

构建标准化建模代码（下）





课程目录

ONTENTS

1 变量分箱

2 变量评价指标计算和woe编码

3 变量筛选

4 模型训练

5 模型评估

6 模型结果标准化输出



模型训练



```

47 class Standard_Model_Building(Base_Feature_Analyse):
48
49     def __init__(self, flag_name, data=pandas.DataFrame(), reject_name='', data_reject=pandas.DataFrame(), data_oot=pandas.DataFrame(), train_rate=0.6,
50                 bad_name='#Bad', good_name='#Good', piece=5, rate=0.05,
51                 min_bin_size=50, not_in_list=['None', 'NaN', 'NA', 'nan', 'None'], not_var_list=[], flag_var_list=[], cut_method='cut_ks',
52                 combine_method='combine_iv', bin_thought='Up-Bottom',
53                 iv=0.02, ks=None, mic=None, single=0.95, min_bin_num=2, ar=None, corr_p=None, corr_s=None, ptest=None, ltest=None, ftest=None,
54                 C=3, selection_threshold=0.1, random_state=0, corr_rate=0.8, p_value=0.05, requirement='offline', train_name='Train',
55                 test_name='Test', oot_name='Oot', nprocess=4, output_path='', output_name='', base_odds=50, base_score=500, add_odds=2, add_score=20,
56                 var_description=pandas.DataFrame(), ks_decline_rate=0.1, reject_correct_rate=0.7, model_iter_max_time=6, ks_min=0.3, ks_cut=False,
57                 score_bin_method='best_ks', discrete_list=[], single_bin_value=[], no_control_direction=[]):
58
59     """
60     Explanation
61     传入数据集，获得最终模型结果、以及所有中间结果
62
63     Parameters
64
65     flag_name: string
66     | 标签名称
67
68     data: dataframe
69     | 建模样本集
70
71     reject_name: string
72     | 拒绝列名称
73
74     data_reject: dataframe, default=pandas.DataFrame()
75     | 拒绝样本集
76
77     data_oot: dataframe, default=pandas.DataFrame()
78     | 外验证样本集
79
80     train_rate: float, default=0.6
81     | 训练集占比
82
83     bad_name: string, default='bad'
84     | 坏标签列名
85
86     good_name: string, default='good'
87     | 好标签列名
88

```

```
89 piece: int, default=5
90 | 分组最大数量
91
92 rate: float, default=0.05
93 | 分组每组样本最小占比
94
95 min_bin_size: int, default=50
96 | 分组每组样本最小数量
97
98 not_in_list: list, default=['None', 'NaN', 'NA', 'nan', None]
99 | 空值列表
100
101 not_var_list: list
102 | 不需要计算的列名，比如姓名、手机号码、指标时间、身份证号码等
103
104 flag_var_list: list
105 | 不需要计算，但是需要进入模型的列名，比如y标签
106
107 cut_method: string, default='cut_ks'
108 | 分bin的方法
109
110 combine_method: string, default='combine_iv'
111 | 聚合bin的方法
112
113 bin_thought: string, default='Up-Bottom'
114 | 分bin的思想
115
116 iv: float, default=0.02
117 | iv阈值
118
119 ks: float, default=None
120 | ks阈值
121
122 mic: float, default=None
123 | 最大互信息数阈值
124
125 single: float, default=0.85
126 | 单一阈值阈值
127
128 min_bin_num: int, default=2
129 | 分组最小数量，除了缺失值以外
130
131 ar: float, default=None
132 | ar阈值
```

```
133     corr_p: float, default=None  
134     | 皮尔逊相关系数阈值  
135  
136  
137     corr_s: float, default=None  
138     | 斯皮尔曼相关系数阈值  
139  
140     pctest: float, default=None  
141     | 皮尔逊卡方检验p-value阈值  
142  
143     ltest: float, default=None  
144     | 似然比检验p-value阈值  
145  
146     fctest: float, default=None  
147     | f检验p-value阈值  
148  
149     c: int, default=3  
150     | 变量筛选的惩罚系数  
151  
152     selection_threshold: float, default=0.1  
153     | 变量筛选阈值  
154  
155     random_state: int, default=0  
156     | 随机建模种子  
157  
158     corr_rate: float, default=0.8  
159     | 相关系数阈值  
160  
161     p_value: float, default=0.05  
162     | pvalue阈值  
163  
164     requirement: string, default='offline'  
165     | 线上计算或者线下计算  
166  
167     train_name: string  
168     | 训练集名称  
169  
170     test_name: string  
171     | 测试集名称  
172  
173     oot_name: string  
174     | 外验证集名称  
175  
176     nprocess: int, default=4  
177     | 多进程数量, 用于控制变量分bin的进程数  
178
```

```
178 output_path:string, default=''
179 | 最终模型结果保存的路径
180
181 output_name: string, default=''
182 | 结果保存文件夹名称
183
184 bad_odds:float, default=50
185 | 基准好坏比
186
187 base_score:int, default=500
188 | 基准分
189
190 add_odds:intfloat, default=2
191 | odds变化步长
192
193 add_score:int, default=20
194 | odds变化一个步长, 相应增长的分数
195
196 var_description: dataframe, default=pandas.DataFrame()
197 | 指标含义
198
199 ks_decline_rate: float, default=0.1
200 | abs(训练集ks-测试集ks)/ 训练集ks 的上限, 满足则模型通过, 否则继续重新切分训练集、测试集建模
201
202 reject_correct_rate: float, default=0.7
203 | 本次拒绝推断预测的标签与上一次拒绝推断标签对比, 正确率大于reject_correct_rate则拒绝推断完成, 否则继续迭代
204
205 model_iter_max_time: int,default=6
206 | 模型迭代最大次数, 如果超过该限制, 训练集ks与测试集ks还是达不到效果, 强制结束, 以最后一次作为模型结果
207
208 ks_min: float,default=0.3
209 | ks达到的最小值
210
211 ks_cut: boolean,default=False
212 | 是否启用ks最小值循环
213
214 score_bin_method: string,default='best_ks'
215 | 模型得分的分组方法
216
217 discrete_list: list,default=[]
218 | 指定离散变量的名称list
219
220 single_bin_value: dict,default={}
221 | 指定需要单独分为一组的指标取值(不包括None, 因为缺失值默认单独分为一组)
222
```

```
223 no_control_direction: list,default=[]
224 | 指定不需要对分组bad_rate方向进行控制的变量, 否则只会分出正向、负向、正U向、负U向四种类型
225 """
226
227 if flag_name not in flag_var_list:
228     flag_var_list.append(flag_name)
229 if reject_name and reject_name not in flag_var_list:
230     flag_var_list.append(reject_name)
231 not_var_list.extend([flag_name,reject_name])
232 Base_Feature_Analyse.__init__(self,flag_name, data,data_oot,train_rate,bad_name,
233     good_name, piece, rate, min_bin_size, not_in_list,
234     not_var_list, flag_var_list,cut_method,combine_method,bin_thought,
235     train_name,test_name,oot_name,nprocess,requirement,discrete_list,single_bin_value,no_control_direction)
236 #####
237 self.iv=iv
238 self.ks=ks
239 self.mic =mic
240 self.single =single
241 self.min_bin_num=min_bin_num
242 self.ar =ar
243 self.corr_p =corr_p
244 self.corr_s =corr_s
245 self.ptest =ptest
246 self.ltest =ltest
247 self.ftest =ftest
248 #####
249 self.base_odds=base_odds
250 self.base_score = base_score
251 self.add_odds=add_odds
252 self.add_score = add_score
253 self.var_description=self.check_var_description(var_description)
254 #####
255 self.reject_name=reject_name
256 self.data_reject=data_reject
257 #####
258 self.C=C
259 self.selection_threshold=selection_threshold
260 self.random_state =random_state
261 self.corr_rate =corr_rate
262 self.p_value =p_value
263 #####
264 self.output_path =output_path
265 self.output_name=output_name
266 #####
267 self.ks_decline_rate=ks_decline_rate
```

```

269 self.reject_correct_rate=reject_correct_rate
270 self.model_iter_max_time=model_iter_max_time
271 self.ks_min=ks_min
272 self.ks_cut=ks_cut
273 #####
274 self.score_bin_method=score_bin_method
275 #####
276 self.reject_len=len(self.data_reject)
277 self.data_total=copy.deepcopy(self.data)
278 self.iter_time=0
279
280 @staticmethod
281 def check_var_description(var_description):
282     """
283     Explanation
284     校验指标含义
285     """
286     if type(var_description) != pandas.core.frame.DataFrame:
287         return var_description
288     elif len(var_description) == 0 or len(var_description.columns) > 2:
289         return var_description
290     else:
291         index = list(var_description.columns)
292         tmp = var_description[index]
293         tmp['len1'] = list(map(lambda x: len(x), tmp[index[0]]))
294         tmp['len2'] = list(map(lambda x: len(x) if type(x)==str else 0, tmp[index[1]]))
295         if tmp['len1'].max() > tmp['len2'].max():
296             var_description.columns = ['Description', 'Characteristic']
297         else:
298             var_description.columns = ['Characteristic', 'Description']
299         var_description = var_description[['Characteristic', 'Description']]
300         if len(var_description[var_description['Characteristic'] == 'intercept']) == 0:
301             var_description.loc[len(var_description)] = ['intercept', u'常数项']
302         return var_description
303
304 @staticmethod
305 def str_to_unicode(x):
306     """
307     Explanation
308     将x转成unicode编码
309     """
310     try:
311         return x.decode('utf-8')

```

```

except:
    return x
314
315
316 @staticmethod
317 def get_logit_pre(result, train_data_woe, flag_name, final_list):
318     """
319     Explanation
320     -----
321     根据逻辑回归模型和样本，获得该样本的预测结果
322     """
323     if len(train_data_woe)==0:
324         return pandas.DataFrame([], columns=[flag_name, 'predict'])
325     train_pre=train_data_woe[[flag_name]]
326     train_pre['predict'] = result.predict(train_data_woe[final_list])
327     return train_pre
328
329 @staticmethod
330 def iteration_logistic(data_woe, flag_name, train_rate, final_list, ks_cut=False, ks_min=None, ks_decline_rate=0.1, iter=0, model_iter_max_time=6):
331     """
332     Explanation
333     -----
334     根据woe建模样本进行logistic模型迭代，直到训练集ks、测试集ks达到要求
335     一般要求如下：
336     (1) 训练集、测试集ks均要大于一个阈值，比如0.3，低于0.3视为模型无效
337     (2) abs(训练集ks-测试集ks) / 训练集ks 需要小于一个阈值，否则视为模型在测试集上表现不稳定，模型无效
338     """
339     train_data_woe, test_data_woe=Standard_Model_Building.offline_train_test(data_woe, flag_name, train_rate)
340     train_data_woe['intercept']=1
341     test_data_woe['intercept'] = 1
342     # The feature selection is done, start to modeling
343     logit = sm.Logit(train_data_woe[flag_name], train_data_woe[final_list])
344     result = logit.fit()
345     train_pre=Standard_Model_Building.get_logit_pre(result, train_data_woe, flag_name, final_list)
346     test_pre = Standard_Model_Building.get_logit_pre(result, test_data_woe, flag_name, final_list)
347     #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
348     train_ks=Feature_Evaluate_Method.KS(train_pre, flag_name, 'predict')[0]
349     test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
350     print('train_ks:', train_ks)
351     print('test_ks:', test_ks)
352     # plot the feature important
353     # import matplotlib.pyplot as plt
354     # plt.figure(figsize=(12, 8))
355     # coef = pandas.Series(result.feature_importances_, final_list).sort_values(ascending=False)
356     # new_coef = coef[coef > 0.001]
357     # new_coef.plot(kind='bar', title='Feature Importances')
358     # plt.show()

```

```

359 # plt.savefig('test.pdf')
360 # plt.savefig('test.pdf')
361 #####
362 if iter+1==model_iter_max_time:
363     print('The maximum number of iterations reached the model, and no expected model was found.')
```

