = np_regroup[chi_min_index + 1, 1]
                d1 = np_regroup[chi_min_index + 1, 2]
                chi_table[chi_min_index] = (a1 + b1 + c1 + d1) * (a1*d1 - b1*c1) ** 2 /((a1 + b1) * (c1 + d1) * (a1 + c1) * (b1 + d1)) #计算与后一组的卡方值

                chi_table = numpy.delete(chi_table, chi_min_index + 1, axis=0)  #删除被替换组的卡方值

        else:
            break
    #平滑分割点
    cutpoint=list(regroup[regroup[var].isin(np_regroup[:, 0])].index)[0:-1]
    return cutpoint
```

④根据切分点统计分组信息


```
593
594     @staticmethod
595     def important_indicator_calculator(data_df, good_name, bad_name, factor_name, knots_list, na_df,na_zero):
596         """
597         Explanation
598         ----------
599         计算最大ks对应的点
600
601         Parameters
602         ----------
603         data_df: dataframe
604             转换后的数据
605
606         good_name: string
607             好样本个数列名
608
609         bad_name: string
610             坏样本个数列名
611
612         factor_name: string
613             指标列名
614
615         knots_list: list
616             最佳分组点集合
617
618         na_df: dict
619             指标值单独分为一组的样本
620
621         na_zero: Boolean
622             缺失值数量是否小于最小分组数量阈值或者最小分组占比阈值，如果低于阈值，则WOE为0，否则正常计算
623
624         Return
625         ------
626         result_indicator: dataframe
627             分bin结果
628         """
629         flag = data_df['Char_Type'].max()
630         temp_df_list = []
631         bin_list = []
632         for i in range(1, len(knots_list)):
633             if i == 1:
```

```
            temp_df_list.append(data_df.loc[knots_list[i - 1]:knots_list[i]])
            if flag=='numeric':
                bin_list.append('(-inf, ' + Best_Bin_Cut.get_str(data_df[factor_name][knots_list[i]]) + ']')
            else:
                bin_list.append(list(data_df[factor_name])[knots_list[i - 1]:knots_list[i]+1])
        else:
            # if knots_list[i - 1]==knots_list[i]:
            #     temp_df_list.append(data_df.loc[knots_list[i - 1] :knots_list[i]])
            # else:
            #     if knots_list[len(knots_list) - 2]==knots_list[len(knots_list)-1] and i==len(knots_list)-2:
            #         temp_df_list.append(data_df.loc[knots_list[i - 1] + 1:knots_list[i]-1])
            #     else:
            #         temp_df_list.append(data_df.loc[knots_list[i - 1] + 1:knots_list[i]])
            temp_df_list.append(data_df.loc[knots_list[i - 1] + 1:knots_list[i]])
            if flag=='numeric':
                if i == len(knots_list) - 1:
                    bin_list.append('(' + Best_Bin_Cut.get_str(data_df[factor_name][knots_list[i - 1]]) + ', inf)')
                else:
                    bin_list.append(
                        '(' + Best_Bin_Cut.get_str(data_df[factor_name][knots_list[i - 1]]) + ', ' + Best_Bin_Cut.get_str(data_df[factor_name][knots_list[i]]
            else:
                bin_list.append(list(data_df[factor_name])[knots_list[i - 1] + 1:knots_list[i]+1])
    if len(knots_list)==2:
        bin_list=['(-inf, inf)']
    bin_type_list=['Order_Bin']*len(bin_list)
    if len(na_df) != 0:
        tmp_value=list(na_df.values())
        total_good = sum(data_df[good_name]) + sum([sum(key[good_name]) for key in tmp_value])
        total_bad = sum(data_df[bad_name]) + sum([sum(key[bad_name]) for key in tmp_value])
        temp_df_list.extend(tmp_value)
        bin_list.extend(na_df.keys())
        bin_type_list=bin_type_list+['Single_Bin']*len(tmp_value)
    else:
        total_good = sum(data_df[good_name])
        total_bad = sum(data_df[bad_name])
    good_percent_series = pandas.Series(list(map(lambda x: float(sum(x[good_name])) / total_good, temp_df_list)))
    bad_percent_series = pandas.Series(list(map(lambda x: float(sum(x[bad_name])) / total_bad, temp_df_list)))
    woe_list = list(numpy.log(good_percent_series / bad_percent_series))
    IV_list = list((good_percent_series - bad_percent_series) * numpy.log(good_percent_series / bad_percent_series))
    tmp_key=list(na_zero.keys())
    for zero in na_zero:
        if na_zero[zero]:
            tmp_num=len(tmp_key)-tmp_key.index(zero)
            woe_list[-tmp_num]=0
            IV_list[-tmp_num]=0
```

