二〇一九年一月二日星期三

上午10时28分

元旦节后开工。

早上提交了工程师转助研申请。

开始元旦假期中自己的实验假设，即基于静态的26JS与33维度的ATF的分析，主要分析对象为CERT5.2用户。

基于静态分析假设，更容易调整参数得到最好的结果。

先基于原有的26JS维度与新的33维度的ATF进行研究吧！

下午3时36分

分别按照传统的JS与ATF进行实验，首先，进行KMeans分析，可以得到：

26JS: [-1]用户可以分成2个群簇，分别包含

0 1564

1 197 （显然应该是我们特别关注的部分）

而使用33维度的ATF实验，有 针对ATF\_lst的

最佳自动K值为[2:10] 2 : 0.36697501148725215

0 197 个

1 1564 个

下面继续使用后续的ATF聚类结果进行实验，并且默认采用1564个作为训练集，剩余197个[-1]+[+1]作为测试集输出结果；

计算召回率与FPR

197/1564=0.12...

下面开始记录针对ATF特征的预测结果：

自动PCA（）

[**0.05**, 0.9, 0.47005241751310445, 0.5362318840579711, **0.12556278139069535**, 0.6651270207852193, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, **random\_state=15**, shrinking=True, tol=0.01,

verbose=False), **0.5172413793103449, 0.6, 0.4**]

[**0.01**, 1.0, 0.5362318840579711, 0.5362318840579711, **0.128064032016008**, 0.6766743648960739, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.01, **random\_state=15**, shrinking=True, tol=0.01,

verbose=False), **0.5172413793103449, 0.6, 0.4**]

[0.01, 1.0, 0.5362318840579711, 0.5362318840579711, **0.128064032016008**, 0.6766743648960739, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.01, **random\_state=0**, shrinking=True, tol=0.01,

verbose=False), **0.5172413793103449, 0.6, 0.4**]

[0.05, 1.0, 0.5362318840579711, 0.5362318840579711, 0.12556278139069535, 0.6651270207852193, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, **random\_state=5**, shrinking=True, tol=0.01,

verbose=False), **0.5172413793103449, 0.6, 0.4**]

[0.05, 1.0, 0.5362318840579711, 0.5362318840579711, 0.12556278139069535, 0.6651270207852193, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, **random\_state=10**, shrinking=True, tol=0.01,

verbose=False), **0.5172413793103449, 0.6, 0.4**]

[0.05, 1.0, 0.5362318840579711, 0.5362318840579711, 0.12556278139069535, 0.6651270207852193, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, **random\_state=100**, shrinking=True, tol=0.01,

verbose=False), **0.5172413793103449, 0.6, 0.4**]

下面开始调整PCA的维度

PCA=3

[0.05, 1.0, 0.6811594202898551, 0.6811594202898551, 0.13356678339169584, 0.7251732101616628, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, **random\_state=100**, shrinking=True, tol=0.01,

verbose=False), **0.6551724137931034, 0.6, 1.0**]

[0.05, 1.0, 0.6811594202898551, 0.6811594202898551, 0.13356678339169584, 0.7251732101616628, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, **random\_state=0**, shrinking=True, tol=0.01,

verbose=False), **0.6551724137931034, 0.6, 1.0**]

PCA=1

*[0.05, 1.0, 0.8695652173913043, 0.8695652173913043, 0.17588204318062137, 0.19709854927463732, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',*

*max\_iter=-1, nu=0.05, random\_state=10, shrinking=True, tol=0.01,*

*verbose=False), 0.7931034482758621, 0.9, 1.0]*

[0.05, 1.0, 0.8695652173913043, 0.8695652173913043, 0.16708354177088544, 0.9099307159353349, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, **random\_state=10**, shrinking=True, tol=0.01,

verbose=False), **0.7931034482758621, 0.9, 1.0**]

[0.15, 0.9, 0.6888400722100181, **0.782608695652174**, 0.15507753876938468, 0.8406466512702079, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.15, random\_state=10, shrinking=True, tol=0.01,

verbose=False), **0.7586206896551724, 0.7666666666666667, 0.9**]