364 return train_data_woe,x_test_data_woe,x_train_pre,x_test_pre,result

365 #

366 ks_prove=False

367 if ks_cut==True:

368 if train_ks>=ks_min or test_ks>=ks_min:

369 if abs(train_ks - test_ks) / max(train_ks, test_ks) <= ks_decline_rate:

370 return train_data_woe,x_test_data_woe,x_train_pre,x_test_pre,result

371 iter+=1

372 return Standard_Model_Building.iteration_logistic(data_woe,x_flag_name,x_train_rate,x_final_list,ks_cut,ks_min,ks_decline_rate,iter)

373 else:

374 if abs(train_ks - test_ks) / max(train_ks, test_ks) <= ks_decline_rate:

375 return train_data_woe, test_data_woe, train_pre,x_test_pre,result

376 iter += 1

377 return Standard_Model_Building.iteration_logistic(data_woe,x_flag_name,x_train_rate,x_final_list,ks_cut,ks_min,ks_decline_rate,iter)

378

379 @staticmethod

380 def iteration_RF(data_original,x_flag_name,x_train_rate,x_final_list,ks_cut=False,ks_min=None,ks_decline_rate=0.1,iter=0,model_iter_max_time=6):

381 """

382 Explanation

383 根据原始样本进行随机森林模型的最优参数网格搜索和迭代，直到训练集ks、测试集ks达到要求

384 一般要求如下：

385 (1) 训练集、测试集ks均要大于一个阈值，比如0.3，低于0.3视为模型无效

386 (2) abs(训练集ks-测试集ks) / 训练集ks 需要小于一个阈值，否则视为模型在测试集上表现不稳定，模型无效

387 调参规则：

388 1、先用默认参数看预测结果

389 2、然后用gridsearchcv探索n_estimators的最佳值：

390 param_test1 = {'n_estimators': list(range(10, 200, 10))}

391 3、然后确定n_estimators，据此再搜索另外两个参数：决策树最大深度max_depth和内部节点在划分所需最小样本数min_samples_split一起调参：

392 param_test2 = {'max_depth': list(range(3, 14, 2)), 'min_samples_split': list(range(50, 201, 20))}

393 4、然后再对内部节点再划分所需最小样本数min_samples_split和叶子节点最少样本数min_samples_leaf一起调参：

394 param_test3 = {'min_samples_split': list(range(80, 150, 20)), 'min_samples_leaf': list(range(10, 60, 10))}

395 5、最后我们再对最大特征数max_features做调参：

396 param_test4 = {'max_features': list(range(3, 11, 2))}

397 6、最后用得到的参数再次带入模型，得到结果。

398

399 """

400 train_data_original,x_test_data_original=Standard_Model_Building.offline_train_test(data_original, flag_name, train_rate)

401 x_train=train_data_original[final_list]

402 y_train=train_data_original[[flag_name]]

```
403 x_test=test_data_original[final_list]
404 y_test=test_data_original[[flag_name]]
405 train_pre=copy.deepcopy(y_train)
406 test_pre = copy.deepcopy(y_test)
407 # The feature selection is done,start to modeling
408 result=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
409                               max_depth=None, max_features=0.2, max_leaf_nodes=None,
410                               min_samples_leaf=1, min_samples_split=2,
411                               min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
412                               oob_score=True, random_state=None, verbose=0,
413                               warm_start=False)
414 #result = RandomForestClassifier(oob_score=True, random_state=10)
415 result.fit(x_train, y_train)
416 #
417 train_pre['predict']=result.predict_proba(x_train)[:,-1]
418 test_pre['predict']=result.predict_proba(x_test)[:,-1]
419 #oot_pre = Standard Model Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
420 train_ks=Feature_Evaluate_Method.KS(train_pre,flag_name,'predict')[0]
421 test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
422 print('train_ks:',train_ks)
423 print('test_ks:',test_ks)
424 if iter+1==model_iter_max_time:
425     print('The maximum number of iterations reached the model, and no expected model was found.')
426     return train_data_original,test_data_original,x_train_pre,x_test_pre,result
427 # 首先对n_estimators进行网格搜索
428 param_test1 = {'n_estimators': list(range(10, 200, 10))}
429 gsearch1 = GridSearchCV(estimator=RandomForestClassifier(min_samples_split=50,
430                                                         min_samples_leaf=20, max_depth=6, max_features=0.5,
431                                                         random_state=10),
432                        param_grid=param_test1, scoring='roc_auc', cv=5)
433 gsearch1.fit(x_train, y_train[[flag_name]])
434 #gsearch1.grid_scores_ gsearch1.best_params_ gsearch1.best_score_
435 best_n_estimators=gsearch1.best_params_['n_estimators']
436 # 对决策树最大深度max_depth的内部节点在划分所需最小样本数min_samples_split进行网格搜索
437 param_test2 = {'max_depth': list(range(3, 14, 2)), 'min_samples_split': list(range(50, 201, 20))}
438 gsearch2 = GridSearchCV(estimator=RandomForestClassifier(n_estimators=best_n_estimators, min_samples_leaf=20,
439                                                         max_features='sqrt', oob_score=True, random_state=10),
440                        param_grid=param_test2, scoring='roc_auc', cv=5)
441 gsearch2.fit(x_train, y_train[[flag_name]])
442 best_max_depth=gsearch2.best_params_['max_depth']
443 best_min_samples_split=gsearch2.best_params_['min_samples_split']
444 #gsearch2.grid_scores_ gsearch2.best_params_ gsearch2.best_score_
445 # 此时袋外分数有一些提高, 泛化能力增强了, 对于内部节点再划分所需最小样本数
446 # min_samples_split,暂时不能一起定下来, 因为这个和决策树其他参数存在关联, 再对
447 # 内部节点再划分所需最小样本数min_samples_split和叶子节点最少样本数min_samples_leaf一起调参
```

```

448 param_test3 = {'min_samples_split': list(range(80, 150)), 'min_samples_leaf': list(range(10, 60, 10))}
449 gsearch3 = GridSearchCV(estimator=RandomForestClassifier(n_estimators=best_n_estimators, max_depth=best_max_depth,
450                                                         max_features='sqrt', oob_score=True, random_state=10),
451                       param_grid=param_test3, scoring='roc_auc', iid=False, cv=5)
452 gsearch3.fit(x_train, y_train[flag_name])
453 best_min_samples_split = gsearch3.best_params_['min_samples_split']
454 best_min_samples_leaf = gsearch3.best_params_['min_samples_leaf']
455 #gsearch3.grid_scores_, gsearch3.best_params_, gsearch3.best_score_
456
457 # 最后对最大特征数max_features做调参
458 # GridSearchCV中param_grid参数是字典构成的列表
459 param_test4 = {'max_features': list(range(3, 11, 2))}
460 gsearch4 = GridSearchCV(estimator=RandomForestClassifier(n_estimators=best_n_estimators, max_depth=best_max_depth,
461                                                         min_samples_leaf=best_min_samples_leaf, min_samples_split=best_min_samples_split,
462                                                         oob_score=True, random_state=10),
463                       param_grid=param_test4, scoring='roc_auc', iid=False, cv=5)
464 gsearch4.fit(x_train, y_train[flag_name])
465 best_max_features = gsearch4.best_params_['max_features']
466 #gsearch4.grid_scores_, gsearch4.best_params_, gsearch4.best_score_
467
468 # 找到最佳参数, 看看最终模型
469 result = RandomForestClassifier(n_estimators=best_n_estimators, max_depth=best_max_depth
470                               , min_samples_leaf=best_min_samples_leaf, min_samples_split=best_min_samples_split,
471                               oob_score=True, random_state=10)
472
473 result.fit(y_train, x_train)
474 train_pre['predict']=result.predict_proba(x_train)[: ,1]
475 test_pre['predict']=result.predict_proba(x_test)[: ,1]
476 #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
477 train_ks=Feature_Evaluate_Method.KS(train_pre,flag_name,'predict')[0]
478 test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
479 #####
480 ks_prove=False
481 if ks_cut==True:
482     if train_ks>=ks_min or test_ks>=ks_min:
483         if abs(train_ks - test_ks) / max(train_ks, test_ks) <= ks_decline_rate:
484             return train_data_original,x_test_data_original,x_train_pre,x_test_pre,result
485         iter+=1
486     return Standard_Model_Building.iteration_logistic(data_original,x_flag_name,x_train_rate,x_final_list,x_ks_cut,ks_min,ks_decline_rate,iter)
487 else:
488     if abs(train_ks - test_ks) / max(train_ks, test_ks) <= ks_decline_rate:
489         return train_data_original, test_data_original, train_pre,x_test_pre,result
490     iter += 1
491     return Standard_Model_Building.iteration_logistic(data_original,x_flag_name,x_train_rate,x_final_list,x_ks_cut,ks_min,ks_decline_rate,iter)
492

```

```

493 @staticmethod
494 def iteration_lightgbm(data_original, flag_name, train_rate, final_list, ks_cut=False, ks_min=None, ks_decline_rate=0.1, iter=0, model_iter_max_time=6):
495     """
496     Explanation
497
498     根据原始样本进行GBDT模型的最优参数网格搜索和迭代，直到训练集ks、测试集ks达到要求
499     一般要求如下：
500     (1) 训练集、测试集ks均要大于一个阈值，比如0.3，低于0.3视为模型无效
501     (2) abs(训练集ks-测试集ks) / 训练集ks 需要小于一个阈值，否则视为模型在测试集上表现不稳定，模型无效
502     """
503     from lightgbm import LGBMClassifier
504     import lightgbm as lgb
505     from sklearn.metrics import roc_auc_score
506     #####
507     train_data_original, test_data_original = Standard_Model_Building.offline_train_test(data_original, flag_name, train_rate)
508     x_train = train_data_original[final_list]
509     y_train = train_data_original[[flag_name]]
510     x_test = test_data_original[final_list]
511     y_test = test_data_original[[flag_name]]
512     train_pre = copy.deepcopy(y_train)
513     test_pre = copy.deepcopy(y_test)
514     predictors = final_list
515     result = LGBMClassifier(objective='regression', num_leaves=31, learning_rate=0.05, n_estimators=20)
516     result.fit(x_train, y_train, eval_set=[(x_test, y_test)], eval_metric='l1', early_stopping_rounds=5)
517     #
518     train_pre['predict'] = result.predict(x_train)
519     test_pre['predict'] = result.predict(x_test)
520     #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
521     train_ks = Feature_Evaluate_Method.KS(train_pre, flag_name, 'predict')[0]
522     test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
523     print('train_ks:', train_ks)
524     print('test_ks:', test_ks)
525     if iter+1 == model_iter_max_time:
526         print('The maximum number of iterations reached the model, and no expected model was found.')
527         return train_data_original, test_data_original, train_pre, test_pre, result
528     #原生态Lightgbm
529     lgb_train = lgb.Dataset(x_train, y_train)
530     lgb_eval = lgb.Dataset(x_test, y_test, reference=lgb_train)
531
532     # 将参数写成字典下形式
533     params = {
534         'task': 'train',
535         'boosting_type': 'gbdt', # 设置提升类型
536         'objective': 'regression', # 目标函数
537         'metric': {'l2', 'auc'}, # 评估函数