```
    #
    # if na_zero:
    #     woe_list=woe_list[0:-1]+[0]
    #     IV_list=IV_list[0:-1]+[0]
    IV_total=sum(IV_list)
    KS_list=list(map(lambda x:max(abs(numpy.cumsum(x[good_name]) / float(sum(x[good_name]))-numpy.cumsum(x[bad_name]) / float(sum(x[bad_name])))),temp_df_list))
    ks_total=max(abs(numpy.cumsum(data_df[good_name]) / float(sum(data_df[good_name]))-numpy.cumsum(data_df[bad_name]) / float(sum(data_df[bad_name]))))
    good_list = list(map(lambda x: sum(x[good_name]), temp_df_list))
    bad_list = list(map(lambda x: sum(x[bad_name]), temp_df_list))
    total_list = list(map(lambda x: sum(x[good_name])+sum(x[bad_name]), temp_df_list))
    total_bad_rate_list = list(map(lambda x: float(sum(x[bad_name])) / total_bad, temp_df_list))
    cumsum_total_bad_rate_list=numpy.cumsum(total_bad_rate_list)
    bin_bad_rate_list = list(
        map(lambda x: float(sum(x[bad_name])) / (sum(x[good_name]) + sum(x[bad_name])), temp_df_list))
    bin_rate_list=list(map(lambda x: float(sum(x[good_name]) + sum(x[bad_name])) / (total_good+total_bad), temp_df_list))
    non_na_indicator = pandas.DataFrame({'Bin': bin_list,
                                         'Bin_Type':bin_type_list,
                                         'WOE': woe_list,
                                         'IV_Bin': IV_list,
                                         'IV_Total':IV_total,
                                         'KS':KS_list,
                                         'KS_Total':ks_total,
                                         '#Obs':total_list,
                                         '#Good':good_list,
                                         '#Bad': bad_list,
                                         '%Obs': bin_rate_list,
                                         '%Bad':total_bad_rate_list,
                                         '%Cumulative_Bad': cumsum_total_bad_rate_list,
                                         '%Bad_Rate': bin_bad_rate_list,
                                         'Char_Type': flag})
    non_na_indicator['%Cumulative_Bad_Rate']=numpy.cumsum(non_na_indicator[bad_name])/numpy.cumsum(non_na_indicator['#Obs'])
    result_indicator = non_na_indicator.reset_index(drop=True)
    result_indicator['Characteristic']=factor_name
    cop_bad_rate=list(result_indicator[result_indicator['Bin_Type']=='Order_Bin']['%Bad_Rate'])
    result_indicator['Char_Direction']=Best_Bin_Cut.trend_analyse(cop_bad_rate)
    columns_order=['Characteristic','Char_Type','Bin','Char_Direction','Bin_Type','WOE','IV_Bin','IV_Total','KS','KS_Total','#Obs','#Good','#Bad','%Obs','%Bad'
    result_indicator=result_indicator[columns_order]
    return result_indicator
```

2
code demo

变量评价指标计算和woe编码

1 变量评价指标函数

①变量评价指标函数


```python
class Feature_Evaluate_Method(object):

    def __init__(self):
        pass

    @staticmethod
    def single_threshold(data,factor_name):
        """
        Explanation
        ----------
        计算单一阈值

        Remark
        (1) 适用于连续性指标、离散有序型指标、离散无序型指标
        (2) 适用于粗分类后的新指标(哑变量或者woe证据权重)
        (3) 缺失值也参与计算
        """
        percent=data[factor_name].value_counts(normalize=True, dropna=False)
        return percent.max()

    @staticmethod
    def correlation(data,flag_name,factor_name,corr_method):
        """
        Explanation
        ----------
        计算相关系数

        Remark
        (1) 适用于连续性指标、离散有序型指标
        (2) 适用于粗分类后的新指标(哑变量或者woe证据权重)
        (3) 如果指标中存在缺失值，剔除缺失值计算
        """
        try:
            data[factor_name]=data[factor_name].astype(float)
            return data[[flag_name,factor_name]].corr(method=corr_method)[flag_name][factor_name]
        except:
            return None

    @staticmethod
    def KS(data,flag_name,factor_name):
        """
```

```
        Explanation
        ----------
        计算KS,以及KS对应的指标取值

        Remark
        (1) 适用于连续性指标、离散有序型指标、离散无序型指标
        (2) 适用于粗分类后的新指标(哑变量或者woe证据权重)
        (3) 如果指标中存在缺失值，剔除缺失值计算
        """
        data1 = data[[flag_name, factor_name]]
        if len(data1)==0:
            return None,None
        data1=data1[data1[factor_name].notnull()]
        data1=Best_Bin_Cut.group_by_df(data1,flag_name,factor_name,'bad','good')
        if len(data1)==0:
            return 0,None
        default_CDF = numpy.cumsum(data1['bad']) / sum(data1['bad'])
        undefault_CDF = numpy.cumsum(data1['good']) / sum(data1['good'])
        ks_CDF=abs(default_CDF - undefault_CDF)
        ks = max(ks_CDF)
        if numpy.isnan(ks):
            return None,None
        index=list(filter(lambda x:ks_CDF.loc[x]==ks,ks_CDF.index))[0]
        value=data1[factor_name][index]
        return ks,value

    @staticmethod
    def IV(data,flag_name,factor_name):
        """
        Explanation
        ----------