PCA=2

[0.05, 1.0, 0.7971014492753623, 0.7971014492753623, 0.14357178589294647, 0.789838337182448, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, **random\_state=10**, shrinking=True, tol=0.01,

verbose=False), **0.7931034482758621, 0.7666666666666667, 0.9**]

[0.05, 1.0, 0.8260869565217391, 0.8260869565217391, 0.1535767883941971, 0.8406466512702079, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto',

kernel='linear', max\_iter=-1, nu=0.05, random\_state=10,

shrinking=True, tol=0.01, verbose=False), 0.7931034482758621, 0.8, 1.0]

今天下午最大的结果就是，通过PCA=1的KMeans+OCSVM 可以得到了recall基本满意，FPR=0.167的结果

二〇一九年一月三日星期四

下午3时34分

今天下午继续昨天的实验，上午终于把此次工程师转助研的手续搞定。

昨天最好结果的实验中，分析DF函数的排序关系：一共433个用户，一半是：217

**Insiders\_1 in Risk\_Users\_Sort:...**

['KEW0198', 1, 1, '0.37430310077277085'] : 62

['DAS1320', -1, 1, '-22.23026921734859'] : 431

['GFM1815', 1, 1, '0.014400357126774566'] : 284

['EPG1196', 1, 1, '0.005637024245942257'] : 341

['KBC1390', -1, 1, '-0.0018274702185152591'] : 413

['PBC0077', -1, 1, '-0.0015464048043298817'] : 412

['SAF1942', -1, 1, '-22.14763131692173'] : 430

['ALT1465', 1, 1, '0.026804458461665348'] : 190

['SLL0193', 1, 1, '0.011109562879745738'] : 313

['IHC0561', 1, 1, '0.6022116960271156'] : 21

['JKB0287', 1, 1, '0.02678587283564937'] : 192

['DNJ0740', 1, 1, '0.004218641270885115'] : 353

['MIB0203', -1, 1, '-0.7209941744624366'] : 419

['REF1924', 1, 1, '0.015845453967635592'] : 273

['AYG1697', -1, 1, '-0.17277538219780197'] : 417

['ISW0738', 1, 1, '0.017838005871674056'] : 253

['FZG0389', 1, 1, '0.6268173201910905'] : 8

['ZKP0542', 1, 1, '0.0024564748586293206'] : 365

['PTH0005', 1, 1, '0.07336473924896225'] : 165

['ALW0764', 1, 1, '0.01843704214893549'] : 251

['ELT1370', 1, 1, '0.018709174248449756'] : 248

['NIV1608', 1, 1, '0.03132570395781187'] : 185

['JUP1472', 1, 1, '0.5830946344728041'] : 26

['WHB1247', 1, 1, '0.0266964563180494'] : 196

['TMC0934', 1, 1, '0.009238971102202953'] : 323

['WSK1857', 1, 1, '0.021208592547502292'] : 236

['ETW0002', 1, 1, '0.012960862896708392'] : 299

['WWW0701', 1, 1, '0.0006780833208104298'] : 390

['VAH1292', 1, 1, '0.6009325182006826'] : 23

**Insiders\_2 in Risk\_Users\_Sort:... 一半是217 如果严格按照217取一半，则只有16/30的召回率**

*['VCF1602', 1, 1, '0.0012927582339194998'] : 381*

['CKP0630', 1, 1, '0.12537746087586754'] : 139

['ZIE0741', 1, 1, '0.1621446298423379'] : 104

*['SIS0042', -1, 1, '-0.0019075955446155035'] : 415*

['TNB1616', 1, 1, '0.0147230069068911'] : 280

['TRC1838', 1, 1, '0.1733531135544517'] : 101

['MDS0680', 1, 1, '0.06374327823238346'] : 169

['WDT1634', 1, 1, '0.009124014033783823'] : 324