```

```

538     'num_leaves': 31, # 叶子节点数
539     'learning_rate': 0.05, # 学习速率
540     'feature_fraction': 0.9, # 建树的特征选择比例
541     'bagging_fraction': 0.8, # 建树的样本采样比例
542     'bagging_freq': 5, # k 意味着每 k 次迭代执行bagging
543     'verbose': 1 # <0 显示致命的, =0 显示错误 (警告), >0 显示信息
544 }
545
546 print('Start training...')
547 # 训练 cv and train
548 gbm = lgb.train(params, lgb_train, num_boost_round=20, early_stopping_rounds=5)
549 train_pre['predict'] = gbm.predict(x_train)
550 test_pre['predict'] = gbm.predict(x_test)
551 #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
552 train_ks = Feature_Evaluate_Method.KS(train_pre, flag_name, 'predict')[0]
553 test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
554 print('train_ks:', train_ks)
555 print('test_ks:', test_ks)
556
557 print('Save model...')
558 # 保存模型到文件
559 gbm.save_model('model.txt')
560
561 print('Start predicting...')
562 # 预测数据集
563 y_pred = gbm.predict(x_test, num_iteration=gbm.best_iteration)
564
565 @staticmethod
566 def iteration_gbdtd(data_original, flag_name, train_rate, final_list, ks_cut=False, ks_min=None, ks_decline_rate=0.1, iter=0, model_iter_max_time=6):
567     """
568     Explanation
569     根据原始样本进行GBDT模型的最优参数网格搜索和迭代, 直到训练集ks、测试集ks达到要求
570     一般要求如下:
571     (1) 训练集、测试集ks均要大于一个阈值, 比如0.3, 低于0.3视为模型无效
572     (2) abs(训练集ks - 测试集ks) / 训练集ks 需要小于一个阈值, 否则视为模型在测试集上表现不稳定, 模型无效
573     """
574
575     from sklearn.ensemble import GradientBoostingClassifier
576     from sklearn import cross_validation, metrics
577     from sklearn.grid_search import GridSearchCV
578     import matplotlib.pyplot as plt
579     from matplotlib.pyplot import rcParams
580     #####
581     train_data_original, test_data_original = Standard_Model_Building.offline_train_test(data_original, flag_name, train_rate)
582     x_train = train_data_original[final_list]

```

```

584 y_train=train_data_original[[flag_name]]
585 x_test=test_data_original[final_list]
586 y_test=test_data_original[[flag_name]]
587 train_pre=copy.deepcopy(y_train)
588 test_pre = copy.deepcopy(y_test)
589 predictors = final_list
590 result = GradientBoostingClassifier(random_state=10)
591 result.fit(x_train,y_train)
592 #
593 train_pre['predict']=result.predict(x_train)
594 test_pre['predict']=result.predict(x_test)
595 #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
596 train_ks=Feature_Evaluate_Method.KS(train_pre,flag_name,'predict')[0]
597 test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
598 print('train_ks:',train_ks)
599 print('test_ks:',test_ks)
600 if iter+1==model_iter_max_time:
601     print('The maximum number of iterations reached the model, and no expected model was found.')
602     return train_data_original,test_data_original,train_pre,test_pre,result
603 #首先我们从步长(learning_rate)和迭代次数(n_estimators)入手。一般来说,开始选择一个较小的步长来网格搜索最好的迭代次数。
604 # 这里,我们将步长初始值设置为0.1。对于迭代次数进行网格搜索如下
605 param_test1 = {'n_estimators': range(20, 81, 10)}
606 gsearch1 = GridSearchCV(estimator=GradientBoostingClassifier(learning_rate=0.1, min_samples_split=300,
607     min_samples_leaf=20, max_depth=8,
608     max_features='sqrt', subsample=0.8,
609     random_state=10),
610     param_grid=param_test1, scoring='roc_auc', iid=False, cv=5)
611 gsearch1.fit(x_train, y_train[flag_name])
612 #gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
613
614 #找到了一个合适的迭代次数。现在我们开始对决策树进行调参。首先我们对决策树最大深度max_depth和内部节点再划分所需最小样本数min_samples_split进行网格搜索
615 param_test2 = {'max_depth': range(3, 14, 2), 'min_samples_split': range(100, 801, 200)}
616 gsearch2 = GridSearchCV(
617     estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, min_samples_leaf=20,
618     max_features='sqrt', subsample=0.8, random_state=10),
619     param_grid=param_test2, scoring='roc_auc', iid=False, cv=5)
620 gsearch2.fit(x_train, y_train[flag_name])
621 #gsearch2.grid_scores_, gsearch2.best_params_, gsearch2.best_score_
622
623 #对于内部节点再划分所需最小样本数min_samples_split,我们暂时不能一起定下来,因为这个还和决策树其他的参数存在关联。
624 # 下面我们再对内部节点再划分所需最小样本数min_samples_split和叶子节点最少样本数min_samples_leaf一起调参
625 param_test3 = {'min_samples_split': range(800, 1900, 200), 'min_samples_leaf': range(60, 101, 10)}
626 gsearch3 = GridSearchCV(estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, max_depth=7,
627     max_features='sqrt', subsample=0.8,
628     random_state=10),

```

```

628         param_grid=param_test3, scoring='roc_auc', iid=False, cv=5)
629 gsearch3.fit(x_train, y_train[flag_name])
630 #gsearch3.grid_scores_, gsearch3.best_params_, gsearch3.best_score_
631
632 #现在我们对最大特征数max_features进行网格搜索
633 param_test4 = {'max_features': range(7, 20, 2)}
634 gsearch4 = GridSearchCV(
635     estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, max_depth=7, min_samples_leaf=60,
636                                         min_samples_split=1200, subsample=0.8, random_state=10),
637     param_grid=param_test4, scoring='roc_auc', iid=False, cv=5)
638 gsearch4.fit(x_train, y_train[flag_name])
639 #gsearch4.grid_scores_, gsearch4.best_params_, gsearch4.best_score_
640
641 #再对子采样的比例进行网格搜索
642 param_test5 = {'subsample': [0.6, 0.7, 0.75, 0.8, 0.85, 0.9]}
643 gsearch5 = GridSearchCV(
644     estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, max_depth=7, min_samples_leaf=60,
645                                         min_samples_split=1200, max_features=9, random_state=10),
646     param_grid=param_test5, scoring='roc_auc', iid=False, cv=5)
647 gsearch5.fit(x_train, y_train[flag_name])
648 #gsearch5.grid_scores_, gsearch5.best_params_, gsearch5.best_score_
649
650 #最后再对学习速率和最大迭代次数进行网格搜索
651 param_test6 = {'learning_rate': [0.01, 0.02, 0.05, 0.1, 0.1, 0.2], 'n_estimators': list(range(20, 600, 20))}
652 gsearch6 = GridSearchCV(
653     estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, max_depth=7, min_samples_leaf=60,
654                                         min_samples_split=1200, max_features=9, random_state=10),
655     param_grid=param_test5, scoring='roc_auc', iid=False, cv=5)
656 gsearch6.fit(x_train, y_train[flag_name])
657 #gsearch6.grid_scores_, gsearch6.best_params_, gsearch6.best_score_
658
659 def standard_logistic_model(self):
660     """
661     Explanation
662     模型最终结果
663     """
664     t1=time.time()
665     train_name = self.str_to_unicode(self.train_name)
666     test_name = self.str_to_unicode(self.test_name)
667     oot_name = self.str_to_unicode(self.oot_name)
668     # feature analyse and woe
669     if self.requirement=='online':
670         train_data_bin, data_woe, oot_data_woe, train_data_stat, oot_data_stat, data_original, oot_data_original = self.trans_woe_multiprocess(
671             self.data_total,

```

```
674         pandas.DataFrame(),
675         self.data_out,
676         self.train_name,
677         self.test_name,
678         self.oot_name)
679     else:
680         if self.requirement=='multiprocess':
681             sk=self.trans_woe_multiprocess
682         else:
683             sk=self.trans_woe_no_multiprocess
684         train_data_bin, data_woe, _, oot_data_woe, train_data_stat, _, oot_data_stat, data_original, _, oot_data_original = sk(
685             self.data_total,
686             pandas.DataFrame(),
687             self.data_out,
688             self.train_name,
689             self.test_name,
690             self.oot_name)
691     #feature_select
692     initial_feature_list,delete_dict=Base_Feature_Select_Rule.initial_feature_select(data_original,train_data_stat,self.discrete_list,self.single,self.iv,self
693     furture_feature_list = Base_Feature_Select_Rule.further_feature_select(self.flag_name, self.not_var_list,
694                                     train_data_stat[train_data_stat[
695                                         'Characteristic'].isin(
696                                             initial_feature_list)],
697                                     data_woe[[self.flag_name] + initial feature list],
698                                     self.C, self.selection_threshold,
699                                     self.random_state, self.p_value)
700     print(delete_dict)
701     final_list=furture_feature_list+['intercept']
702     # Model interation
703     train_data_woe, test_data_woe, train_pre,test_pre,result = Standard_Model_Building.iteration_logistic(data_woe, self.flag_name,
704                                                         self.train_rate, final_list,
705                                                         self.ks_cut, self.ks_min,
706                                                         self.ks_decline_rate)
707     # oot and original data
708     oot_data_woe['intercept']=1
709     oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, self.flag_name, final_list)
710     train_data_original=data_original.loc[train_data_woe.index]
711     test_data_original=data_original.loc[test_data_woe.index]
712     # model evaluation
713     flag_list = [self.flag_name]
714     if not self.reject_name:
715         print('Start without Reject Inference')
716         tmp_pre = Summary_Of_Model_Result.model_score(train_pre, 'predict',train_name,'Logistic')
717         if self.score_bin_method=='best_ks':
718             score_bin = self.Best_Bin(self.flag_name, '%s_score' % train_name, tmp_pre, piece=10,rate=0.02,bin_thought=self.bin_thought)
```

```
719         score_bin=Summary_Of_Model_Result.get_10_percentile(tmp_pre,self.flag_name,x=train_name,num=20)
720         if len(score_bin)==0:
721             score_bin = self.Best_Bin(self.flag_name, '%s_score' % train_name, tmp_pre, piece=10)
722     train_data_stat.insert(1,'Sample','Development Sample')
723     if len(oot_data_stat)>0:
724         oot_data_stat.insert(1, 'Sample', oot_name)
725     # combine result
726     Summary_Of_Model_Result.combine_model_result(self.flag_name,self.not_var_list,x=self.reject_name,self.var_description,score_bin, train_data_bin, train_data_stat,
727         oot_data_stat, train_data_original, test_data_original, oot_data_original,
728         self.output_path,self.output_name,x=train_name,test_name, oot_name,self.not_in_list,'Logistic',result,x=train_data_woe,test_data_woe,oot_data_woe)
729     print(time.time() - t1)
730 else:
731     print('Start with Reject Inference')
732     flag_list.append(self.reject_name)
733     agb_data_woe=data_woe
734     agb_data_original=data_original
735     kgb_data_woe=agb_data_woe[agb_data_woe[self.reject_name]==0]
736     kgb_data_original=agb_data_original[agb_data_original[self.reject_name]==0]
737     #pre
738     agb_pre = pandas.concat([train_pre, test_pre])
739     kgb_pre=Standard_Model_Building.get_logit_pre(result,kgb_data_woe,self.flag_name,final_list)
740     #stat
741     agb_stat=train_data_stat
742     kgb_stat = [self.get_data_stat_else(kgb_data_original, kgb_data_woe, self.flag_name, var,
743         train_data_bin[train_data_bin['Characteristic'] == var][
744             'Char_Type'].max()) for var in future_feature_list]
745     kgb_stat = pandas.DataFrame(kgb_stat)
746     agb_stat.insert(1,'Sample','Development - AGB')
747     kgb_stat.insert(1, 'Sample', 'Development - KGB')
748     if len(oot_data_stat)>0:
749         oot_data_stat.insert(1, 'Sample', oot_name)
750     tmp_pre = Summary_Of_Model_Result.model_score(agb_pre, 'predict','Development - AGB','Logistic')
751     if self.score_bin_method=='best_ks':
752         score_bin = self.Best_Bin(self.flag_name, 'Development - AGB_score', tmp_pre, piece=10)
753     else:
754         score_bin=Summary_Of_Model_Result.get_10_percentile(tmp_pre,self.flag_name,'Development - AGB_score',num=20)
755     # combine result
756     Summary_Of_Model_Result.combine_model_result(self.flag_name,self.not_var_list,x=self.reject_name, self.var_description,score_bin,train_data_bin, agb_stat,
757         kgb_stat,
758         oot_data_stat, agb_data_original, kgb_data_original, oot_data_original,
759         self.output_path,self.output_name, 'Development - AGB', 'Development - KGB', oot_name, self.not_in_list,
760         'Logistic',result,x=agb_data_woe,kgb_data_woe,oot_data_woe)
```

```
766 if name == 'main':
767     data = pandas.read_excel('E:/zhangxianda/标准化建模工具/standard_model_tool_20180320_V08/test_data/model_sample.xlsx')
768     flag_name='lights'
769     a4 = Standard_Model_Building(flag_name=flag_name, data=data, not_var_list=[flag_name],
770                                 flag_var_list=[flag_name], data_reject=pandas.DataFrame(), reject_name='',
771                                 data_oot=pandas.DataFrame(), oot_name='Validation', var_description=pandas.DataFrame(),
772                                 output_name='u', ks_min=0.1, ks_cut=True, score_bin_method='best_ks', requirement='no-mu', discrete_list=[])
773     #a4.reject_inference()
774     a4.standard_logistic_model()
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
```



模型评估



图文类

1. ROC图
2. KS图
3. PR-F1图
4. 模型分数分布图
5.



表格类

1. 训练集、测试集、外验证集的KS、ROC、GINI、AR、F1、Precision、Recall等统计
2. 模型分数分布表
3. PSI统计表(模型分数&变量)
4. Swap set
5.

