总行零售风险管理部数据分析岗（智贷方向）小任务报告

【已删去个人信息】

环境、框架

本次实验我是在 Google Colab 上 CPU 机器上进行的，配置应该是 4 核。主要用了 numpy, pandas, lightgbm, sklearn, feature_selector 等库。环境 conda python3。

代码

本报告同一目录下，留有 pa-task.py， pa-task.ipynb（建议）， 数据集 pa-task-data.csv。建议使用 ipynb 打开，保留有所有执行完的结果，部分实验的特征挖掘可能会花费点时间，不过应该不会超过半个小时，但会根据机器性能进行浮动。

数据初探

首先拿到数据后对数据进行分析。对各特征的含义进行揣摩以及范围进行大概的认知。

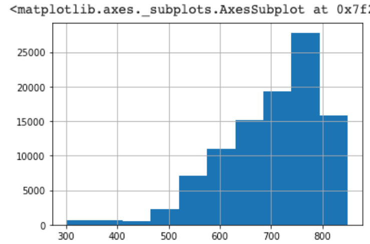
```
[ ] # 数据特征猜测： 学历
    df_data['diploma'].value_counts()

    1    40045
    2    24862
    3    20188
    4     9950
    5     4955
    Name: diploma, dtype: int64
```

```
[ ] # 数据特征猜测： 是否拥有房产
    df_data['home_ownership'].value_counts()

    1    80731
    0    19269
    Name: home_ownership, dtype: int64
```

```
[ ] # 数据特征猜测： 是否拥有车
    df_data['car_ownership'].value_counts()

    1    60829
    0    39171
    Name: car_ownership, dtype: int64
```

```
[ ] df_data['location'].value_counts()

    1    70757
    0    29243
    Name: location, dtype: int64
```

```
[ ] df_data['scr'].hist()

    <matplotlib.axes._subplots.AxesSubplot at 0x7f21bd804710>
```




```
[ ] # 数据特征猜测： 星座
    df_data['constellation'].value_counts()

    # 牡羊座/白羊座 (3/21 - 4/20)的英文名: Aries
    # 金牛座 (4/21 - 5/20)的英文名:  Taurus
    # 双子座 (5/21 - 6/21)的英文名:  Gemini
    # 巨蟹座 (6/22 - 7/22)的英文名:  Cancer
    # 狮子座 (7/23 - 8/22)的英文名:  Leo
    # 处女座/室女座 (8/23 - 9/22)的英文名:  Virgo
    # 天秤座 (9/23 - 10/22)的英文名:  Libra
    # 天蝎座 (10/23 - 11/21)的英文名:  Scorpio
    # 射手座/人马座 (11/22 - 12/21)的英文名:  Sagittarius
    # 魔羯座/山羊座 (12/22 - 1/19)的英文名:  Capricorn
    # 水瓶座 (1/20 - 2/18)的英文名:  Aquarius
    # 双鱼座 (2/19 - 3/20)的英文名:  Pisces
    # https://www.8s8s.com/xingzuo/xingzuozhishi/18624.html

    Libra          8938
    Virgo          8933
    Capricom       8812
    Taurus         8793
    Aries          8391
    Sagittarius    8321
    Aquarius       8031
    Pisces         8013
    Scorpio        7979
    Leo            7948
    Gemini         7942
    Cacer          7899
    Name: constellation, dtype: int64
```

```
[ ] # 数据特征猜测： 用户等级（分段）
    df_data['grade'].value_counts()

    3     29239
    4     24703
    2     19448
    5      9988
    6      5136
    1      4831
    8      3533
    7      3122
    Name: grade, dtype: int64
```

```
[ ] df_data['index'].value_counts()

    2      10102
    7      10079
    6      10060
    8      10052
    4      10045
    10     10027
    1      10012
    9       9998
    5       9871
    3       9754
    Name: index, dtype: int64
```

```
[ ] # 数据特征猜测： 贷款数量
    df_data['dk_cnt'].value_counts()

    2     51393
    1     33740
    3      8239
    4      3044
    6      2194
    5      1390
    Name: dk_cnt, dtype: int64
```