['OSS1463', 1, 1, '0.014395350212229374'] : 285

['CIF1430', 1, 1, '0.005506677019958772'] : 342

['MCP0611', 1, 1, '0.2563333269471464'] : 76

['CHP1711', 1, 1, '0.0552890363477907'] : 175

['GWG0497', 1, 1, '0.0618355714999268'] : 170

['KSS1005', 1, 1, '0.0021440004798805035'] : 366

['NAH1366', -1, 1, '-1.4854964916386066'] : 420 （全勤）

['RRS0056', 1, 1, '0.11198917020182719'] : 145

['ICB1354', 1, 1, '0.18827216216679687'] : 97

['BYO1846', 1, 1, '0.009371152081886436'] : 322

['HXP0976', 1, 1, '0.4857005567121533'] : 35

['HMS1658', 1, 1, '0.4246631586547167'] : 50

['HIS1394', 1, 1, '0.015579900707372474'] : 275

['LVF1626', 1, 1, '0.6099363019644386'] : 17

['MGB1235', 1, 1, '0.026757305975312562'] : 195

['DCC1119', 1, 1, '0.02350410810993253'] : 224

['SNK1280', -1, 1, '-0.00031667240468991054'] : 395

['ITA0159', 1, 1, '0.6138104688498807'] : 15

['JAL0811', 1, 1, '0.11014801702344101'] : 146

['OKM1092', 1, 1, '0.005071790815957655'] : 348

['HSN0675', 1, 1, '0.0005269671761425343'] : 391

['TMT0851', 1, 1, '0.0016731237254745679'] : 371

**Insiders\_3 in Risk\_Users\_Sort:...**

['MPF0690', 1, 1, '0.3594052219463215'] : 63

['CRD0272', 1, 1, '0.02171323050862739'] : 235

['VRP0267', 1, 1, '0.024757288365549357'] : 216

['ELM1123', 1, 1, '0.0036080763461114884'] : 359

['GKW0043', 1, 1, '0.026772242315434625'] : 194

['ACA1126', 1, 1, '0.026458459631438558'] : 203

['KCM0466', 1, 1, '0.13424506282532533'] : 135

['ZEH0685', 1, 1, '0.25570502196787004'] : 77

['LAH0463', 1, 1, '0.6248992267356286'] : 11

['CWW1120', 1, 1, '0.012644522112569945'] : 302

从上图中分析，如果以排序302作为分界线，即高于302之后的用户不再考虑，那么对于Insiders2而言Recall变化为(16+4)/30， Insiders3的Recall变化为9/10-->0.9，Insiders1的Recall为17/30,

而此时有302个用户则

Recall:

FPR: (302-46) / 2000 = 0.128 --- > 目标：FPR: 0.05-0.09

目前KMeans+OCSVM的最好结果PCA=1，

如果直接使用所有DF的中位数，即取前一半的RiskUsers，则有Insiders\_3的召回率迅速调到0.7，整体Risk比例为216 / 2000.0 = 0.108

Risk\_1结果中，433个测试用户中竟然识别Risk达到394个，其中误报太多了！

PCA=1

*[0.05, 1.0, 0.8695652173913043, 0.8695652173913043, 0.17588204318062137, 0.19709854927463732, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',*

*max\_iter=-1, nu=0.05, random\_state=10, shrinking=True, tol=0.01,*

*verbose=False), 0.7931034482758621, 0.9, 1.0]*

验证

Risk\_1\_ATF文件：

**HBW0057**,35.0,33.0,43.0,44.0,29.0,-16.99,-32.84,20.47,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,-12.26,-16.2,13.2053177459,0.293451505464,0.254693593253,0.00565985762785,-0.261707988981,5.02222222222,12.7555555556,428930.066327,136300.475979,3.46666666667,11.5777777778,2.97777777778,5.08888888889,5.91111111111,5.0,9.0,343.0,**0.0167852630153,**