① 模型评估-画图

```
48 class Model_Plot(object):
49
50     def __init__(self):
51         pass
52
53     @staticmethod
54     def get_roc(train_plt, key, x_label='False_Positive_Rate', y_label='True_Positive_Rate'):
55         """
56         Explanation
57         -----
58         计算画roc需要的灵敏度和1-特异度
59
60         Parameters
61         -----
62         train_plt:pandas.core.frame.DataFrame
63         | 训练集预测值，分组统计后的结果
64
65         key:string
66         | 样本集名称
67
68         x_label:string
69         | 灵敏度英文名称
70
71         y_label:string
72         | 1-特异度英文名称
73
74         Return
75         -----
76         final_data:pandas.core.frame.DataFrame
77         | roc曲线画图需要的数据，包括三列：样本名称、灵敏度、1-特异度
78         """
79         x_data=1-numpy.cumsum(train_plt['good']/float(sum(train_plt['good'])))
80         y_data =1-numpy.cumsum(train_plt['bad']/float(sum(train_plt['bad'])))
81         final_data=pandas.DataFrame([x_data,y_data]).T
82         final_data.columns=[x_label,y_label]
83         final_data['type']=key
84         return final_data
85
86     @staticmethod
87     def plot_ROC(train_pre, test_pre, oot_pre, flag_name, train_name, test_name, oot_name, image_path, x_label='False_Positive_Rate', y_label='True_Positive_Rate'):
88         """
89         Explanation
```

```
87 @staticmethod
88 def plot_ROC(train_pre, test_pre, oot_pre, flag_name, train_name, test_name, oot_name, image_path, x_label='False Positive Rate', y_label='True Positive Rate'):
89     """
90     Explanation
91     将训练集、测试集、外验证集（如果有）画到一张roc图上，并保存到指定路径
92
93     Parameters
94
95     train_pre:pandas.core.frame.DataFrame
96     | 训练集预测结果
97
98     test_pre:pandas.core.frame.DataFrame
99     | 测试集预测结果
100
101     oot_pre:pandas.core.frame.DataFrame
102     | 外验证集预测结果
103
104     flag_name:string
105     | 标签名称
106
107     train_name:string
108     | 训练集名称
109
110     test_name:string
111     | 测试集名称
112
113     oot_name:string
114     | 外验证集名称
115
116     image_path:string
117     | 图保存路径
118
119     x_label:string
120     | 灵敏度英文名称
121
122     y_label:string
123     | 1-特异度英文名称
124     """
125     train_plt = Best_Bin_Cut.group_by_df(train_pre, flag_name, 'predict', 'bad', 'good')
126     test_plt = Best_Bin_Cut.group_by_df(test_pre, flag_name, 'predict', 'bad', 'good')
127     train_roc=Model_Plot.get_roc(train_plt, train_name, x_label=x_label, y_label=y_label)
128     test_roc = Model_Plot.get_roc(test_plt, test_name, x_label=x_label, y_label=y_label)
129     train_random=train_roc[x_label]
130     train_random[y_label]=train_roc[x_label]
```

```

131 train_random['type'] = 'random'
132 final_roc=pandas.concat([train_roc,test_roc,train_random])
133 train_value=roc_auc_score(train_pre[flag_name],train_pre['predict'])
134 test_value = roc_auc_score(test_pre[flag_name], test_pre['predict'])
135 title="Receiver Operating Characteristic Curve\n%s_ROC=%.2f , %s_ROC=%.2f"%(train_name,train_value,test_name,test_value)
136 if len(oot_pre)>0:
137     oot_plt = Best_Bin_Cut.group_by_df(oot_pre, flag_name, 'predict', 'bad', 'good')
138     oot_roc = Model_Plot.get_roc(oot_plt, oot_name,x_label='False_Positive_Rate',y_label='True_Positive_Rate')
139     final_roc=pandas.concat([final_roc,oot_roc])
140     oot_value = roc_auc_score(oot_pre[flag_name], oot_pre['predict'])
141     title=title+', %s_ROC=%.2f'%(oot_name,oot_value)
142 fig = plt.figure(figsize=(12,7))
143 for i in final_roc.groupby('type'):
144     ax1 = fig.add_subplot(111)
145     ax1.patch.set_facecolor("gainsboro")
146     ax1.plot(i[1]['False_Positive_Rate'],i[1]['True_Positive_Rate'],label=i[0])
147     ax1.legend(loc='upper left')
148     #plt.plot(i[1]['False_Positive_Rate'],i[1]['True_Positive_Rate'])
149 plt.xlabel(x_label)
150 plt.ylabel(y_label)
151 plt.title(title)
152 plt.rcParams['savefig.facecolor'] = 'whitesmoke'
153 plt.savefig(image_path+'ROC.png')
154

```

@staticmethod

```
def plot_KS(train_pre,flag_name,key,image_path):
```

Explanation

画出样本集的KS图，并保存到指定路径

Parameters

train_pre:pandas.core.frame.DataFrame

样本集预测结果

flag_name:string

标签名称

key:string

样本集名称

image_path:string

图保存路径

"""

② 模型评估-表格统计

```
17 class Summary_Of_Model_Result(object):
18
19     @staticmethod
20     def model_information(train_pre, predict_name, flag_name, sample_type=''):
21         """
22         Parameters
23         -----
24         train_pre:pandas.core.frame.DataFrame
25             样本预测结果
26
27         data:pandas.core.frame.DataFrame
28             样本集
29
30         flag_name:string
31             标签列名
32
33         sample_type:float, default=''
34             是否输入样本集的名称
35         """
36         if len(train_pre) == 0:
37             return pandas.DataFrame()
38         train_fpr, train_tpr, _ = roc_curve(train_pre[flag_name], train_pre[predict_name])
39         train_roc_auc = auc(train_fpr, train_tpr)
40         train_AR = 2 * train_roc_auc - 1
41         train_sample = copy.deepcopy(train_pre)
42         ks = Feature_Evaluate_Method.KS(train_sample, flag_name, predict_name)
43         ks_value = float(ks[1])
44         train_sample['predict_flag'] = list(map(lambda x: 1 if x > ks_value else 0, train_sample[predict_name]))
45         train_sample['key'] = 1
46         stack = train_sample.groupby([flag_name, 'predict_flag']).count().unstack()
47         information = {}
48         information['Sample'] = sample_type
49         information = pandas.DataFrame([information])
50         information['KS'] = ks[0]
51         information['AR'] = train_AR
52         information['AUC'] = train_roc_auc
53         information['GINI'] = train_AR
54         information['F1_Score'] = f1_score(train_sample[flag_name], train_sample['predict_flag'])
55         information['Recall_Score'] = recall_score(train_sample[flag_name], train_sample['predict_flag'])
56         information['Precision_Score'] = precision_score(train_sample[flag_name], train_sample['predict_flag'])
57         information['Odds_Ratio'] = float(stack.iloc[0, 0] * stack.iloc[1, 1]) / (stack.iloc[1, 0] * stack.iloc[0, 1])
58         columns_order = ['Sample', 'KS', 'AR', 'AUC', 'GINI', 'F1_Score', 'Recall_Score', 'Precision_Score',
```

```

60         'Odds_Ratio']
61     information = information[columns_order]
62     return information
63
64 @staticmethod
65 def model_vif(data, final_list, key=''):
66     """
67     Explanation
68     计算样本集指标的VIF
69
70     Parameters
71
72     data:pandas.core.frame.DataFrame
73     | 样本集
74
75     final_list:list
76     | 最终筛选出的变量列表，不包含常数项
77
78     key:float, default=''
79     | 是否输入样本集的名称
80
81     Return
82
83     vif:pandas.core.frame.DataFrame
84     | 返回样本集指标的VIF
85     """
86     if len(data) == 0:
87         return pandas.DataFrame()
88     vif_dic = []
89     for i in range(len(final_list)):
90         dic = {}
91         try:
92             vif = variance_inflation_factor(data[final_list].as_matrix(), final_list.index(final_list[i]))
93         except:
94             vif='SVD did not converge'
95             dic['Characteristic'] = final_list[i]
96             dic[key + 'VIF'] = vif
97             vif_dic.append(dic)
98     vif = pandas.DataFrame(vif_dic)
99     vif = vif[['Characteristic', key + 'VIF']]
100     return vif
101
102 @staticmethod
103 def model_psi(data, data_opposite, final_list, key=''):

```

```
104 """
105 Explanation
106
107 样本集相对于训练集的指标 $\psi_i$ 
108
109 Parameters
110
111 data: pandas.core.frame.DataFrame
112 | 训练集样本
113
114 data_opposite: pandas.core.frame.DataFrame
115 | 需要对比的样本集
116
117 final_list: list
118 | 最终筛选出的变量列表, 不包含常数项
119
120 key: float, default=''
121 | 是否输入样本集的名称
122
123 Return
124
125 final_psi: pandas.core.frame.DataFrame
126 | 样本集相对于训练集的指标 $\psi_i$ 
127 """
128 if len(data_opposite) == 0:
129     return pandas.DataFrame(columns=['Characteristic'])
130 data['key'] = 0
131 data_opposite['key'] = 1
132 data_total = pandas.concat([data[final_list + ['key']], data_opposite[final_list + ['key']]])
133 psi = []
134 for i in final_list:
135     psi_sub = Feature_Evaluate_Method.IV(data_total, 'key', i)
136     psi.append({'Characteristic': i, key + 'PSI': psi_sub})
137 final_psi = pandas.DataFrame(psi)[['Characteristic', key + 'PSI']]
138 return final_psi
```

```
140 @staticmethod
141 def model_score_describe(score_bin, flag_name, not_in_list, train_pre, train_name='Train'):
142     """
```

```
143 Explanation
144
145 计算出样本集的模型得分分布情况
```

```
147 Parameters
```