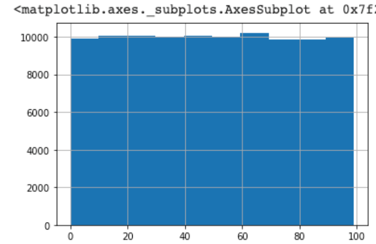
```
[ ] # 数据特征猜测： 总金额?
    df_data['tot_amnt'].value_counts()

    2     38539
    1     33740
    3     13687
    4      6900
    6      4343
    5      2791
    Name: tot_amnt, dtype: int64
```

```
[ ] # 数据特征猜测： 收入（分段）
    df_data['income'].value_counts()

    1     46209
    2     19926
    3     10942
    4      7887
    5      5953
    6      4973
    7      4110
    Name: income, dtype: int64
```

```
[ ] # 数据特征猜测： 年龄
    df_data['gender'].value_counts()

    M     51379
    F     48621
    Name: gender, dtype: int64
```

```
[ ] # 数据特征猜测： 职业领域
    df_data['occupation'].value_counts()

    G     14392
    F     14373
    A     14311
    E     14306
    D     14271
    B     14257
    C     14090
    Name: occupation, dtype: int64
```

```
[ ] # 数据特征猜测： 贷款金额
    df_data['dk_amnt(k)'].hist()

    <matplotlib.axes._subplots.AxesSubplot at 0x7f21bd75afd0>
```




```
[ ] # 数据特征猜测： 工作时长
    df_data['emp_length'].value_counts()

    1     20146
    4     20008
    5     20006
    3     19991
    2     19849
    Name: emp_length, dtype: int64
```

```
[ ] # 数据特征猜测： dq? 数量
    df_data['dq_cnt'].value_counts()

    1     75520
    5     14048
    2      4368
    3      4320
    4      1744
    Name: dq_cnt, dtype: int64
```

```
[ ] # label 分布
    df_data['y'].value_counts()

    0     96003
    1      3997
    Name: y, dtype: int64
```

接着，再对 label 的分布进行查看，发现极度不平衡


```
[ ] # label 分布
    df_data['y'].value_counts()

    0     96003
    1      3997
    Name: y, dtype: int64
```


在对数据初探后，主要认知：

- 部分特征可以通过 key 进行猜测，但是有些还是无法确定。
- 数据看上去比较干净，甚至某些类别的分布太过于平均，猜想这个数据可能并非真正来源于业务数据加密，或者进行了多次抽样，或者对数据进行了人为修改以达到这个目的。
- 类别十分不平均（符合真实业务场景），因此这里主要使用树模型进行建模。

\# 数据预处理

使用 feature_selector 进行缺省值，单一变量，高相关度， 0 重要度等进行数据删选，发现无需任何动作，数据非常干净， 与之前的猜测一致。猜想可能是为了降低小任务难度，或者数据本身非来源于业务场景，数据无需进行清洗和预处理。

feature_selector 库源码地址：https://github.com/WillKoehrsen/feature-selector

```
[ ] # from feature_selector import FeatureSelector
    # https://github.com/WillKoehrsen/feature-selector
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.u
  import pandas.util.testing as tm
```

```
[ ] # fs = FeatureSelector(data = df_data.drop(columns = ['y', 'type']), labels = df_data['y'])
```

```
[ ] # fs.identify_missing(missing_threshold=0.01)
```

```
0 features with greater than 0.01 missing values.
```

```
[ ] # fs.identify_single_unique()
```

```
0 features with a single unique value.
```

```
[ ] # fs.identify_collinear(correlation_threshold=0.9)
```

```
0 features with a correlation magnitude greater than 0.90.
```

```
[ ] # fs.identify_zero_importance(task = 'classification', eval_metric = 'auc',
    #                             n_iterations = 10, early_stopping = True)
```

```
Training Gradient Boosting Model

Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[166]   valid_0's auc: 0.96828  valid_0's binary_logloss: 0.0698629
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[147]   valid_0's auc: 0.964939 valid_0's binary_logloss: 0.074618
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[132]   valid_0's auc: 0.955877 valid_0's binary_logloss: 0.0784346
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[127]   valid_0's auc: 0.957627 valid_0's binary_logloss: 0.0761528
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[126]   valid_0's auc: 0.964634 valid_0's binary_logloss: 0.0724945
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[145]   valid_0's auc: 0.960566 valid_0's binary_logloss: 0.0770967
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[125]   valid_0's auc: 0.961834 valid_0's binary_logloss: 0.0720314
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[151]   valid_0's auc: 0.962909 valid_0's binary_logloss: 0.0739902
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[184]   valid_0's auc: 0.968504 valid_0's binary_logloss: 0.0697789
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[172]   valid_0's auc: 0.964429 valid_0's binary_logloss: 0.0721406

0 features with zero importance after one-hot encoding.
```

模型评估

由于本个任务我进行了五组实验，因此先描述模型评估的指标和不同维度。

评估指标

对于预测值（概率）和真实值序列，使用 roc_auc 分数。对于预测值（label）和真实值序列，由于类别不平滑，使用 0.3 和 0.5 不同的阈值进行评测，计算 accuracy, recall, precision 和 f1。

```
[ ] def pre2label(pred_list, threshold):
      return [0 if pred < threshold else 1 for pred in pred_list]
```

```
[ ] def score(labels, pres, threshold=0.5):
      print("roc_auc_score: "+str(roc_auc_score(labels, pres)))
      print("accuracy_score: "+str(accuracy_score(labels, pre2label(pres, threshold))))
      print("recall_score: "+str(recall_score(labels, pre2label(pres, threshold))))
      print("precision_score: "+str(precision_score(labels, pre2label(pres, threshold))))
      print("f1_score: "+str(f1_score(labels, pre2label(pres, threshold))))
```