**HBW0057,1,-1,0.016785263015268725,**

**验证通过！**

或许，在中低满意度群体上，出勤率可以作为一个有效的进一步筛选的表示器

初步对Risk\_1中判定为Risk的用户继续做自动KMeans，则有

Insider\_3: 9个一类，1个另一类

Insiders\_2: 除去3个未检测出，剩余又丢掉5个，总计检测22个一类

Insiders\_1中也有5个分在一类，剩余19个在另一类

356

38

如果把38个的一类去掉，变成356个Risk，占比17.8%， 且Recall降低明显

Insiders\_2的Recall从0.9-->0.73

Insiders\_3的Recall从1.0-->0.9

二〇一九年一月四日星期五

下午12时42分

下午2时46分

昨天对于Risk\_1的用户进行KMeans效果并不好，Recall降低且FPR降低有限。故我们今天尝试从late\_days, early\_days, cnt\_days来进行分析，主要依据scale,minmax两种依据，查看Insiders1/2/3的结果

先来看Insiders\_3的结果：

全周期的Late/Early占比约在0.2-0.4之间

insider\_3 MPF0690 [0.3898305084745763, 0.1864406779661017, 118.0] [0.38983051 0.36269334 0.20210526] [ 0.59505882 -0.34835138 -1.4560034 ]

insider\_3 CRD0272 [0.34057971014492755, 0.12318840579710146, 138.0] [0.34057971 0.2396452 0.24421053] [ 0.34745024 -0.66968901 -1.2729927 ]

insider\_3 VRP0267 [0.37089201877934275, 0.14553990610328638, 213.0] [0.37089202 0.28312681 0.40210526] [ 0.49984549 -0.5561377 -0.58670257]

insider\_3 ELM1123 [0.4527363184079602, 0.12437810945273632, 201.0] [0.45273632 0.2419596 0.37684211] [ 0.91131802 -0.66364501 -0.69650899]

insider\_3 GKW0043 [0.4027777777777778, 0.14814814814814814, 216.0] [0.40277778 0.28820077 0.40842105] [ 0.66015126 -0.54288717 -0.55925097]

insider\_3 ACA1126 [0.36324786324786323, 0.1282051282051282, 234.0] [0.36324786 0.24940451 0.44631579] [ 0.46141447 -0.64420278 -0.39454134]

insider\_3 KCM0466 [0.3665480427046263, 0.12099644128113879, 281.0] [0.36654804 0.23538106 0.54526316] [ 0.47800613 -0.68082474 0.0355338 ]

insider\_3 ZEH0685 [0.36908517350157727, 0.1608832807570978, 317.0] [0.36908517 0.31297513 0.62105263] [ 0.49076157 -0.47818946 0.36495306]

insider\_3 LAH0463 [0.37941176470588234, 0.1588235294117647, 340.0] [0.37941176 0.30896818 0.66947368] [ 0.54267854 -0.48865352 0.57541537]

insider\_3 CWW1120 [0.3710144927536232, 0.11884057971014493, 345.0] [0.37101449 0.23118714 0.68 ] [ 0.50046122 -0.69177707 0.62116804]

再来看看Insiders\_2

如果单独看缺勤率比例，则Insiders\_2中至少有12/27的用户会被筛选出去；

insider\_2 VCF1602 [0.36492890995260663, 0.45023696682464454, 211.0] [0.36492891 0.87587082 0.39789474] [ 0.46986593 0.99180082 -0.60500364]

insider\_2 CKP0630 [0.10843373493975904, 0.09036144578313253, 166.0] [0.10843373 0.17578511 0.30315789] [-0.81966454 -0.83645831 -1.01677772]

insider\_2 ZIE0741 [0.39520958083832336, 0.3592814371257485, 167.0] [0.39520958 0.69893001 0.30526316] [ 0.62210212 0.52972361 -1.00762718]

insider\_2 TNB1616 [0.36312849162011174, 0.4134078212290503, 179.0] [0.36312849 0.80422505 0.33052632] [ 0.46081432 0.8046994 -0.89782076]

insider\_2 TRC1838 [0.0, 0.0, 179.0] [0. 0. 0.33052632] [-1.36481555 -1.29551742 -0.89782076]

insider\_2 MDS0680 [0.0, 0.0, 181.0] [0. 0. 0.33473684] [-1.36481555 -1.29551742 -0.87951969]