        Remark
        (1) 适用于连续性指标、离散有序型指标、离散无序型指标
        (2) 适用于粗分类后的新指标(哑变量或者woe证据权重)
        (3) 如果指标中存在缺失值，剔除缺失值计算
        """
        data1=data[[flag_name,factor_name]]
        data1=data1[data1[factor_name].notnull()]
        data1=Best_Bin_Cut.group_by_df(data1,flag_name,factor_name,'bad','good')
        good=float(sum(data1['good']))
        bad=float(sum(data1['bad']))
        data1['iv']=list(map(lambda x,y:(x/good-y/bad)*numpy.log(x*bad/(good*y)) if good*y!=0 else numpy.inf,data1['good'],data1['bad']))
        return data1['iv'].sum()
```

轻量级

样本描述：样本数20000；变量数6000
计算方法：线下python多进程计算
代码效率：完成所有变量的分箱只需要1.5min～2min；分箱的同时再计算变量评价指标、woe转换，只需要4min～5min
Ps：线下即使不使用多进程，通过优化分箱代码，循环所有变量可以使得速度达到3min

重量级

样本描述：样本数100万；变量数40000
计算方法：线上spark分布式计算
代码效率：完成所有变量的分箱只需要15min～25min；分箱的同时再计算变量评价指标、woe转换，只需要25min～35min

线上spark分布式计算

在数据量达到100万，变量数达到40000时，针对单个变量进行分箱，python尚能支持，但是对40000变量全部分箱，python根本无法运行起来，必须结合spark分布式计算，但是了解spark的同学都知道，spark的map过程，每次传入的对象是一行，因此无法针对标签列和指标列进行计算，也就无法实现变量分箱计算

通过spark将样本行列转置之后，map过程每次传入的对象实际上就变成了样本的每列，因此可以计算变量分箱，再结合分布式任务，可以实现快速计算

②线上spark分布式变量分组


```
257
258     @staticmethod
259     def online_data_trans_rdd(data, data_oot,flag_name, not_var_list=[]):
260         """
261         Explanation
262         ----------
263         (1) 线上：对spark里的dataframe进行行列转置,变成rdd
264         (2) 线下：或者对pandas里的dataframe进行转置
265         """
266         if type(data_oot) == dataframe.DataFrame:
267             data_total = data.union(data_oot)
268         else:
269             data_total = data
270         y = data_total.select(flag_name).toPandas()
271         n_sample = data.count()
272         data_total = data_total.select(list(set(data_total.columns) - set(not_var_list)))
273         columns = data_total.columns
274         # 获得变量列名，用于标记spark每次map时计算的变量名称，变量名称和变量在list中的序号一一对应
275         rdd_columns = sc.parallelize([tuple(columns)])
276         # 将变量名称和样本数据构造成rdd，此时每次map的对象仍然是样本的一行
277         rdd_tuple = rdd_columns + data_total.rdd.map(tuple)
278         # 将m个样本n个指标拉评为m*n条数据，每行数据为变量取值、变量名称、变量序号、并且通过zipwithindex从0到m*n-1对m*n条数据进行编号
279         rdd_add_index = rdd_tuple.zipWithIndex().flatMap(lambda x: [(x[1], k[0], k[1]) for k in enumerate(x[0])])
280         # 再根据变量序号进行分组，将同一变量的所有取值合并到一起，此时每次map的对象已经变为同一变量所有样本取值
281         rdd_groupby_index = rdd_add_index.map(lambda x: (x[1], (x[0], x[2]))).groupByKey().sortByKey()
282         # 再根据样本index排序，使得不同变量的取值一一对应
283         rdd_combine = rdd_groupby_index.map(lambda x: sorted(list(x[1]), key=lambda k:k[0]))
284         # 再通过map函数去掉index，只保留变量名称和变量取值列表
285         rdd_transed = rdd_combine.map(lambda x: map(lambda y: y[1], x))
286         # 最后通过map函数将变量名称和变量取值列表分开
287         rdd_final = rdd_transed.map(lambda x: (x[0], x[1:]))
288         # 至此，已经实现样本的行列转置，得到我们需要的数据
289         return y, rdd_final, n_sample
290
```

③样本WOE转换

原始样本只保留标签列和指标列

对变量分组的同时，根据分组结果将对应的变量进行WOE编码（主要应对于逻辑回归入模）

③样本woe转换


```
    @staticmethod
    def get_trans_woe_data(data_bin,data,factor_name):
        """
        Explanation
        ----------
        根据分bin结果将指标原始值转换为woe
        1. 缺失值单独使用pandas.replace函数
        2. 对于连续型指标，使用pandas.cut函数
        3. 对于离散型指标，使用pandas.replace函数
        """
        data_select=data[[factor_name]]
        tmp1=data_bin[data_bin['Bin_Type']=='Order_Bin']
        tmp2 = data_bin[data_bin['Bin_Type'] != 'Order_Bin']
        char_type=data_bin['Char_Type'][0]
        if char_type=='numeric':
            knot_list = [-numpy.inf] + list(map(lambda x: float(x.split(', ')[-1][:-1]), tmp1['Bin']))
            woe_list = list(tmp1['WOE'])
            if len(tmp2)==0:
                return pandas.DataFrame(pandas.cut(data_select[factor_name], bins=knot_list, labels=woe_list,include_lowest=True))
            else:
                single_dict=dict(zip(tmp2['Bin'],tmp2['WOE']))
                single_bin_value, not_in_single = Base_Feature_Analyse.add_single_value(factor_name)
                data1=data_select[(data_select[factor_name].isin(not_in_single)) | data_select[factor_name].isnull()]
                data2=data_select.loc[data_select.index^data1.index]
                data2[factor_name]=pandas.cut(data2[factor_name], bins=knot_list, labels=woe_list,include_lowest=True)
                data2[factor_name]=data2[factor_name].astype(float)
                for key in single_dict:
                    data1=data1.replace(single_bin_value[key],single_dict[key])
                if 'NA' in single_dict:
                    data1=data1.fillna(single_dict['NA'])
                trans_data=pandas.concat([data1,data2]).sort_index()
                return trans_data
        else:
            single_dict = dict(zip(data_bin['WOE'],data_bin['Bin']))
            if len(tmp2)==0:
                for key in single_dict:
                    data_select = data_select.replace(single_dict[key],key)
                return data_select
            else:
                single_bin_value, not_in_single = Base_Feature_Analyse.add_single_value(factor_name)
                for woe in single_dict:
```