评估维度

1. 由于本次任务所有实验都使用 10 折 lightgbm, 每次实验的每一折都保存 dev 的预测，并且最后计算 dev 每个指标的评测结果。后续图中标记为 “dev”。

2. 在数据集中，随机取出 1/5 的 与原始数据集同样分布的数据作为测试集，在所有实验中，测试集都没有进行训练，仅用于预测，来评估所有实验模型的泛化能力。后续图中标记为 “test” 。

```
label_0_idxs = df_feature.index[df_feature['y'] == 0].to_list()
label_1_idxs = df_feature.index[df_feature['y'] == 1].to_list()
test_size_0 = 19200
test_size_1 = 800
import random
random.seed(2020)
label_0_test_idxs = random.sample(label_0_idxs, test_size_0)
label_1_test_idxs = random.sample(label_1_idxs, test_size_1)
```

```
for idx in label_0_test_idxs + label_1_test_idxs:
  df_feature.iloc[idx, df_feature.columns.get_loc('type')] = 'test'
```

3. 同时，由于任务要求进行 “全数据集” 上的建模，每个实验的每个模型，都对整个数据集进行了预测（包括之前抽取出的 test），后续图中标记为 “full” 。

模型建模与数据挖掘 （五组实验）

本 section 介绍不同实验（数据挖掘） 的生成方式，下一 section 主要分析，讨论结果。

Baseline 实验

Baseline 实验没有进行任何的数据特征挖掘，直接使用得到的数据，喂入 lightgbm 中，10 折训练，作为 Baseline 模型。后续的实验使用 lightgbm，不同的是输入特征不同，所有模型参数如下。

```
model = lgb.LGBMClassifier(num_leaves=64,
                           max_depth=10,
                           learning_rate=0.1,
                           n_estimators=10000000,
                           subsample=0.8,
                           feature_fraction=0.8,
                           reg_alpha=0.5,
                           reg_lambda=0.5,
                           random_state=seed,
                           metric=None
                           )
```

实验结果 和 特征重要性排行如下

	column	importance
0	scr	820.9
1	dk_amnt(k)	629.6
2	constellation	408.8
3	index	393.2
4	grade	390.6
5	tot_amnt	294.5
6	income	281.7
7	diploma	276.6
8	dk_cnt	223.0
9	occupation	210.7
10	emp_length	170.0
11	home_ownership	122.9
12	car_ownership	109.5
13	location	101.5
14	dq_cnt	63.7
15	gender	32.5

Baseline

```
score(df_oof_train['y'], df_oof_train['pred'], 0.5)
```

```
roc_auc_score: 0.9631984268450245
accuracy_score: 0.9750125
recall_score: 0.5333124804504222
precision_score: 0.7707956600361664
f1_score: 0.6304307635422444
```

```
score(df_oof_train['y'], df_oof_train['pred'], 0.3)
```

```
roc_auc_score: 0.9631984268450245
accuracy_score: 0.972175
recall_score: 0.633406318423522
precision_score: 0.6576810652809354
f1_score: 0.6453154875717017
```

```
score(df_test[ycol], prediction['target'], 0.5)
```

```
roc_auc_score: 0.9625846354166665
accuracy_score: 0.9754
recall_score: 0.52875
precision_score: 0.7862453531598513
f1_score: 0.6322869955156951
```

```
score(df_test[ycol], prediction['target'], 0.3)
```

```
roc_auc_score: 0.9625846354166665
accuracy_score: 0.97285
recall_score: 0.6475
precision_score: 0.6649550706033376
f1_score: 0.6561114629512349
```

```
score(df_train_all[ycol], prediction_train_all['target'], 0.5)
```

```
roc_auc_score: 0.9822388431272205
accuracy_score: 0.97952
recall_score: 0.5884413309982487
precision_score: 0.8537205081669691
f1_score: 0.6966824644549764
```

```
score(df_train_all[ycol], prediction_train_all['target'], 0.3)
```

```
roc_auc_score: 0.9822388431272205
accuracy_score: 0.97841
recall_score: 0.7200400300225169
precision_score: 0.734558448187851
f1_score: 0.7272267845862286
```

Basic Features 实验

即基于之前的特征，添加 手动特征 的实验。根据推测的特征含义，进行手动特征的挖掘。

将 是否拥有房和车 组合，将 职位和收入分桶 组合

```
df_feature['home_car_ownership'] = df_feature[['home_ownership', 'car_ownership']].apply(lambda x: x[0]+x[1]*2, axis=1)
```

```
df_feature['income_occupation'] = df_feature[['income', 'occupation']].apply(lambda x: x[0]*7+x[1], axis=1)
```