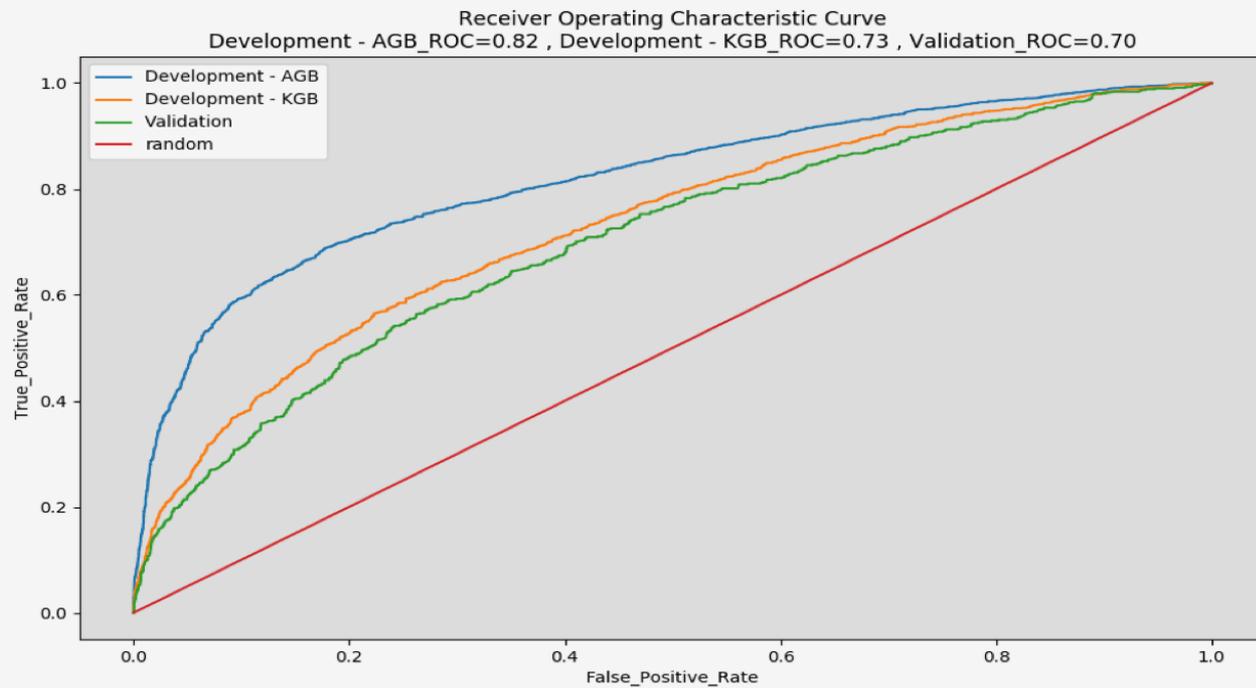
```
148 score_bin: float, default=0.1
```

```

150 | score_bin:pandas.core.frame.DataFrame
151 |     训练集模型得分最优分bin
152 |
153 | flag_name:string
154 |     标签列名
155 |
156 | not_in_list:list
157 |     空值列表
158 |
159 | train_pre:pandas.core.frame.DataFrame
160 |     训练集模型得分
161 |
162 | test_pre:pandas.core.frame.DataFrame, default=pandas.DataFrame()
163 |     测试集模型得分
164 |
165 | oot_pre:pandas.core.frame.DataFrame, default=pandas.DataFrame()
166 |     外验证集模型得分
167 |
168 | Return
169 |
170 | final_score:pandas.core.frame.DataFrame
171 |     计算出样本集的模型得分分布情况
172 |
173 | """
174 | if len(train_pre) == 0:
175 |     return pandas.DataFrame()
176 | bin_dic = Base_Feature_Analyse.get_bin_dic(score_bin)
177 | char_type = score_bin['Char_Type'].max()
178 | train_woe = copy.deepcopy(train_pre)
179 | train_woe['%s_score' % train_name] = list(map(
180 |     lambda x: Base_Feature_Analyse.get_var_woe(x, bin_dic, not_in_list, char_type, x, x[]),
181 |     train_pre['%s_score' % train_name]))
182 | train_score = Best_Bin_Cut.group_by_woe(train_woe, train_pre, score_bin, flag_name, '%s_score' % train_name,
183 |     '#Bad', '#Good', if_score=True)
184 |
185 | final_score = train_score
186 | del final_score['WOE']
187 | max_ks = Feature_Evaluate_Method.KS(train_pre, flag_name, 'predict')[0]
188 | t_bad_rate = final_score.iloc[len(final_score) - 1]['%Cumulative_Bad_Rate']
189 | final_score[['Bin', '#Obs', '#Good', '#Bad', '%Obs',
190 | '%Bad', '%Cumulative_Bad', '%Cumulative_Good', '%Bad_Rate',
191 | '%Cumulative_Bad_Rate', 'KS']]
192 | final_score.loc[len(final_score)] = ['Total', final_score['#Obs'].sum(), final_score['#Good'].sum(),
193 |     final_score['#Bad'].sum(), final_score['%Obs'].sum(),
194 |     final_score['%Bad'].sum(), 1.0, 1.0,
195 |     t_bad_rate, t_bad_rate, max_ks]

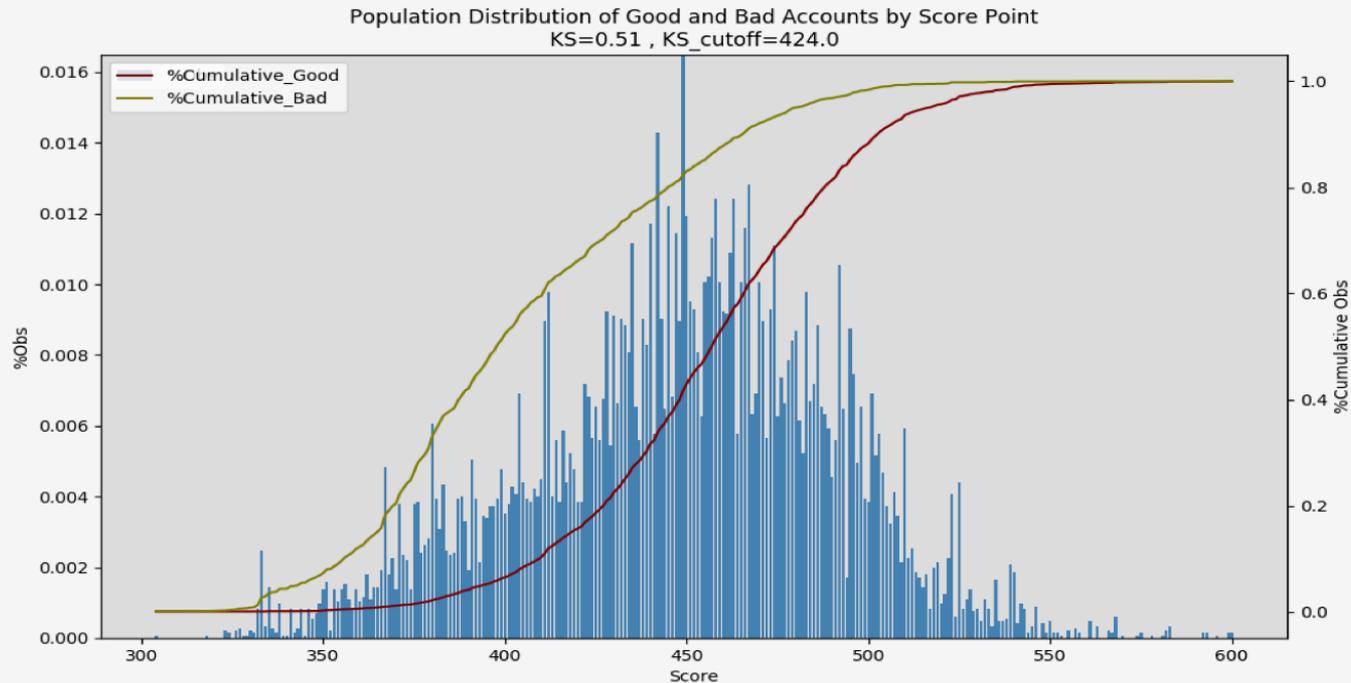
```

ROC Curve



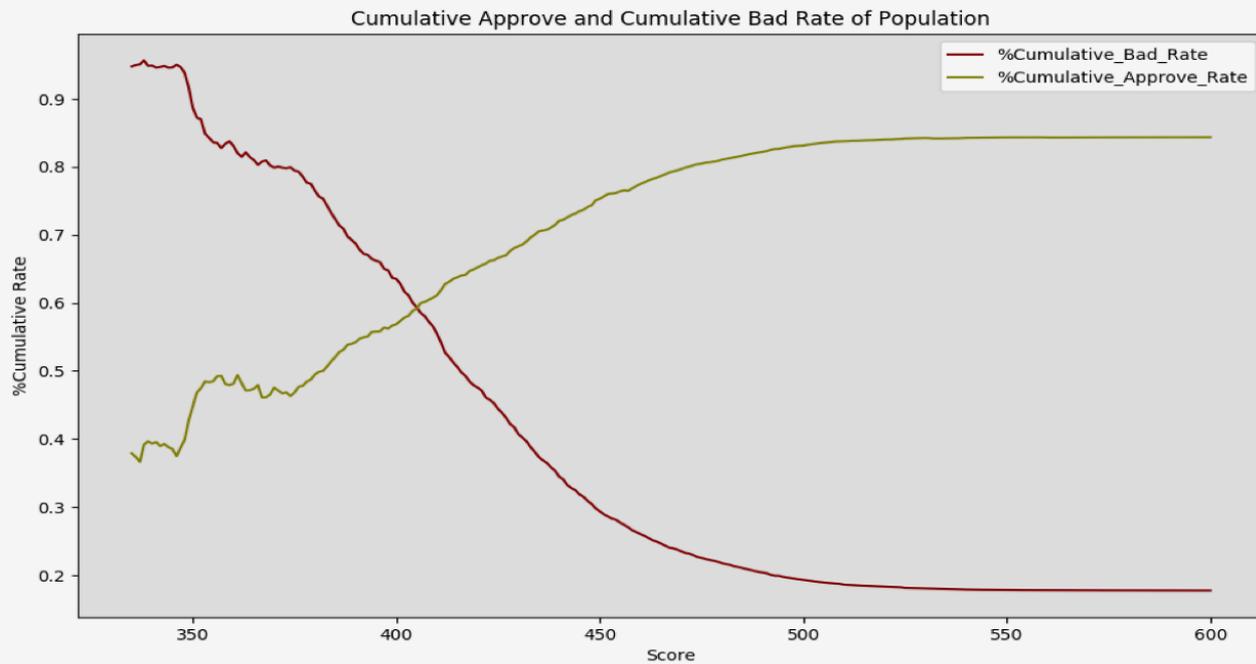
Ascending Score Report

a. Population Distribution of Good and Bad Accounts by Score Point



Ascending Score Report

b. Cumulative Approval and Cumulative Bad Rate of Population



Model Effect Statistics

Sample	KS	AR	AUC	GINI	F1_Score	Recall_Score	Precision_Score	Odds_Ratio
Development - AGB	0.51	0.64	0.82	0.64	0.55	0.69	0.45	10.22
Development - KGB	0.34	0.46	0.73	0.46	0.36	0.57	0.27	4.52
Validation	0.30	0.40	0.70	0.40	0.33	0.56	0.24	3.63

Model Swap Set

a. Num of Obfuscation Matrix

Swap Set (#)	Actual_Accept	Actual_Reject	Total
Predict_Accept	10681	1253	11934
Predict_Reject	1545	1018	2563
Total	12226	2271	14497

b. Rate of Obfuscation Matrix

Swap Set (%)	Actual_Accept	Actual_Reject	Total
Predict_Accept	73.68%	8.64%	82.32%
Predict_Reject	10.66%	7.02%	17.68%
Total	84.33%	15.67%	100.00%

c. Effect Promotion

Promotion	Old_Screening_Process	New_Screening_Process	Improvement_Bad_rate
Bad Rate	12.67%	8.99%	29.03%

Score Distribution

Score Scaling:

The scorecard is scaled so that every 20 points doubles the odds of an applicant in the development sample being a good account.

Thus, an applicant scoring 350 is approximately twice as likely to be good as an applicant scoring 330.

In addition, the scorecards are scaled such that, assuming minimal overriding, at score of 500, odds of 50:1 are likely to be achieved.