```
            if type(single_dict[woe])!=list:
                if single_dict[woe] in single_bin_value:
                    if single_dict[woe]=='NA':
                        data_select = data_select.fillna(woe)
                    single_dict[woe]=single_bin_value[single_dict[woe]]
                data_select = data_select.replace(single_dict[woe], woe)
        return data_select
```

结果展示

一、所有变量分组信息结果展示如下

Characterist	Description	Bin	WOE	IV_Bin	IV_Total	KS	#Obs	#Good	#Bad	%Obs	%Bad	%Cumulative_Bad	%Bad_Rate	%Cumulative_Bad_Rate
RH_out	变量含义	(-inf, 57.5]	0.83	0.05	0.06	0.17	844	772	72	9.70%	4.68%	4.68%	8.53%	8.53%
		(57.5, 63.0]	0.10	0.00	0.06	0.07	459	384	75	5.28%	4.88%	9.56%	16.34%	11.28%
		(63.0, 71.3333333333]	0.00	0.00	0.06	0.09	994	818	176	11.43%	11.44%	21.00%	17.71%	14.06%
		(71.3333333333, 88.5]	-0.03	0.00	0.06	0.07	3244	2657	587	37.30%	38.17%	59.17%	18.09%	16.42%
		(88.5, inf)	-0.15	0.01	0.06	0.05	3156	2528	628	36.29%	40.83%	100.00%	19.90%	17.68%
T2	变量含义	(-inf, 20.9725]	-0.12	0.01	0.13	0.13	5899	4748	1151	67.83%	74.84%	74.84%	19.51%	19.51%
		(20.9725, 22.4266666667]	-0.10	0.00	0.13	0.14	1419	1146	273	16.32%	17.75%	92.59%	19.24%	19.46%
		(22.4266666667, 23.29]	0.34	0.01	0.13	0.09	522	453	69	6.00%	4.49%	97.07%	13.22%	19.04%
		(23.29, inf)	1.35	0.11	0.13	0.18	857	812	45	9.85%	2.93%	100.00%	5.25%	17.68%
RH_3	变量含义	(-inf, 39.2285714286]	0.33	0.06	0.19	0.08	4976	4312	664	57.22%	43.17%	43.17%	13.34%	13.34%
		.2285714286, 42.13333333	-0.09	0.00	0.19	0.23	1707	1381	326	19.63%	21.20%	64.37%	19.10%	14.81%
		.1333333333, 44.29666666	-0.31	0.01	0.19	0.14	1102	853	249	12.67%	16.19%	80.56%	22.60%	15.92%
		(44.2966666667, 45.09]	-0.31	0.01	0.19	0.18	467	361	106	5.37%	6.89%	87.45%	22.70%	16.30%
		(45.09, inf)	-1.27	0.11	0.19	0.35	445	252	193	5.12%	12.55%	100.00%	43.37%	17.68%
T8	变量含义	(-inf, 21.1611111111]	-0.45	0.07	0.22	0.06	2659	1991	668	30.57%	43.43%	43.43%	25.12%	25.12%
		.1611111111, 22.61111111	-0.08	0.00	0.22	0.06	2615	2120	495	30.07%	32.18%	75.62%	18.93%	22.05%
		.6111111111, 24.06666666	0.32	0.02	0.22	0.04	2161	1869	292	24.85%	18.99%	94.60%	13.51%	19.57%
		(24.0666666667, 25.18]	0.95	0.06	0.22	0.14	825	762	63	9.49%	4.10%	98.70%	7.64%	18.38%
		(25.18, inf)	1.50	0.07	0.22	0.16	437	417	20	5.02%	1.30%	100.00%	4.58%	17.68%
Press_mm_hg	变量含义	(-inf, 752.15]	0.15	0.01	0.07	0.14	2726	2302	424	31.34%	27.57%	27.57%	15.55%	15.55%