将 dk_cnt 和 dq_cnt 两个数量特征 组合

```
df_feature['dk_dq_cnt_ratio'] = df_feature['dk_cnt'] / df_feature['dq_cnt']
df_feature['dk_dq_cnt_sum'] = df_feature['dk_cnt'] + df_feature['dq_cnt']
```

将 dk_amnt 分别于 dk_cnt, income, emp_length 相除

```
df_feature['dk_amnt_per_cnt'] = df_feature['dk_amnt(k)'] / df_feature['dk_cnt']
```

```
df_feature['dk_amnt_income_ratio'] = df_feature['dk_amnt(k)'] / df_feature['income']
```

```
df_feature['dk_amnt_emp_length_ratio'] = df_feature['dk_amnt(k)'] / df_feature['emp_length']
```

然后继续喂入树模型，10 折，
实验结果 和 特征重要性排行如下

	column	importance
0	scr	585.8
1	constellation	396.3
2	dk_amnt_per_cnt	391.2
3	dk_amnt_emp_length_ratio	390.0
4	dk_amnt_income_ratio	370.0
5	grade	308.9
6	index	291.4
7	income_occupation	280.7
8	dk_amnt(k)	260.6
9	diploma	222.6
10	tot_amnt	202.4
11	dk_cnt	170.1
12	occupation	144.7
13	dk_dq_cnt_ratio	136.4
14	home_car_ownership	132.4
15	location	131.7
16	dk_dq_cnt_sum	95.6
17	emp_length	71.0
18	home_ownership	67.2
19	income	64.6
20	car_ownership	30.0
21	dq_cnt	22.2
22	gender	22.2

Basic Features

```
[ ] score(df_oof_train['y'], df_oof_train['pred'], 0.5)
```

```
roc_auc_score: 0.9628956910589479
accuracy_score: 0.9755125
recall_score: 0.5361276196434157
precision_score: 0.782648401826484
f1_score: 0.6363467607202524
```

```
[ ] score(df_oof_train['y'], df_oof_train['pred'], 0.3)
```

```
roc_auc_score: 0.9628956910589479
accuracy_score: 0.9723875
recall_score: 0.6249609008445418
precision_score: 0.6642287234042553
f1_score: 0.6439967767929091
```

```
[ ] score(df_test[ycol], prediction['target'], 0.5)
```

```
roc_auc_score: 0.9621458984375001
accuracy_score: 0.9752
recall_score: 0.5225
precision_score: 0.7857142857142857
f1_score: 0.6276276276276276
```

```
[ ] score(df_test[ycol], prediction['target'], 0.3)
```

```
roc_auc_score: 0.9621458984375001
accuracy_score: 0.97325
recall_score: 0.64
precision_score: 0.6745718050065876
f1_score: 0.6568313021167416
```

```
[ ] score(df_train_all[ycol], prediction_train_all['target'], 0.5)
```

```
roc_auc_score: 0.9849011994665718
accuracy_score: 0.98038
recall_score: 0.6014510883162372
precision_score: 0.86693112152903
f1_score: 0.710192023633678
```

```
[ ] score(df_train_all[ycol], prediction_train_all['target'], 0.3)
```

```
roc_auc_score: 0.9849011994665718
accuracy_score: 0.97992
recall_score: 0.7380535401551164
precision_score: 0.7542827921247762
f1_score: 0.7460799190692969
```

Basic Group 挖掘实验

即基于之前的特征，添加 组合特征挖掘 的实验。

```
[ ] def group_base_mean(df_feature, group, base):
      group_array = df_feature[group].unique()
      f_dict = {}
      for group_element in group_array:
        f_dict[group_element] = df_feature[df_feature[group] == group_element][base].mean()
      df_feature[group+"_"+base+"_mean_diff"] = df_feature[[base, group]].apply(lambda x: x[0]-f_dict[x[1]], axis=1)
```

```
[ ] def group_base_median(df_feature, group, base):
      group_array = df_feature[group].unique()
      f_dict = {}
      for group_element in group_array:
        f_dict[group_element] = df_feature[df_feature[group] == group_element][base].median()
      df_feature[group+"_"+base+"_median_diff"] = df_feature[[base, group]].apply(lambda x: x[0]-f_dict[x[1]], axis=1)
```

```
[ ] def group_base_max(df_feature, group, base):
      group_array = df_feature[group].unique()
      f_dict = {}
      for group_element in group_array:
        f_dict[group_element] = df_feature[df_feature[group] == group_element][base].max()
      df_feature[group+"_"+base+"_max_diff"] = df_feature[[base, group]].apply(lambda x: x[0]-f_dict[x[1]], axis=1)
```

```
[ ] def group_base_min(df_feature, group, base):
      group_array = df_feature[group].unique()
      f_dict = {}
      for group_element in group_array:
        f_dict[group_element] = df_feature[df_feature[group] == group_element][base].min()
      df_feature[group+"_"+base+"_min_diff"] = df_feature[[base, group]].apply(lambda x: x[0]-f_dict[x[1]], axis=1)
```

```
[ ] for group in ['constellation', 'grade', 'index', 'occupation', 'diploma']:
      for base in ['scr', 'dk_cnt', 'tot_amnt', 'income', 'dk_amnt(k)', 'emp_length', 'dq_cnt']:
        for f in [group_base_mean, group_base_median, group_base_max, group_base_min]:
          f(df_feature, group, base)
```