insider\_2 WDT1634 [0.5, 0.47802197802197804, 182.0] [0.5 0.92992254 0.33684211] [ 1.14893636 1.13295574 -0.87036916]

insider\_2 OSS1463 [0.47549019607843135, 0.45588235294117646, 204.0] [0.4754902 0.8868531 0.38315789] [ 1.02571322 1.02048082 -0.66905739]

insider\_2 CIF1430 [0.43243243243243246, 0.3621621621621622, 185.0] [0.43243243 0.70453404 0.34315789] [ 0.80924015 0.54435842 -0.84291755]

insider\_2 MCP0611 [0.0, 0.00510204081632653, 196.0] [0. 0.00992528 0.36631579] [-1.36481555 -1.26959776 -0.74226167]

insider\_2 CHP1711 [0.0, 0.0, 199.0] [0. 0. 0.37263158] [-1.36481555 -1.29551742 -0.71481006]

insider\_2 GWG0497 [0.46766169154228854, 0.4527363184079602, 201.0] [0.46766169 0.88073295 0.37684211] [ 0.98635539 1.00449816 -0.69650899]

insider\_2 KSS1005 [0.011857707509881422, 0.05138339920948617, 253.0] [0.01185771 0.09995896 0.48631579] [-1.30520088 -1.03447669 -0.22068118]

insider\_2 RRS0056 [0.0, 0.0, 239.0] [0. 0. 0.45684211] [-1.36481555 -1.29551742 -0.34878867]

insider\_2 ICB1354 [0.0, 0.0, 242.0] [0. 0. 0.46315789] [-1.36481555 -1.29551742 -0.32133706]

insider\_2 BYO1846 [0.004132231404958678, 0.01652892561983471, 242.0] [0.00413223 0.03215463 0.46315789] [-1.34404075 -1.21154628 -0.32133706]

insider\_2 HXP0976 [0.46938775510204084, 0.3795918367346939, 245.0] [0.46938776 0.73844095 0.46947368] [ 0.99503318 0.6329056 -0.29388546]

insider\_2 HMS1658 [0.41832669322709165, 0.47410358565737054, 251.0] [0.41832669 0.92229987 0.48210526] [ 0.73832349 1.11304931 -0.23898225]

insider\_2 HIS1394 [0.350597609561753, 0.3545816733067729, 251.0] [0.35059761 0.6897873 0.48210526] [ 0.39781527 0.50584761 -0.23898225]

insider\_2 LVF1626 [0.4521072796934866, 0.45977011494252873, 261.0] [0.45210728 0.89441618 0.50315789] [ 0.90815552 1.04023164 -0.1474769 ]

insider\_2 MGB1235 [0.0037593984962406013, 0.0037593984962406013, 266.0] [0.0037594 0.00731337 0.51368421] [-1.34591516 -1.27641872 -0.10172422]

insider\_2 DCC1119 [0.3754646840148699, 0.14869888475836432, 269.0] [0.37546468 0.28927215 0.52 ] [ 0.52283458 -0.54008929 -0.07427262]

insider\_2 ITA0159 [0.0, 0.0, 285.0] [0. 0. 0.55368421] [-1.36481555 -1.29551742 0.07213594]

insider\_2 JAL0811 [0.010309278350515464, 0.030927835051546393, 291.0] [0.01030928 0.06016562 0.56631579] [-1.31298562 -1.13839616 0.12703915]

insider\_2 OKM1092 [0.4208955223880597, 0.4537313432835821, 335.0] [0.42089552 0.88266862 0.65894737] [0.75123829 1.00955314 0.52966269]

insider\_2 HSN0675 [0.0029850746268656717, 0.005970149253731343, 335.0] [0.00298507 0.01161406 0.65894737] [-1.34980808 -1.26518754 0.52966269]

insider\_2 TMT0851 [0.3699421965317919, 0.3786127167630058, 346.0] [0.3699422 0.73653621 0.68210526] [0.49507025 0.62793142 0.63031858]