		(752.15, 756.488888889]	0.14	0.00	0.07	0.09	1903	1603	300	21.88%	19.51%	47.07%	15.76%	15.64%
		6.488888889, 764.1333333	0.03	0.00	0.07	0.12	2888	2389	499	33.21%	32.44%	79.52%	17.28%	16.27%
		(764.133333333, 766.6]	-0.33	0.01	0.07	0.22	740	570	170	8.51%	11.05%	90.57%	22.97%	16.87%
		(766.6, inf)	-0.83	0.04	0.07	0.15	440	295	145	5.06%	9.43%	100.00%	32.95%	17.68%
RH_2	变量含义	(-inf, 33.5266666667]	0.51	0.01	0.03	0.23	445	394	51	5.12%	3.32%	3.32%	11.46%	11.46%
		.5266666667, 35.62666666	0.27	0.00	0.03	0.19	511	439	72	5.88%	4.68%	8.00%	14.09%	12.87%
		.6266666667, 37.69666666	0.09	0.00	0.03	0.11	1056	882	174	12.14%	11.31%	19.31%	16.48%	14.76%
		(37.6966666667, 41.718]	0.05	0.00	0.03	0.08	3380	2806	574	38.86%	37.32%	56.63%	16.98%	16.15%
		(41.718, inf)	-0.16	0.01	0.03	0.10	3305	2638	667	38.00%	43.37%	100.00%	20.18%	17.68%

二、变量评价信息计算结果展示如下

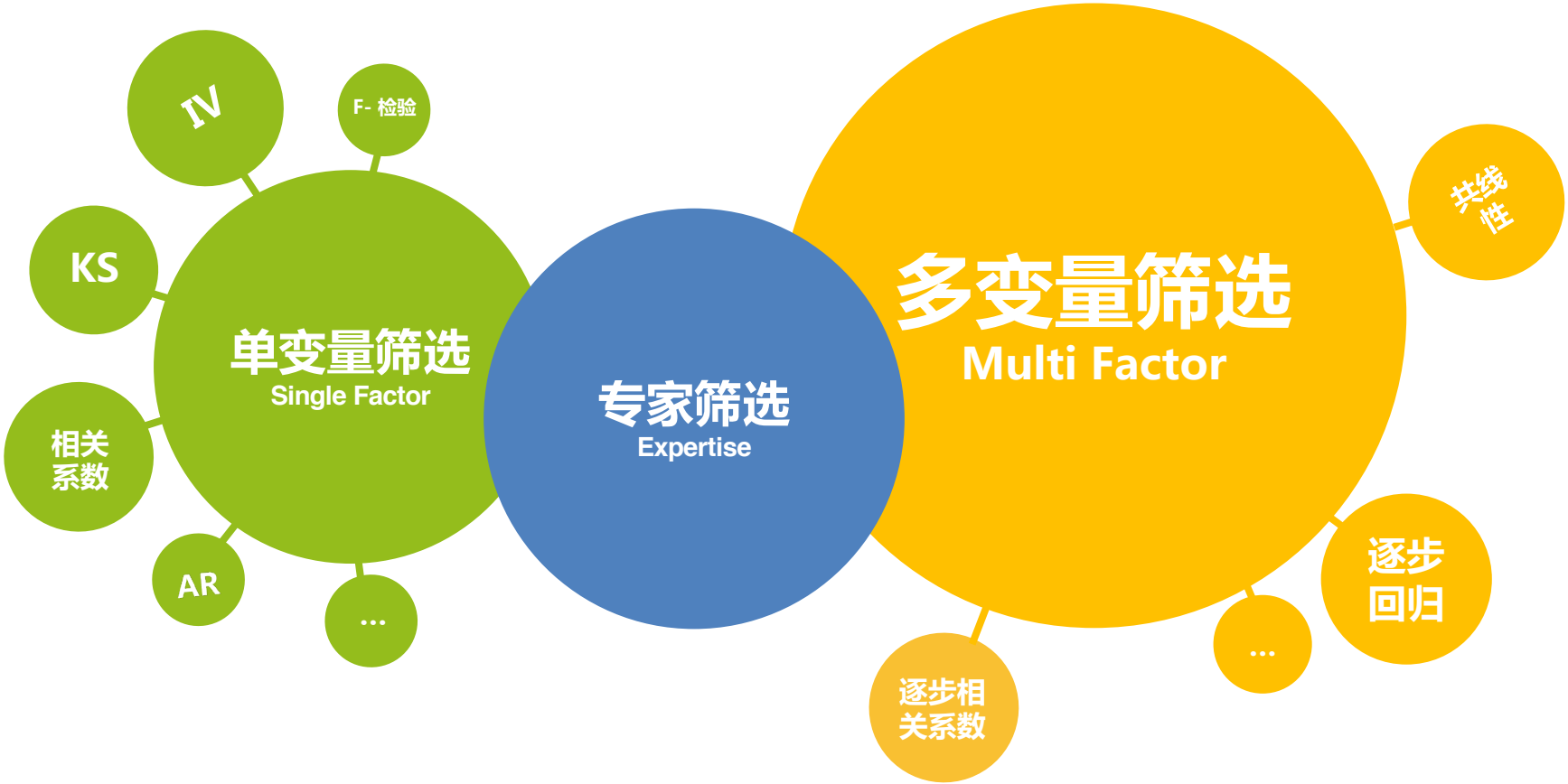
14.变量评价信息(Variables Effect)

Characteristic	Sample	AR	Corr_Pearson	Corr_Spearman	GINI	IV	KS	Nonempty_Bin_Num	Single_Threshold
samegps_xf_custnum_before1m_after1d_6geohash	Development Sample	0.28	0.22	0.17	0.28	0.29	0.21	4	43.22%
samegpscustnum1m3dec	Development Sample	0.26	0.20	0.15	0.26	0.28	0.20	5	3.03%
samegps_xf_over80_accountnum_before1m_after3d_6geohash	Development Sample	0.26	0.21	0.16	0.26	0.28	0.20	4	49.96%
samegpsapplnum1m3dec	Development Sample	0.26	0.20	0.14	0.26	0.28	0.20	5	2.95%
samegps_appl_count_1m	Development Sample	0.26	0.20	0.15	0.26	0.28	0.20	5	2.95%
samegpspreriskpolicyrejcustnum1m3dec	Development Sample	0.26	0.20	0.14	0.26	0.26	0.19	5	4.30%
samegpsrejcustnum1m3dec	Development Sample	0.25	0.20	0.14	0.25	0.26	0.21	5	3.54%
samegps_xf_custnum_before1m_after3d_6geohash	Development Sample	0.27	0.21	0.16	0.27	0.25	0.20	4	34.92%
samegps_tx_or_xf_over80_accountnum_before1m_after3d_6geohash	Development Sample	0.28	0.21	0.16	0.28	0.25	0.23	5	20.85%
samegps_tx_or_xf_custnum_before1m_after3d_6geohash	Development Sample	0.27	0.21	0.15	0.27	0.25	0.22	5	20.72%
samegps_xf_over80_accountnum_before1m_after1d_6geohash	Development Sample	0.24	0.21	0.15	0.24	0.25	0.19	3	57.62%
samegpsapprcustnum1m3dec	Development Sample	0.26	0.20	0.14	0.26	0.25	0.20	5	11.92%
samegpsrejcustnum7d3dec	Development Sample	0.25	0.18	0.14	0.25	0.24	0.18	5	11.71%
samegpscustnum7d3dec	Development Sample	0.24	0.18	0.14	0.24	0.23	0.17	5	9.48%