如上所示，对每一种类别特征，在每一个定量特征上， 进行 于组内 max 值， min 值， median 值，mean 值的差距的特征构造。

所有特征，再次喂入之前的树模型中。10 折训练。
最终结果和重要度排名（前 30）如下所示。

	column	importance
0	grade	220.3
1	index	206.8
2	dk_amnt_income_ratio	180.4
3	dk_amnt_per_cnt	173.3
4	constellation	171.6
5	dk_amnt_emp_length_ratio	166.5
6	diploma	154.2
7	occupation_tot_amnt_mean_diff	144.0
8	constellation_tot_amnt_mean_diff	138.5
9	occupation_dk_cnt_mean_diff	134.2
10	constellation_dk_cnt_mean_diff	119.2
11	constellation_emp_length_mean_diff	111.9
12	index_income_mean_diff	111.3
13	home_car_ownership	110.1
14	constellation_income_mean_diff	103.5
15	constellation_dq_cnt_mean_diff	100.6
16	location	94.4
17	index_dk_cnt_mean_diff	88.8
18	occupation_income_mean_diff	88.2
19	index_tot_amnt_mean_diff	87.8
20	index_dq_cnt_mean_diff	85.4
21	grade_scr_median_diff	82.9
22	income_occupation	81.0
23	occupation_emp_length_mean_diff	80.0
24	index_emp_length_mean_diff	79.9
25	grade_emp_length_mean_diff	77.1
26	grade_dk_cnt_mean_diff	76.2
27	grade_scr_mean_diff	75.4
28	grade_income_mean_diff	72.9
29	index_scr_mean_diff	70.2
30	occupation_dq_cnt_mean_diff	66.8

After Data mining (Basic Group)

```
[ ] score(df_oof_train['y'], df_oof_train['pred'], 0.5)

roc_auc_score: 0.9623005151955559
accuracy_score: 0.975175
recall_score: 0.5351892399124178
precision_score: 0.7738579828132067
f1_score: 0.632766272189349
```

```
[ ] score(df_oof_train['y'], df_oof_train['pred'], 0.3)

roc_auc_score: 0.9623005151955559
accuracy_score: 0.9725
recall_score: 0.6362214576165155
precision_score: 0.6623249755779876
f1_score: 0.6490108487555839
```

```
[ ] score(df_test[ycol], prediction['target'], 0.5)

roc_auc_score: 0.9621179687500001
accuracy_score: 0.97535
recall_score: 0.53
precision_score: 0.7837338262476895
f1_score: 0.6323639075316928
```

```
[ ] score(df_test[ycol], prediction['target'], 0.3)

roc_auc_score: 0.9621179687500001
accuracy_score: 0.97255
recall_score: 0.6375
precision_score: 0.6631989596879063
f1_score: 0.6500956022944551
```

```
[ ] score(df_train_all[ycol], prediction_train_all['target'], 0.5)

roc_auc_score: 0.9875234775195488
accuracy_score: 0.98197
recall_score: 0.6307230422817113
precision_score: 0.8851825842696629
f1_score: 0.7365960555149744
```

```
[ ] score(df_train_all[ycol], prediction_train_all['target'], 0.3)

roc_auc_score: 0.9875234775195488
accuracy_score: 0.9817
recall_score: 0.7713284963722792
precision_score: 0.770942735683921
f1_score: 0.7711355677838919
```

Scr Group 挖掘实验

即基于之前的特征，添加 基于 scr 分桶的组合特征挖掘 的实验。

```
df_feature['scr_bin'] = pd.cut(df_feature['scr'], bins=5, labels=['scr_bin_0', 'scr_bin_1', 'scr_bin_2', 'scr_bin_3', 'scr_bin_4'])
```

```
for group in ['scr_bin']:
  for base in ['scr', 'diploma', 'constellation', 'grade', 'index', 'dk_cnt', 'tot_amnt', 'income', 'occupation', 'dk_amnt(k)', 'emp_length', 'dq_cnt']:
    for f in [group_base_mean, group_base_median, group_base_max, group_base_min]:
      f(df_feature, group, base)
```

```
df_feature['scr_bin'] = lbl.fit_transform(df_feature['scr_bin'])
```