Score Distribution of All Applications

Bin	#Obs	#Good	#Bad	%Obs	%Bad	%Cumulative_Bad	%Bad_Rate	Cumulative_Bad_Ra	KS
(-inf, 376.0]	908	188	720	6.26%	28.07%	28.07%	79.30%	79.30%	0.17
(376.0, 396.0]	977	453	524	6.74%	20.43%	48.50%	53.63%	65.99%	0.12
(396.0, 408.0]	745	484	261	5.14%	10.18%	58.67%	35.03%	57.22%	0.11
(408.0, 424.0]	1281	1017	264	8.84%	10.29%	68.97%	20.61%	45.23%	0.06
(424.0, 432.0]	846	729	117	5.84%	4.56%	73.53%	13.83%	39.65%	0.10
(432.0, 454.0]	2985	2695	290	20.59%	11.31%	84.83%	9.72%	28.11%	0.03
(454.0, 468.0]	2065	1894	171	14.24%	6.67%	91.50%	8.28%	23.93%	0.04
(468.0, 479.0]	1284	1197	87	8.86%	3.39%	94.89%	6.78%	21.95%	0.09
(479.0, 508.0]	2507	2394	113	17.29%	4.41%	99.30%	4.51%	18.73%	0.04
(508.0, inf)	899	881	18	6.20%	0.70%	100.00%	2.00%	17.69%	0.19
Total	14497	11932	2565	100.00%	100.00%	100.00%	17.69%	17.69%	0.51

Score Distribution of Previously Approved Application

Bin	#Obs	#Good	#Bad	%Obs	%Bad	%Cumulative_Bad	%Bad_Rate	Cumulative_Bad_Ra	KS
(-inf, 376.0]	433	188	245	3.54%	15.82%	15.82%	56.58%	56.58%	0.19
(376.0, 396.0]	619	433	186	5.06%	12.01%	27.82%	30.05%	40.97%	0.10
(396.0, 408.0]	538	397	141	4.40%	9.10%	36.93%	26.21%	35.97%	0.11
(408.0, 424.0]	1002	808	194	8.20%	12.52%	49.45%	19.36%	29.55%	0.05
(424.0, 432.0]	693	582	111	5.67%	7.17%	56.62%	16.02%	26.70%	0.10
(432.0, 454.0]	2612	2329	283	21.36%	18.27%	74.89%	10.83%	19.67%	0.03
(454.0, 468.0]	1883	1712	171	15.40%	11.04%	85.93%	9.08%	17.11%	0.04
(468.0, 479.0]	1187	1100	87	9.71%	5.62%	91.54%	7.33%	15.81%	0.09
(479.0, 508.0]	2416	2303	113	19.76%	7.30%	98.84%	4.68%	13.45%	0.04
(508.0, inf)	843	825	18	6.90%	1.16%	100.00%	2.14%	12.67%	0.19
Total	12226	10677	1549	100.00%	100.00%	100.00%	12.67%	12.67%	0.34

Population Stability Index

Population Stability (AGB with Indeterminates)

Bin	%Obs_of_Development [A]	%Obs_of_Validation [B]	Difference [C]: (B-A)*100	Log Odd [D]: Log (B/A)	PSI Contribution: (C*D)
(-inf, 376.0]	0.06	0.04	2.67	0.56	1.49
(376.0, 396.0]	0.07	0.05	1.57	0.26	0.41
(396.0, 408.0]	0.05	0.04	0.90	0.19	0.17
(408.0, 424.0]	0.09	0.09	0.13	0.01	0.00
(424.0, 432.0]	0.06	0.06	0.01	0.00	0.00
(432.0, 454.0]	0.21	0.21	-0.10	-0.01	0.00
(454.0, 468.0]	0.14	0.15	-0.61	-0.04	0.03
(468.0, 479.0]	0.09	0.10	-1.20	-0.13	0.15
(479.0, 508.0]	0.17	0.21	-3.33	-0.18	0.58
(508.0, inf)	0.06	0.06	-0.04	-0.01	0.00
Total	1.00	1.00		PSI=	2.84304389

Population Stability (KGB with Indeterminates)

Bin	%Obs_of_Development [A]	%Obs_of_Validation [B]	Difference [C]: (B-A)*100	Log Odd [D]: Log (B/A)	PSI Contribution: (C*D)
(-inf, 376.0]	0.04	0.04	-0.05	-0.01	0.00
(376.0, 396.0]	0.05	0.05	-0.11	-0.02	0.00
(396.0, 408.0]	0.04	0.04	0.16	0.04	0.01
(408.0, 424.0]	0.08	0.09	-0.51	-0.06	0.03
(424.0, 432.0]	0.06	0.06	-0.15	-0.03	0.00
(432.0, 454.0]	0.21	0.21	0.67	0.03	0.02
(454.0, 468.0]	0.15	0.15	0.55	0.04	0.02
(468.0, 479.0]	0.10	0.10	-0.35	-0.04	0.01
(479.0, 508.0]	0.20	0.21	-0.86	-0.04	0.04
(508.0, inf)	0.07	0.06	0.65	0.10	0.06
Total	1.00	1.00		PSI=	0.19906937

6

code demo

模型结果 标准化输出



- (1) 将所有结果自动输出为一个文件夹
- (2) 那这个文件夹长什么样子呢？

xxxxx项目模型	--	文件夹	今天 下午12:45
a. Summary of Results	--	文件夹	今天 下午12:45
Image	--	文件夹	今天 下午12:45
Cumulative_Approve.png	58 KB	PNG 图像	今天 下午12:45
KS_1.png	68 KB	PNG 图像	今天 下午12:45
PR.png	64 KB	PNG 图像	今天 下午12:45
ROC.png	87 KB	PNG 图像	今天 下午12:45
Score_Point_wise_Bad_Rate.png	35 KB	PNG 图像	今天 下午12:45
Validation_KS.png	61 KB	PNG 图像	今天 下午12:45
Validation_PR_F1.png	62 KB	PNG 图像	今天 下午12:45
Model Element	--	文件夹	今天 下午12:45
Model_Result.model	1.8 MB	文稿	今天 下午12:45
Model Sample	--	文件夹	今天 下午12:45
Development - AGB_Data_Original.pkl	3.9 MB	文稿	今天 下午12:45
Development - AGB_Data_WOE.pkl	3.7 MB	文稿	今天 下午12:45
Development - KGB_Data_Original.pkl	3.3 MB	文稿	今天 下午12:45
Development - KGB_Data_WOE.pkl	3.1 MB	文稿	今天 下午12:45
Model_Bin_Data.pkl	35 KB	文稿	今天 下午12:45
Validation_Data_Original.pkl	1.4 MB	文稿	今天 下午12:45
Validation_Data_WOE.pkl	1.3 MB	文稿	今天 下午12:45
Model_Results_Combine.xls	158 KB	Micros...ok (.xls)	今天 下午12:45
Summary_of_Results.xlsx	282 KB	Micros...k (.xlsx)	今天 下午12:45
b. Final Scorecard (Additional Results)	--	文件夹	今天 下午12:45
c. Performance Definitions	--	文件夹	今天 下午12:45
d. Characteristics Analysis Report	--	文件夹	今天 下午12:45
Characteristics_Analysis_Report.xlsx	27 KB	Micros...k (.xlsx)	今天 下午12:45
e. Characteristic Analysis of Validation	--	文件夹	今天 下午12:45
f. Score Point Distribution data	--	文件夹	今天 下午12:45
Score_Point_Distribution_Data.xlsx	21 KB	Micros...k (.xlsx)	今天 下午12:45
g. List of Application Data Variables	--	文件夹	今天 下午12:45

模型结果
汇总文件

▼	h. Technical Specifications Document	--	文件夹	今天 下午12:45
▼	i. Datasets used for Analysis	--	文件夹	今天 下午12:45
▼	j. Project Tracker	--	文件夹	今天 下午12:45
▼	k. Issues Log	--	文件夹	今天 下午12:45

① 创建模型输出结构

```
1  # -*- coding:utf-8 -*-
2  # creator:
3
4  import ...
5
6
7
8  class Make_Director(object):
9
10     def __init__(self):
11         pass
12
13     @staticmethod
14     def mk_dir(path):
15         """
16         Explanation
17         -----
18         判断路径文件夹是否存在
19         (1) 如果存在则删除重建
20         (2) 如果不存在直接新建
21         """
22         if os.path.isdir(path):
23             shutil.rmtree(path)
24         os.mkdir(path)
25
26     @staticmethod
27     def create_dir_for_scorecard(output_path,result_name=''):
28         """
29         Explanation
30         -----
31         建立模型输出文件夹结构
32
33         Parameters
34         -----
35         output_path: string
36         | 模型结果保存路径
37
38         result_name: string,default=''
39         | 模型结果文件夹名称, 默认为空, 此时结果文件夹名称默认为 "model_result_当前年月日"
40
41         Return
42         -----
43         summary_result: string
44         | 模型结果摘要路径
```

模型结果摘要路径

Characteristics_Analysis_Report: string

变量分析报告储存路径

Characteristic_Analysis_of_Validation: string

外验证集变量分析报告储存路径

Performance_Definitions: string

标签定义结果储存路径

Score_Point_Distribution: string

模型得分详细分析保存路径

image_path: string

模型中间图形结果储存路径

sample_path: string

模型样本储存路径

model_path: string

模型model文件储存路径

```
"""
date=datetime.datetime.now().strftime('%Y%m%d')
file_name=result_name if result_name else 'model_result_%s'%date
if output_path=="":
    output_path = os.getcwd() + '/' +file_name
    print("The path of the model result saved is '"+output_path+"!")
elif output_path[-1]=='/':
    output_path+=file_name
else:
    output_path += '/' +file_name
Make_Director.mk_dir(output_path)
summary_result=output_path+'/a. Summary of Results'
Characteristics_Analysis_Report=output_path+'/d. Characteristics Analysis Report'
Performance_Definitions=output_path+'/c. Performance Definitions'
Final_Scorecard=output_path+'/b. Final Scorecard (Additional Results)'
Characteristic_Analysis_of_Validation=output_path+'/e. Characteristic Analysis of Validation'
Technical_Specifications_Document=output_path+'/h. Technical Specifications Document'
Project_Tracker=output_path+'/j. Project Tracker'
Score_Point_Distribution=output_path+'/f. Score Point Distribution data'
Application_Data_Variables=output_path+'/g. List of Application Data Variables'
Datasets_used_for_Analysis=output_path+'/i. Datasets used for Analysis'
Issues_Log=output_path+'/k. Issues Log'
#####
```

```
89     image_path=summary_result+'/Image'
90     result_path = summary_result + '/Result'
91     sample_path = summary_result + '/Model Sample'
92     model_path=summary_result+'/Model Element'
93     #####
94     Make_Director.mk_dir(summary_result)
95     Make_Director.mk_dir(Characteristics_Analysis_Report)
96     Make_Director.mk_dir(Performance_Definitions)
97     Make_Director.mk_dir(Final_Scorecard)
98     Make_Director.mk_dir(Characteristic_Analysis_of_Validation)
99     Make_Director.mk_dir(Technical_Specifications_Document)
100    Make_Director.mk_dir(Project_Tracker)
101    Make_Director.mk_dir(Score_Point_Distribution)
102    Make_Director.mk_dir(Application_Data_Variables)
103    Make_Director.mk_dir(Datasets_used_for_Analysis)
104    Make_Director.mk_dir(Issues_Log)
105    #####
106    Make_Director.mk_dir(image_path)
107    Make_Director.mk_dir(result_path)
108    Make_Director.mk_dir(sample_path)
109    Make_Director.mk_dir(model_path)
110    return summary_result,Characteristics_Analysis_Report,Characteristic_Analysis_of_Validation,Performance_Definitions,Score_Point_Distribution,image_path,sam
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
```