samegps_tx_or_xf_over80_accountnum_before1m_after1d_6geohash	Development Sample	0.26	0.21	0.15	0.26	0.23	0.19	5	21.40%
samegpsapplnum7d3dec	Development Sample	0.24	0.18	0.14	0.24	0.23	0.17	5	9.27%
samegps_tx_or_xf_recordnum_before1m_after1d_6geohash	Development Sample	0.26	0.19	0.14	0.26	0.22	0.21	5	15.67%
samegpspreriskpolicyrejcustnum7d3dec	Development Sample	0.25	0.18	0.14	0.25	0.22	0.19	5	12.64%
samegps_tx_or_xf_custnum_before1m_after1d_6geohash	Development Sample	0.26	0.21	0.15	0.26	0.22	0.20	5	21.82%
samegps_tx_or_xf_recordnum_before1m_after3d_6geohash	Development Sample	0.25	0.17	0.14	0.25	0.20	0.20	5	12.38%
samegps_xf_custnum_before1m_after1h_6geohash	Development Sample	0.20	0.18	0.14	0.20	0.19	0.18	3	66.39%
samegps_xf_recordnum_before1m_after1d_6geohash	Development Sample	0.24	0.16	0.14	0.24	0.18	0.19	5	43.22%
samegps_xf_recordnum_before1m_after3d_6geohash	Development Sample	0.23	0.13	0.13	0.23	0.17	0.17	5	34.92%
samegps_xf_custratio_before1m_after1d_6geohash	Development Sample	0.22	0.12	0.13	0.22	0.16	0.19	5	43.22%
samegps_tx_or_xf_recordnum_before1m_after1h_6geohash	Development Sample	0.21	0.17	0.12	0.21	0.16	0.16	5	22.45%
samegps_tx_custnum_before1m_after1d_6geohash	Development Sample	0.17	0.17	0.10	0.17	0.16	0.16	3	21.90%
samegps_first_xf_min_diff_before1m_after1d_6geohash	Development Sample	-0.17	-0.08	-0.10	-0.17	0.16	0.15	4	52.86%
samegps_tx_or_xf_custnum_before1m_after1h_6geohash	Development Sample	0.21	0.18	0.12	0.21	0.16	0.16	5	24.30%
samegps_tx_custnum_before1m_after3d_6geohash	Development Sample	0.17	0.17	0.10	0.17	0.15	0.17	5	22.87%
samegps_xf_over80_accountratio_before1m_after3d_6geohash	Development Sample	0.21	0.13	0.13	0.21	0.15	0.19	5	49.96%
samegps_xf_ratio_before1m_after1d_6geohash	Development Sample	0.09	0.08	0.06	0.09	0.15	0.12	3	52.86%
samegps_xf_ratio_before1m_after1h_6geohash	Development Sample	0.09	0.10	0.06	0.09	0.15	0.13	2	76.03%
samegps_xf_recordnum_before1m_after1h_6geohash	Development Sample	0.19	0.14	0.14	0.19	0.14	0.18	3	66.39%

3
code demo

变量筛选

变量筛选方法


IV
F- 检验
KS
单变量筛选
Single Factor
相关系数
AR
...
专家筛选
Expertise
多变量筛选
Multi Factor
共线性
逐步回归
...
逐步相关系数

变量筛选规则

多变量分析函数

- 在第二环节已经将单变量分析函数构造完成
- 此部分定义多变量分析函数
- **为何不将单变量分析函数在此部分定义**：所有变量一起分箱时一起计算单变量评价指标，节省时间

变量筛选流程

- **制定标准变量筛选流程**
- 将制定好的变量筛选流程函数化，并且针对每一种筛选方法配制一个控制参数，保证筛选流程可以灵活组合

自动筛选变量

- **单变量筛选和多变量筛选可以实现自动化**
- 专家筛选在很多场景不可缺少，但是专家筛选很难自动化
- 因此预留专家筛选缺口，保证单变量筛选和多变量筛选的自动化

①多变量分析函数