如上图所示，对 scr 值进行分桶处理，并且在其他特征上做组合特征。

所有特征，再次喂入之前的树模型中。10 折训练。
最终结果和重要度排名（前 30）如下所示

	column	importance
0	scr_bin_index_mean_diff	134.2
1	dk_amnt_income_ratio	132.4
2	grade	131.2
3	dk_amnt_emp_length_ratio	127.1
4	dk_amnt_per_cnt	126.7
5	scr_bin_grade_mean_diff	121.5
6	scr_bin_constellation_mean_diff	120.7
7	diploma	111.7
8	occupation_tot_amnt_mean_diff	111.4
9	occupation_dk_cnt_mean_diff	110.3
10	home_car_ownership	109.0
11	constellation_tot_amnt_mean_diff	100.6
12	index	99.9
13	constellation_dk_cnt_mean_diff	96.4
14	location	93.6
15	scr_bin_diploma_mean_diff	92.2
16	constellation_income_mean_diff	91.4
17	scr_bin_scr_mean_diff	88.9
18	index_income_mean_diff	85.7
19	constellation_dq_cnt_mean_diff	85.0
20	scr_bin_scr_max_diff	84.8
21	index_tot_amnt_mean_diff	81.9
22	scr_bin_scr_median_diff	79.4
23	constellation	75.9
24	constellation_emp_length_mean_diff	70.5
25	index_emp_length_mean_diff	69.2
26	occupation_income_mean_diff	69.0
27	income_occupation	68.6
28	scr_bin_occupation_mean_diff	65.8
29	index_dk_cnt_mean_diff	63.7
30	grade_scr_median_diff	59.2

After Data mining (scr Group)

```
[ ] score(df_oof_train['y'], df_oof_train['pred'], 0.5)
```

```
roc_auc_score: 0.962000514207119
accuracy_score: 0.9755125
recall_score: 0.5386299655927432
precision_score: 0.7805983680870353
f1_score: 0.637423653525819
```

```
[ ] score(df_oof_train['y'], df_oof_train['pred'], 0.3)
```

```
roc_auc_score: 0.962000514207119
accuracy_score: 0.9724375
recall_score: 0.633406318423522
precision_score: 0.6621975147155004
f1_score: 0.6474820143884892
```

```
[ ] score(df_test[ycol], prediction['target'], 0.5)
```

```
roc_auc_score: 0.9618130859374998
accuracy_score: 0.97565
recall_score: 0.53125
precision_score: 0.7914338919925512
f1_score: 0.6357516828721017
```

```
[ ] score(df_test[ycol], prediction['target'], 0.3)
```

```
roc_auc_score: 0.9618130859374998
accuracy_score: 0.97225
recall_score: 0.6325
precision_score: 0.6597131681877445
f1_score: 0.6458200382897256
```

```
[ ] score(df_train_all[ycol], prediction_train_all['target'], 0.5)
```

```
roc_auc_score: 0.9874500288932938
accuracy_score: 0.98202
recall_score: 0.6314736052039029
precision_score: 0.8859248859248859
f1_score: 0.7373648846041485
```

```
[ ] score(df_train_all[ycol], prediction_train_all['target'], 0.3)
```

```
roc_auc_score: 0.9874500288932938
accuracy_score: 0.98189
recall_score: 0.7718288716537403
precision_score: 0.7743473895582329
f1_score: 0.7730860794386667
```

dk amnt Group 挖掘实验

即基于之前的特征，添加 基于 dk amnt 分桶的组合特征挖掘 的实验。

```
df_feature['dk_amnt(k)_bin'] = pd.cut(df_feature['dk_amnt(k)'], bins=10, labels=['dk_amnt(k)_bin_0', 'dk_amnt(k)_bin_1', 'dk_amnt(k)_bin_2', 'dk_amnt(k)_bin_3',
                                                                                  'dk_amnt(k)_bin_4', 'dk_amnt(k)_bin_5', 'dk_amnt(k)_bin_6', 'dk_amnt(k)_bin_7',
                                                                                  'dk_amnt(k)_bin_8', 'dk_amnt(k)_bin_9'])

for group in ['dk_amnt(k)_bin']:
  for base in ['scr', 'diploma', 'constellation', 'grade', 'index', 'dk_cnt', 'tot_amnt', 'income', 'occupation', 'dk_amnt(k)', 'emp_length', 'dq_cnt']:
    for f in [group_base_mean, group_base_median, group_base_max, group_base_min]:
      f(df_feature, group, base)
```

```
df_feature['dk_amnt(k)_bin'] = lbl.fit_transform(df_feature['dk_amnt(k)_bin'])
```