2 模型结果汇总输出

```
418 @staticmethod
419 def combine_model_result(flag_name, not_var_list, var_list, reject_name, var_description, score_bin, train_data_bin, train_pre, test_pre, oot_pre,
420                          train_data_stat, test_data_stat,
421                          oot_data_stat, train_data_original, test_data_original, oot_data_original,
422                          output_path, output_name,
423                          train_name='Train', test_name='Test', oot_name='Validation',
424                          not_in_list=['None', 'NaN', 'NA', 'nan', None],
425                          model_type='Logistic', result=None, train_data_woe=pandas.DataFrame(), test_data_woe=pandas.DataFrame(), oot_data_woe=pandas.DataFrame()):
426
427     """
428     Explanation
429     保存最终模型结果
430
431     Parameters
432
433     flag_name: string
434     | 好坏标签名称
435
436     not_var_list: list
437     | 非指标列表
438
439     reject_name: string
440     | 被拒绝标签名称
441
442     score_bin: pandas.core.frame.DataFrame
443     | 模型得分最优分bin
444
445     train_data_bin: pandas.core.frame.DataFrame
446     | 根据训练集计算的最优分bin
447
448     train_pre: pandas.core.frame.DataFrame
449     | 训练集预测结果
450
451     test_pre: pandas.core.frame.DataFrame
452     | 测试集预测结果
453
454     oot_pre: pandas.core.frame.DataFrame
455     | 外验证集预测结果
456
457     train_data_stat: pandas.core.frame.DataFrame
458     | 训练集上, 指标的表现, 指标的iv、ks、ar、相关系数等等
```

```
458 | 训练集上, 指标的表现, 指标的iv、ks、ar、相关系数等等
459 |
460 | test_data_stat: pandas.core.frame.DataFrame
461 | 测试集上, 指标的表现, 指标的iv、ks、ar、相关系数等等
462 |
463 | oot_data_stat: pandas.core.frame.DataFrame
464 | 外验证集上, 指标的表现, 指标的iv、ks、ar、相关系数等等
465 |
466 | train: pandas.core.frame.DataFrame
467 | 切分的训练集原始样本
468 |
469 | test: pandas.core.frame.DataFrame
470 | 切分的测试集原始样本
471 |
472 | data_oot: pandas.core.frame.DataFrame
473 | 外验证集合原始样本
474 |
475 | result: statsmodels.discrete.discrete_model.BinaryResultsWrapper
476 | 模型结果
477 |
478 | pvalue_list: list
479 | 入模变量
480 |
481 | output_path: string, default=''
482 | 结果保存路径
483 |
484 | output_name: string, default=''
485 | 结果保存文件夹名称
486 |
487 | train_name:string, default='train'
488 | 训练集名称
489 |
490 | test_name:string, default='test'
491 | 测试集名称
492 |
493 | oot_name:string, default='oot'
494 | 外验证集名称
495 |
496 | not_in_list: list, default=['None','NaN','NA','nan',None]
497 | 空值列表
498 |
499 | model_type:string, default='Logistic'
500 | 模型类型
501 |
502 | result: , default=None
```

模型结果文件,只在logit模型生效

```
504
505 train_data_woe: pandas.core.frame.DataFrame
506     训练集woe化结果, 只在logit模型生效
507
508 test_data_woe: pandas.core.frame.DataFrame
509     测试集woe化结果, 只在logit模型生效
510
511 oot_data_woe: pandas.core.frame.DataFrame
512     外验证集woe化结果, 只在logit模型生效
513 """
514 if var_list==[]:
515     var_list=list(set(train_data_original.columns)-set(not_var_list))
516 # sample analyse
517 train_analyse = Summary_Of_Model_Result.model_analyse(train_pre, flag_name, train_name)
518 test_analyse = Summary_Of_Model_Result.model_analyse(test_pre, flag_name, test_name)
519 oot_analyse = Summary_Of_Model_Result.model_analyse(oot_pre, flag_name, oot_name)
520 final_analyse = pandas.concat([train_analyse, test_analyse, oot_analyse])
521 # sample corr
522 final_corr = pandas.concat([train_data_original,test_data_original]).corr()
523 # model result
524 train_information = Summary_Of_Model_Result.model_information(train_pre, 'predict', flag_name, train_name)
525 test_information = Summary_Of_Model_Result.model_information(test_pre, 'predict', flag_name, test_name)
526 oot_information = Summary_Of_Model_Result.model_information(oot_pre, 'predict', flag_name, oot_name)
527 final_information = pandas.concat([train_information, test_information, oot_information])
528 # index information
529 blank = pandas.DataFrame([''] * len(train_data_stat.columns), columns=train_data_stat.columns)
530 final_data_stat = pandas.concat([train_data_stat, blank, test_data_stat])
531 if len(oot_data_stat) > 0:
532     final_data_stat = pandas.concat([final_data_stat, blank, oot_data_stat])
533 # model vif
534 if model_type=='Logistic':
535     tmp_list=list(result.params.index)
536     tmp_list.remove('intercept')
537 train_vif = Summary_Of_Model_Result.model_vif(train_data_woe, tmp_list, '%s_' % train_name)
538 test_vif = Summary_Of_Model_Result.model_vif(test_data_woe, tmp_list, '%s_' % test_name)
539 oot_vif = Summary_Of_Model_Result.model_vif(oot_data_woe,tmp_list, '%s_' % oot_name)
540 final_vif = pandas.merge(train_vif, test_vif, on='Characteristic', how='left')
541 if len(oot_vif) > 0:
542     final_vif = pandas.merge(final_vif, oot_vif, on='Characteristic', how='left')
543 else:
544     final_vif=pandas.DataFrame()
545 # model summary
546 final_summary = Summary_Of_Model_Result.model_summary(result)
547 # model description
548 tmp_list=(train_data_original.columns) - set(not_var_list)
```

```

548 tmp_list = list(set(train_data_original.columns) - set(not_var_list))
549 total_desc = Summary_Of_Model_Result.model_describe(pandas.concat([train_data_original, test_data_original]), tmp_list,
550                                                     'All Data')
551 oot_desc = Summary_Of_Model_Result.model_describe(oot_data_original, tmp_list, oot_name)
552 final_desc = pandas.concat([total_desc, oot_desc])
553 # # model psi
554 # test_psi = Summary_Of_Model_Result.model_psi(train_data_woe, test_data_woe, pvalue_list,
555 #                                             '%s_Relative_To_%s_' % (test_name, train_name))
556 # oot_psi = Summary_Of_Model_Result.model_psi(train_data_woe, oot_data_woe, pvalue_list,
557 #                                             '%s_Relative_To_%s_' % (oot_name, train_name))
558 # final_psi = pandas.merge(test_psi, oot_psi, on='Characteristic', how='left')
559
560 # model score
561 flag_list = [flag_name]
562 if reject_name:
563     flag_list.append(reject_name)
564 train_pre = Summary_Of_Model_Result.model_score(train_pre, 'predict', train_name, model_type)
565 test_pre = Summary_Of_Model_Result.model_score(test_pre, 'predict', test_name, model_type)
566 oot_pre = Summary_Of_Model_Result.model_score(oot_pre, 'predict', oot_name, model_type)
567 if reject_name:
568     total_pre = copy.deepcopy(train_pre)
569 else:
570     tmp_pre = copy.deepcopy(test_pre)
571     tmp_pre.columns = train_pre.columns
572     total_pre = Summary_Of_Model_Result.model_score(pandas.concat([train_pre, tmp_pre]),
573                                                     'predict', train_name, model_type)
574 final_score = Summary_Of_Model_Result.model_score_describe(score_bin, flag_name, not_in_list, total_pre, train_name)
575 kgb_pre = copy.deepcopy(test_pre)
576 kgb_pre.columns = train_pre.columns
577 test_score = Summary_Of_Model_Result.model_score_describe(score_bin, flag_name, not_in_list,
578                                                         kgb_pre,
579                                                         train_name)
580 if len(oot_pre) >= 0:
581     oot_pre['%s_score' % train_name] = oot_pre['%s_score' % oot_name]
582     oot_score = Summary_Of_Model_Result.model_score_describe(score_bin, flag_name, not_in_list, oot_pre, train_name)
583     del oot_score['Bin']
584     validation_score = pandas.concat([final_score, oot_score], axis=1)
585     kgb_score = pandas.concat([test_score, oot_score], axis=1)
586 else:
587     validation_score = pandas.DataFrame()
588     kgb_score = pandas.DataFrame()
589 # all bin
590 final_sep_bin = Summary_Of_Model_Result.sep_bin(train_data_bin, var_description)
591 train_data_bin = pandas.merge(var_description, train_data_bin, on='Characteristic', how='right')
592 # bin score
593 if

```

```
593 if model_type=='Logistic':
594     A, B = Summary_Of_Model_Result.model_score_coefficient_by_step(10, 500, 2, 30)
595     final_bin_score = Summary_Of_Model_Result.bin_score(train_data_bin, result, A, B, var_description)
596     final_bin_score_more=Summary_Of_Model_Result.bin_score_more(train_data_bin, result, A, B, var_description)
597 else:
598     final_bin_score=pandas.DataFrame()
599     final_bin_score_more=pandas.DataFrame()
600 # model swap
601 if reject_name:
602     swap_c, swap_p, swap_effect = Summary_Of_Model_Result.model_swap_set(total_pre, flag_name, reject_name)
603 else:
604     swap_c, swap_p, swap_effect = Summary_Of_Model_Result.model_swap_set(total_pre, flag_name, flag_name)
605 swap = pandas.concat([swap_c, pandas.DataFrame([''] * 3, columns=['']), swap_p], axis=1)
606 # Score Point Distribution Data
607 score_point = Best_Bin_Cut.group_by_df_more(total_pre, flag_name, '%s_score' % train_name, '#Bad', '#Good')
608 # model psi
609 model_psi=Summary_Of_Model_Result.model_score_psi(validation_score)
610 if reject_name:
611     model_psi_kgb=Summary_Of_Model_Result.model_score_psi(kgb_score)
612 else:
613     model_psi_kgb=pandas.DataFrame()
614 # validation characteristic
615 if len(oot_pre) > 0:
616     val_list = ['Characteristic', 'Char_Type', 'Bin', 'WOE', '#0bs', '%0bs', '%Bad_Rate', '#0bs_' + oot_name,
617               '%0bs_' + oot_name, '%Bad_Rate_' + oot_name]
618     validation_char = train_data_bin[val_list]
619     validation_char['PSI-' + oot_name] = list(map(lambda x, y: 100 * (x - y) * numpy.log(x / y),
620                                               validation_char['%0bs'], validation_char['%0bs_' + oot_name]))
621     validation_char = Summary_Of_Model_Result.sep_bin(validation_char, var_description)
622 else:
623     validation_char = pandas.DataFrame()
624 # create result dir
625 summary_result, Characteristics Analysis Report, Characteristic Analysis of Validation, Performance Definitions, Score Point Distribution, image_path, samp
626     output_path, output_name)
627 Model_Plot.plot_ROC(train_pre, test_pre, oot_pre, flag_name, train_name, test_name, oot_name, image_path)
628 # get PR plot
629 Model_Plot.plot_PR(train_pre, test_pre, oot_pre, flag_name, train_name, test_name, oot_name, image_path)
630 # get PR_F1 plot
631 Model_Plot.plot_PR_F1(train_pre, flag_name, train_name, image_path)
632 Model_Plot.plot_PR_F1(test_pre, flag_name, test_name, image_path)
633 Model_Plot.plot_PR_F1(oot_pre, flag_name, oot_name, image_path)
634 #
635 Model_Plot.plot_good_bad(train_pre, flag_name, train_name, image_path)
636 # get KS plot
637 Model_Plot.plot_KS(train_pre, flag_name, train_name, image_path)
```

```

639 Model_Plot.plot_KS(test_pre, flag_name, test_name, image_path)
640 Model_Plot.plot_KS(oot_pre, flag_name, oot_name, image_path)
641 Model_Plot.plot_KS_1(train_pre, flag_name, train_name, image_path)
642 Model_Plot.cumulative_approve(train_pre, flag_name, train_name, image_path)
643 Model_Plot.score_point_bad_rate(total_pre, flag_name, train_name, image_path)
644 # Save the combine result of the model to excel
645 model_result_path = summary_result + '/Result/Model_Results_Combine.xls'
646 with pandas.ExcelWriter(model_result_path) as writer:
647     pandas.DataFrame().to_excel(writer, sheet_name=u'01.概述(Overview)', index=False)
648     final_analyse.to_excel(writer, sheet_name=u'02.样本总体分析(Sample Analyse)', index=False)
649     final_summary.to_excel(writer, sheet_name=u'03.模型结果', index=False)
650     if len(final_bin_score)>0:
651         final_bin_score.to_excel(writer, sheet_name='03+.评分卡(Final Scorecard)', index=False)
652         final_bin_score_more.to_excel(writer, sheet_name=u'03+.评分卡分析(Scorecard Analyse)', index=False)
653     final_information.to_excel(writer, sheet_name=u'04.模型表现(Effect Statistics)', index=False)
654     final_score.to_excel(writer, sheet_name=u'05.模型分数分布(Score Distribution)', index=False)
655     validation_score.to_excel(writer, sheet_name=u'05+.外验证分数分布(Val Distribution)', index=False)
656     swap.to_excel(writer, sheet_name=u'06.Swap Set(Screen process)', index=False)
657     swap_effect.to_excel(writer, sheet_name=u'06+.坏样本率提升(Bad Rate Promotion)', index=False)
658     model_psi.to_excel(writer, sheet_name=u'07.模型分数PSI(Model Score PSI)', index=False)
659     score_point.to_excel(writer, sheet_name=u'12.模型分数报告(Score Point)', index=False)
660     final_sep_bin[final_sep_bin.columns[0:18]].to_excel(writer, sheet_name=u"13.变量分组结果(Variables Report)", index=False)
661     train_data_bin[train_data_bin['Characteristic'].isin(var_list)].to_excel(writer, sheet_name=u"13+.入模变量分组结果(Variables Report)",
662                                     index=False)
663     if reject_name:
664         final_data_stat.to_excel(writer, sheet_name=u"14.变量评价信息(Variables Effect)", index=False)
665         train_data_stat.to_excel(writer, sheet_name=u"14+.AGB变量信息(AGB Effect)", index=False)
666         test_data_stat.to_excel(writer, sheet_name=u"14+.KGB变量信息(KGB Effect)", index=False)
667         model_psi_kgb.to_excel(writer, sheet_name=u"14+.Development-KGB_PSI", index=False)
668         test_score.to_excel(writer, sheet_name=u"14+.Development-KGB分数分布", index=False)
669     else:
670         train_data_stat.to_excel(writer, sheet_name=u"14.变量评价信息(Variables Effect)", index=False)
671         oot_data_stat.to_excel(writer, sheet_name=u"15.外验证变量信息(Validation Effect)", index=False)
672         validation_char.to_excel(writer, sheet_name=u"16.外验证分组结果(Validation Report)", index=False)
673         final_corr.to_excel(writer, sheet_name=u"17.变量相关系数(Variables Corr)", index=False)
674         final_desc.to_excel(writer, sheet_name=u"18.变量分布(Variables Distribution)", index=False)
675         if model_type=='Logistic':
676             final_vif.to_excel(writer, sheet_name=u"19.变量VIF(Variables VIF)", index=False)
677     writer.save()
678 clean_path = summary_result + '/Summary_of_Results.xlsx'
679 model_result_beautify(image_path, model_result_path, clean_path, Characteristics_Analysis_Report,
680                       Characteristic_Analysis_of_Validation, Score_Point_Distribution, 10,
681                       500, 2, 30,model_type)
682 summary_path=summary_result + '/Model_Results_Combine.xlsx'
683 model_result_combine(image_path, model_result_path,summary_path, train_name,test_name,oot_name, 10,