```python
class Base_Feature_Select_Method(object):

    def __init__(self):
        pass

    @staticmethod
    def eliminate_high_corr(train_data_woe,train_data_stat, not_var_list, corr_rate):
        """
        Explanation
        ----------
        筛选出相关系数小于指定阈值的变量
        对于相关系数大于等于指定阈值的变量，只保留iv最大的变量

        Parameters
        ----------
        train_data_woe:pandas.core.frame.DataFrame
            训练集woe化样本

        train_data_stat:pandas.core.frame.DataFrame
            训练集指标表现

        not_var_list:list
            不参与筛选的指标列表

        corr_rate:float
            相关系数阈值

        Return
        ------
        corr_list:list
            训练集相关系数小于指定阈值的指标名称list
        """
        var_list = list(set(train_data_stat['Characteristic'])-set(not_var_list))
        '''计算所有指标之间的相关系数，并且转化为矩阵'''
        train_corr = train_data_woe[var_list].corr().as_matrix()
        n=len(var_list)
        for i in range(n):
            '''将矩阵的上半部分变为0，因为相关系数矩阵为对称矩阵，因此nxn矩阵只需要比较n*(n+1)/2即可，不需要n*n次'''
            train_corr[i]=list(map(lambda x:0 if x>=i else train_corr[i][x],range(n)))
        max_corr=1
        while max_corr>=corr_rate:
```

```
        print(max_corr)
        n=train_corr.shape[0]
        '''找到相关系数最大的行列索引'''
        max_arg=numpy.argmax(train_corr)
        max_row=max_arg//n
        max_col=max_arg%n
        iv_row=train_data_stat[train_data_stat['Characteristic']==var_list[max_row]]['IV'].max()
        iv_col = train_data_stat[train_data_stat['Characteristic'] == var_list[max_col]]['IV'].max()
        '''根据iv比较出需要删除的指标'''
        del_max=max_row if iv_row<=iv_col else max_col
        train_corr=numpy.delete(train_corr,del_max,axis=1)
        train_corr = numpy.delete(train_corr, del_max, axis=0)
        var_list.remove(var_list[del_max])
        max_corr=train_corr.max()
        print(max_corr)
        '''如此循环，直到所有指标的相关系数均小于既定的阈值'''
    return var_list
```

②变量筛选流程

优先级顺序	规则名称	规则详细逻辑	规则参数设置	规则状态	备注
CS01	单一阈值筛选	1、计算每个变量的每种取值占比 2、设定单一阈值阀值 3、遍历每个变量： 如果max(变量的不同取值占比)> 单一阈值阀值，则剔除变量；否则保留变量	单一阈值=0.9	启用	每一次筛选都是承接上一次筛选结果
CS02	最小变量分组数量	1. 变量分组默认分为5组，如果存在缺失值，则分6组，缺失值单独分为一组;分组要求，每组数量占比不得低于5% 2. 如果除去缺失值，剩下的无法分出大于等于“最小变量分组数量”的组数，则剔除变量;否则保留	最小变量分组数量=3	启用	
CS03	IV筛选	1、计算每个变量的最优分组 2、设定IV阀值 3、遍历每个变量： 如果变量IV< 单一阈值阀值，则剔除变量；否则保留变量	IV阀值=0.02	启用	
CS04	PSI筛选	1. 如果建模样本的时间点跨度几个月，甚至更长，可以按月切分或者更长步长切分 2. 选择时间最早的一组作为基本样本，剩下的作为验证样本，计算组间的PSI，剔除PSI大于"PSI阀值"的变量	PSI阀值=0.05	启用	该部分视样本时间点实际情况而定
CS05	相关系数筛选	1、计算所有变量之间的相关性 2、设定相关系数阀值 3、遍历每个变量： 选择出与该变量相关系数大于相关系数阀值的所有变量，只保留这些变量中IV 最大额那个变量，其余变量删除	相关系数阀值=0.8	启用	
CS06	专家筛选	1. 专家结合实际业务场景，分析指标实际含义 2. 对于表现不错但是实际含义无法解释，或者分组无法解释的变量，剔除		启用	该部分视实际业务场景和指标含义而定
CS07	AR筛选	1、设定随机切分比例 2、随机切分数据集为训练集、测试集，计算训练集、测试集每个变量的AR值 3、遍历每个变量： 如果该变量在训练集、测试集上的 AR值负号相反，则剔除变量，否则保留	随机切分比例=6:4	未启用	
		1、计算所有变量之间的最大互信息数MIC			

CS08	MIC筛选	2、设定MIC阈值 3、遍历每个变量： 如果该变量的MIC<MIC阈值，则剔除；否则保留	MIC阈值=0.1	未启用	
CS09	皮尔逊卡方检验筛选	1、计算所有变量的皮尔逊卡方检验统计量的pvalue 2、设定置信水平alpha 3、遍历每个变量： 如果该变量的pvalue>alpha，则剔除；否则保留	alpha=0.05	未启用	
CS10	似然比检验筛选	1、计算所有变量的似然比检验统计量的pvalue 2、设定置信水平alpha 3、遍历每个变量： 如果该变量的pvalue>alpha，则剔除；否则保留	alpha=0.05	未启用	
CS11	F检验筛选	1、计算所有变量的F检验统计量的pvalue 2、设定置信水平alpha 3、遍历每个变量： 如果该变量的pvalue>alpha，则剔除；否则保留	alpha=0.05	未启用	
CS12	随机逻辑回归筛选	1、输入所有变量和标签 2、设定变量筛选惩罚系数、变量筛选阈值、随机建模种子 3、通过RLR随机逻辑回归建模，计算出通过筛选的变量	筛选惩罚系数=3 变量筛选阈值=0.1 随机建模种子=0	未启用	
CS13	逐步回归筛选	1、输入所有变量和标签 2、设定选入变量置信水平alpha1、剔除变量置信水平alpha2 3、通过逐步回归，每次对逻辑回归通过p检验的变量进行选入，对未通过p检验的变量进行剔除，逐步回归至结束	alpha1=0.05 alpha2=0.05	未启用	

②变量筛选流程