如上图所示，对 dk amnt 值进行分桶处理，并且在其他特征上做组合特征。

所有特征，再次喂入之前的树模型中。10 折训练。
最终结果和重要度排名（前 30）如下所示

	column	importance
0	dk_amnt(k)_bin_grade_mean_diff	186.3
1	dk_amnt(k)_bin_index_mean_diff	180.3
2	dk_amnt(k)_bin_diploma_mean_diff	157.9
3	dk_amnt(k)_bin_constellation_mean_diff	137.8
4	dk_amnt(k)_bin_dk_amnt(k)_mean_diff	137.1
5	dk_amnt_income_ratio	97.5
6	dk_amnt(k)_bin_dk_cnt_mean_diff	95.9
7	occupation_tot_amnt_mean_diff	94.6
8	dk_amnt_emp_length_ratio	93.7
9	home_car_ownership	92.0
10	dk_amnt(k)_bin_tot_amnt_mean_diff	92.0
11	dk_amnt_per_cnt	89.7
12	constellation_tot_amnt_mean_diff	87.7
13	location	87.2
14	occupation_dk_cnt_mean_diff	83.6
15	constellation_dk_cnt_mean_diff	82.3
16	dk_amnt(k)_bin_occupation_mean_diff	81.3
17	index_income_mean_diff	75.7
18	scr_bin_scr_mean_diff	74.5
19	scr_bin_constellation_mean_diff	73.5
20	grade	71.5
21	constellation_income_mean_diff	70.8
22	scr_bin_scr_max_diff	69.4
23	scr_bin_scr_median_diff	68.3
24	scr_bin_index_mean_diff	67.5
25	constellation_dq_cnt_mean_diff	65.7
26	constellation_emp_length_mean_diff	63.3
27	scr_bin_grade_mean_diff	61.1
28	income_occupation	57.4
29	index_tot_amnt_mean_diff	57.3
30	scr_bin_diploma_mean_diff	56.5

After Data mining (dk amnt Group)

```
score(df_oof_train['y'], df_oof_train['pred'], 0.5)
```

```
roc_auc_score: 0.9626019212550064
accuracy_score: 0.9748875
recall_score: 0.529246168282765
precision_score: 0.7704918032786885
f1_score: 0.6274800667531986
```

```
score(df_oof_train['y'], df_oof_train['pred'], 0.3)
```

```
roc_auc_score: 0.9626019212550064
accuracy_score: 0.9721875
recall_score: 0.6312167657178605
precision_score: 0.6586161879895561
f1_score: 0.6446254591918222
```

```
score(df_test[ycol], prediction['target'], 0.5)
```

```
roc_auc_score: 0.9621425130208333
accuracy_score: 0.9752
recall_score: 0.52875
precision_score: 0.7804428044280443
f1_score: 0.6304023845007451
```

```
score(df_test[ycol], prediction['target'], 0.3)
```

```
roc_auc_score: 0.9621425130208333
accuracy_score: 0.97235
recall_score: 0.6325
precision_score: 0.661437908496732
f1_score: 0.6466453674121405
```

```
score(df_train_all[ycol], prediction_train_all['target'], 0.5)
```

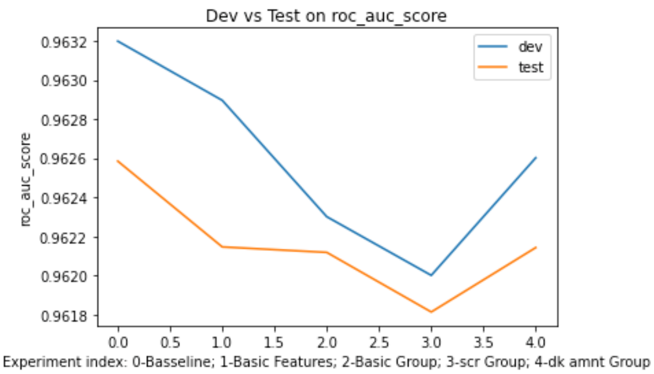
```
roc_auc_score: 0.9882003885443795
accuracy_score: 0.9826
recall_score: 0.6452339254440831
precision_score: 0.8890037917959325
f1_score: 0.7477529718759061
```

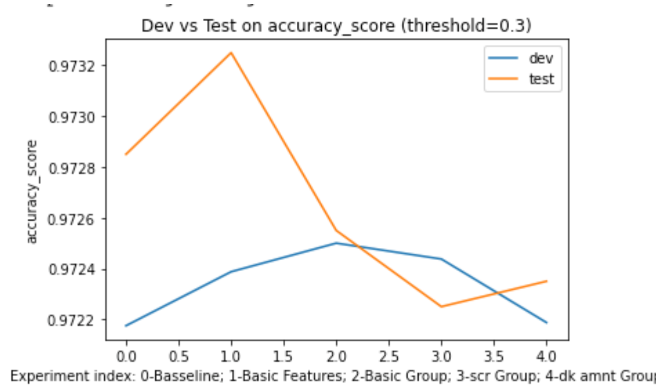
```
score(df_train_all[ycol], prediction_train_all['target'], 0.3)
```