```

```
683         500, 2, 30, model_type)
684 # save data
685 try:
686     train_data_woe.to_pickle(sample_path + '/%s_Data_WOE.pkl' % train_name)
687     test_data_woe.to_pickle(sample_path + '/%s_Data_WOE.pkl' % test_name)
688     oot_data_woe.to_pickle(sample_path + '/%s_Data_WOE.pkl' % oot_name)
689     result.save(model_path + '/Model_Result.model')
690 except:
691     pass
692 train_data_original.to_pickle(sample_path + '/%s_Data_Original.pkl' % train_name)
693 test_data_original.to_pickle(sample_path + '/%s_Data_Original.pkl' % test_name)
694 train_data_bin.to_pickle(sample_path + '/Characteristic_Bin_Data.pkl')
695 score_bin.to_pickle(sample_path + '/Score_Bin_Data.pkl')
696 if len(oot_data_original) > 0:
697     oot_data_original.to_pickle(sample_path + '/%s_Data_Original.pkl' % oot_name)
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
```

3 模型结果美化输出

```
1053 def model_result_beautify(image_path, in_path, out_path, Characteristics_Analysis_Report,  
1054                           Characteristic_Analysis_of_Validation, Score_Point_Distribution, base_odds, base_score, add_odds, add_score, model_type):  
1055     """  
1056     Explanation  
1057     将模型汇总结果美化, 可视化展示  
1058  
1059     Parameters  
1060     image_path:string  
1061     | 图片路径  
1062  
1063     in_path:string  
1064     | 模型汇总结果路径  
1065  
1066     out_path:string  
1067     | 美化后模型汇总结果存储路径  
1068  
1069     Characteristics_Analysis_Report:string  
1070     | 变量分析报告保存路径  
1071  
1072     Characteristic_Analysis_of_Validation: string  
1073     | 外验证集变量分析报告储存路径  
1074  
1075     Score_Point_Distribution: string  
1076     | 模型得分详细分析保存路径  
1077  
1078     base_odds:float, default=50  
1079     | 基准好坏比  
1080  
1081     base_score:int, default=500  
1082     | 基准分  
1083  
1084     add_odds:intfloat, default=2  
1085     | odds变化步长  
1086  
1087     add_score:int, default=20  
1088     | odds变化一个步长, 相应增长的分数  
1089     """  
1090     data = xlrd.open_workbook(in_path)  
1091     name = data.sheet_names()  
1092     wb = xlswriter.Workbook(out_path)
```

```

1094 wb = xlswriter.Workbook(out_path)
1095 # sheet第四行, 写入总标题
1096 body_title1 = wb.add_format(title1_dic)
1097 # sheet第五行, 留白
1098 body_title2 = wb.add_format(titel2_dic)
1099 pps = wb.add_format(pps12_dic)
1100 dic = {'u'03+.评分卡(Final Scorecard)': '1.Final Scorecard',
1101       'u'04.模型表现(Effect Statistics)': '2.Model Effect Statistics',
1102       'u'05.模型分数分布(Score Distribution)': '4.Score Distribution',
1103       'u'06.Swap Set(Screen process)': '6.Model Swap Set',
1104       'u'05+.外验证分数分布(Val Distribution)': '5.Validation Score Distribution',
1105       'u'07.模型分数PSI(Model Score PSI)': '7.Population Stability Index',
1106       'u'13.变量分组结果(Variables Repoort)': 'Characteristic Analyse Report',
1107       'u'12.模型分数报告(Score Piont)': 'Score_Point_Distribution_Data',
1108       'u'16.外验证分组结果(Validation Report)': 'Validation_Analysis'}
1109 for k in name:
1110     table = data.sheet_by_name(k)
1111     nrows = table.nrows # 行数
1112     ncols = table.ncols # 列数
1113     if k in ['u'03+.评分卡(Final Scorecard)', 'u'04.模型表现(Effect Statistics)', 'u'05.模型分数分布(Score Distribution)', 'u'06.Swap Set(Screen process)',
1114            'u'05+.外验证分数分布(Val Distribution)', 'u'07.模型分数PSI(Model Score PSI)']:
1115         ws = wb.add_worksheet(dic[k])
1116         if k == 'u'03+.评分卡(Final Scorecard)':
1117             final_scorecard(wb, ws, table, nrows, ncols,dic[k])
1118         elif k == 'u'04.模型表现(Effect Statistics)':
1119             key_statistics(wb, ws, table, nrows, ncols,dic[k])
1120         try:
1121             ws_roc_plot = wb.add_worksheet('3.ROC Curve')
1122             ws_roc_plot.set_row(0, 7)
1123             ws_roc_plot.set_row(2, 42)
1124             ws_roc_plot.set_column(0, 0, 3)
1125             ws_roc_plot.merge_range(2, 1, 2, image_col, '3.ROC Curve', body_title1)
1126             ws_roc_plot.merge_range(1, 1, 1, image_col, '', body_title2)
1127             ws_roc_plot.merge_range(3, 1, 3, image_col, '', body_title2)
1128             ws_roc_plot.center_horizontally()
1129             ws_roc_plot.center_vertically()
1130             ws_roc_plot.hide_gridlines({'option': 1})
1131             ws_roc_plot.hide_gridlines({'option': 1})
1132             ws_roc_plot.insert_image('B4', image_path + '/ROC.png', {'x_scale': image_len, 'y_scale': image_len})
1133         except:
1134             pass
1135     elif k == 'u'05.模型分数分布(Score Distribution)':
1136         score_distribution(wb, ws, table, nrows, ncols,dic[k],base_odds,base_score,add_odds,add_score,model_type)
1137     elif k == 'u'06.Swap Set(Screen process)':
1138         set_model_swap(wb, ws, table, nrows, ncols,dic[k])

```

```
1140 elif k == u'05+.外验证分数分布(Val Distribution)':
1141     score_distribution_validation(wb, ws, table, nrows, ncols, dic[k])
1142 elif k == u'07.模型分数PSI(Model Score PSI)':
1143     population_stability(wb, ws, table, nrows, ncols, dic[k])
1144     try:
1145         ws_roc_plot = wb.add_worksheet('8.Ascending Score Report')
1146         ws_roc_plot.set_row(0, 7)
1147         ws_roc_plot.set_row(3, 7)
1148         ws_roc_plot.set_column(0, 0, 3)
1149         ws_roc_plot.merge_range(2, 1, 2, image_col, '8.Ascending Score Report', body_title1)
1150         ws_roc_plot.merge_range(1, 1, 1, image_col, '', body_title2)
1151         ws_roc_plot.merge_range(3, 1, 3, image_col, '', body_title2)
1152         ws_roc_plot.center_horizontally()
1153         ws_roc_plot.center_vertically()
1154         ws_roc_plot.hide_gridlines({'option': 1})
1155
1156         p1 = u'a. Population Distribution of Good and Bad Accounts by Score Point'
1157         ws_roc_plot.merge_range(4, 1, 4, image_col, p1, pps)
1158         ws_roc_plot.insert_image('B6', image_path + '/KS-1.png', {'x_scale': image_len, 'y_scale': image_len})
1159         p2 = u'b. Cumulative Approval and Cumulative Bad Rate of Population'
1160         ws_roc_plot.merge_range(32, 1, 32, image_col, p2, pps)
1161         ws_roc_plot.insert_image('B34', image_path + '/Cumulative Approve.png',
1162                                 {'x_scale': image_len, 'y_scale': image_len})
1163         p3 = u'c. Score Point wise Bad-Rate'
1164         ws_roc_plot.merge_range(60, 1, 60, image_col, p3, pps)
1165         ws_roc_plot.insert_image('B62', image_path + '/Score Point wise Bad Rate.png',
1166                                 {'x_scale': image_len, 'y_scale': image_len})
1167     except:
1168         pass
1169 elif k == u'13.变量分组结果(Variables Report)':
1170     wb_bin = xlswriter.Workbook(Characteristics_Analysis_Report + '/Characteristics_Analysis_Report.xlsx')
1171     ws_bin = wb_bin.add_worksheet(dic[k])
1172     all_bin(wb_bin, ws_bin, table, nrows, ncols, dic[k])
1173     wb_bin.close()
1174 elif k == u'12.模型分数报告(Score Point)':
1175     wb_point = xlswriter.Workbook(Score_Point_Distribution + '/Score_Point_Distribution_Data.xlsx')
1176     ws_point = wb_point.add_worksheet(dic[k])
1177     score_point_data(wb_point, ws_point, table, nrows, ncols, dic[k])
1178     wb_point.close()
1179 elif k == u'16.外验证分组结果(Validation Report)':
1180     wb_val = xlswriter.Workbook(Characteristic_Analysis_of_Validation + '/Characteristic_Analysis_of_Validation.xlsx')
1181     ws_val = wb_val.add_worksheet(dic[k])
1182     validation_bin(wb_val, ws_val, table, nrows, ncols, dic[k])
1183     wb_val.close()
1184 elif k == u'06+.坏样本率提升(Bad Rate Promotion)':
1185     ws = wb.get_worksheet_by_name('16.Model_Score_Set1')
```


评分卡全流程示例



T H A N K S