```
class Base_Feature_Select_Rule(object):

    @staticmethod
    def initial_feature_select(data_original,data_stat,discrete_list=[],single_threshold=0.95,iv=0.02,corr=0.85,mic=None,min_bin_num=2):
        """
        Explanation
        -----------
        根据评价指标阀值，初步筛选出符合条件的指标
        """
        crete_list=list(set(data_stat['Characteristic'])-set(discrete_list))
        try:
            discrete_data_stat=data_stat[(data_stat['Single_Threshold']!='nan') & (data_stat['Characteristic'].isin(discrete_list))]
            crete_data_stat = data_stat[(data_stat['Single_Threshold'] != 'nan') & (data_stat['Characteristic'].isin(crete_list))]
        except:
            discrete_data_stat=data_stat[(data_stat['Single_Threshold'].notnull()) & (data_stat['Characteristic'].isin(discrete_list))]
            crete_data_stat = data_stat[(data_stat['Single_Threshold'].notnull()) & (data_stat['Characteristic'].isin(crete_list))]
        exclue_dic={'Single_Threshold':[]}
        if single_threshold:
            crete_data_stat = crete_data_stat[crete_data_stat['Single_Threshold'] <= single_threshold]
            discrete_data_stat = discrete_data_stat[discrete_data_stat['Single_Threshold'] <= single_threshold]
            exclue_dic['Single_Threshold']=list(set(data_stat['Characteristic'])-set(crete_data_stat['Characteristic'])-set(discrete_data_stat['Characteristic']))
        if iv:
            exclue_list=reduce(lambda x,y:x+y,exclue_dic.values())
            crete_data_stat = crete_data_stat[crete_data_stat['IV'] >= iv]
            discrete_data_stat = discrete_data_stat[discrete_data_stat['IV'] >= iv]
            exclue_dic['IV'] = list(set(data_stat['Characteristic']) - set(exclue_list) - set(
                crete_data_stat['Characteristic']) - set(discrete_data_stat['Characteristic']))
        if mic:
            exclue_list = reduce(lambda x, y: x + y, exclue_dic.values())
            crete_data_stat = crete_data_stat[crete_data_stat['MIC'] >= mic]
            discrete_data_stat = discrete_data_stat[discrete_data_stat['MIC'] >= mic]
            exclue_dic['MIC'] = list(set(data_stat['Characteristic']) - set(exclue_list) - set(
                crete_data_stat['Characteristic']) - set(discrete_data_stat['Characteristic']))
        if min_bin_num:
            exclue_list = reduce(lambda x, y: x + y, exclue_dic.values())
            crete_data_stat = crete_data_stat[crete_data_stat['Nonempty_Bin_Num']>=min_bin_num]
            exclue_dic['Min_Bin_Num'] = list(set(data_stat['Characteristic']) - set(exclue_list) - set(
                crete_data_stat['Characteristic']) - set(discrete_data_stat['Characteristic']))
        if corr:
            exclue_list = reduce(lambda x, y: x + y, exclue_dic.values())
            corr_list=Base_Feature_Select_Method.eliminate_high_corr(data_original,crete_data_stat, [], corr)
            exclue_dic['Corr'] = list(set(data_stat['Characteristic']) - set(exclue_list) - set(corr_list) - set(discrete_data_stat['Characteristic']))
```

```
        corr_list=Base_Feature_Select_Method.eliminate_high_corr(data_original,crete_data_stat, [], corr)
        exclue_dic['Corr'] = list(set(data_stat['Characteristic']) - set(exclue_list) - set(corr_list) - set(discrete_data_stat['Characteristic']))
        return corr_list+list(discrete_data_stat['Characteristic']),exclue_dic
    final_list=list(crete_data_stat['Characteristic'])+list(discrete_data_stat['Characteristic'])
    return final_list,exclue_dic
@staticmethod
def further_feature_select(flag_name,not_var_list,train_data_stat,train_data_woe,
                           C=3,selection_threshold=0.1,random_state=0,p_value=0.05,rlr_rule=True,coef_rule=True,pvalue_rule=True):
    """
    Explanation
    ----------
    根据评价指标阈值，进一步筛选出符合条件的指标
    """
    final_list = list(set(train_data_stat['Characteristic'])-set(not_var_list))
    if rlr_rule:
        final_list = Base_Feature_Select_Method.eliminate_instabe_by_RLR(train_data_woe[final_list + [flag_name]],
                                                                         not_var_list, C, selection_threshold,
                                                                         random_state, flag_name)
    if coef_rule:
        final_list = Base_Feature_Select_Method.eliminate_positive_coefficient(train_data_woe[final_list + [flag_name]],
                                                                               train_data_stat[
                                                                                   train_data_stat['Characteristic'].isin(
                                                                                       final_list)], flag_name)
    if pvalue_rule:
        final_list = Base_Feature_Select_Method.eliminate_high_pvalue_backword(train_data_woe[final_list + [flag_name]],
                                                                               flag_name, p_value)
    return final_list
```