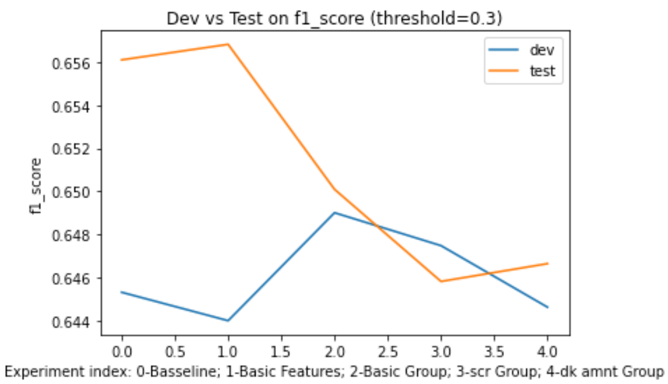
```
roc_auc_score: 0.9882003885443795
accuracy_score: 0.98264
recall_score: 0.7823367525644234
precision_score: 0.7831204608064112
f1_score: 0.7827284105131415
```

讨论与总结

上述五组实验，对于 Dev 和 Test 在 ROC_AUC，Acc，F1 上对比如下

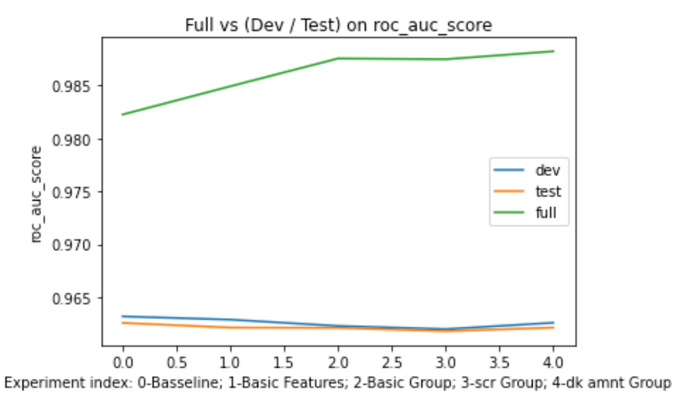


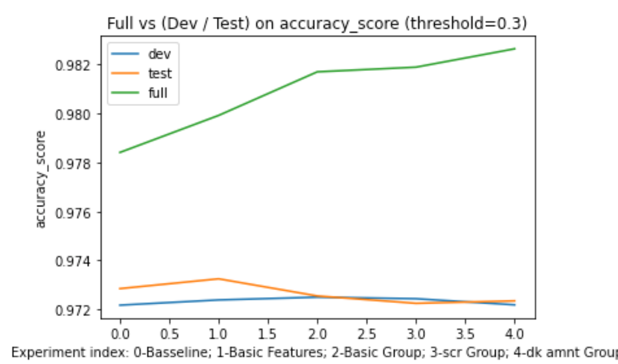


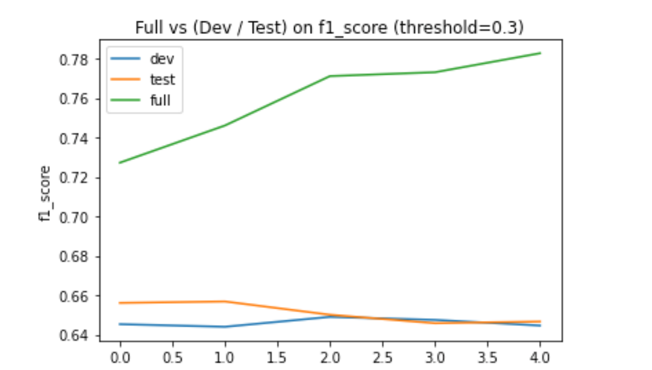


可以看出，深度挖掘数据以后，ROC_AUC 不如 baseline，Acc 和 F1 除了在 Basic Features 手动挖掘有所提升外，其他泛化性能都不在降低。

而加入全数据集评测的对比如下







可以看出，在深度挖掘之后，模型在整个数据集上的表征性能增强，导致其分类能力在不同性能上相比 baseline 大幅提升。

- 因此在本次实验中，针对模型泛化能力 (在 unseen 训练的测试集上 (random 取 0.2)，阈值取 0.3):
    - Baseline 模型分数为，ROC_AUC 为 0.9626，Acc 为 0.9729，Recall 为 0.6475，Precision 为 0.6650，F1 为 0.6561
    - Basic Features （最优）模型分数为，ROC_AUC 为 0.9621，Acc 为 0.9733，Recall 为 0.64，Precision 为 0.6745，F1 为 0.6568
- 因此在本次实验中，针对全数据集上分类能力 (阈值取 0.3):
    - Baseline 模型分数为，ROC_AUC 为 0.9822，Acc 为 0.9784，Recall 为 0.72，Precision 为 0.7346，F1 为 0.7272
    - dk amnt Group（最优）模型分数为，ROC_AUC 为 0.9882，Acc 为 0.9826，Recall 为 0.7823，Precision 为 0.7831，F1 为 0.7827