如果我们直接定义成



则对于Insiders\_3而言，全部用户的LED指标都在0.3以上，准确的说是在0.3-0.4之间

对于Insiders\_2而言，

如果以0.30作为LED的下限，那么Risk\_1中有0.68的用户识别为Risk\_2，其中只有Insiders\_3作为Recall=1，此时

Risk\_2用户个数为: 248.0 0.629441624365 0.124 （Risk比例）

突然想到，如果使用不包含OCEAN的ATF数据呢？结果一样。

突然灵机一动，通过重新修改整体的使用过程，即KMeans选出的用户群簇用来训练OCSVM后直接测试Leave\_Users，依旧是PCA=1时，原始的Static\_ATF文件

[0.05, 1.0, 0.8695652173913043, 0.8695652173913043, 0.0832016850974197, 0.10905452726363181, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, random\_state=10, shrinking=True, tol=0.01,

verbose=False), 0.7931034482758621, 0.9, 1.0

如此，FPR终于降到了0.08左右，而Recall则达到了最好的0.79/0.9/1.0

剩下的只需要使用DF的MinMax值进行预测即可，正常用户默认是0，Risk用户的DF值为+，越大威胁越高；

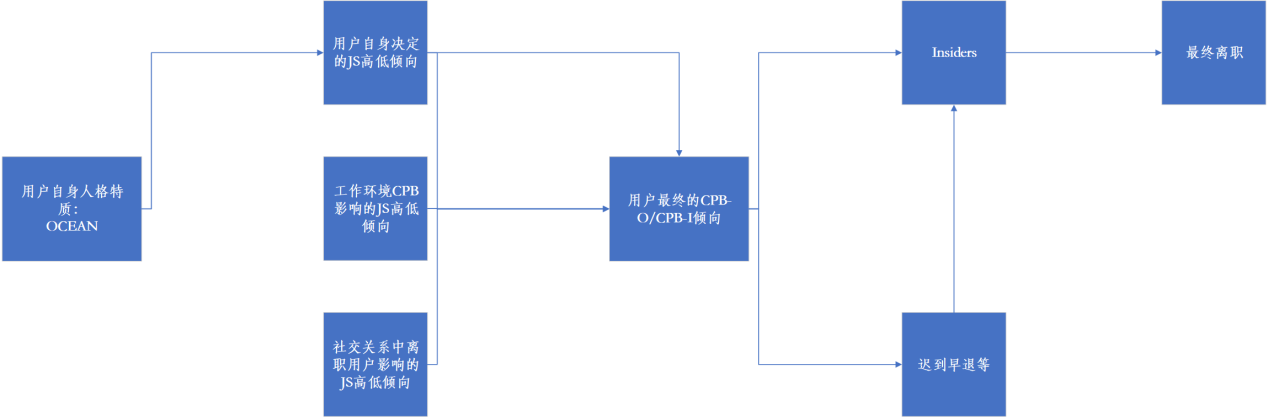
此时的238个LeaveUsers中有218个用户识别为满意度低用户，剩余20个用户为中等满意度；验证了我们的猜想：工作低满意度最终导致的离职，但是离职不一定原因是低满意度；

二〇一九年一月五日星期六

下午7时35分

2019年1月第1周，预感即将对于毕业论文中的主观建模部分有个交代。

目前的模型情况：



上图中，建模的重点是用户的反生产行为CPB作为最终Insider Risk风险，而CPB来源于自身的OCEAN特质的影响与工作满意度的间接影响，最终就是如图中三类JS间接影响因素+OCEAN的直接影响因素共同决定了用户的CPB倾向；而CPB倾向的高低又导致了内部破坏行为与轻度的迟到早退（缺勤发展成内部破坏），最终在实施内部破坏攻击后离职；

因此，依据上述动力模型，对于事后分析检测场景而言，基本步骤如下：

1. 将年度窗口用户中在职与离职用户分类，离职用户作为高CPB的潜在用户；
2. 将在职用户依据提取的ATF特征进行自动KMeans（2-10）聚类，筛选出CPB倾向较低的群簇，训练OCSVM；
3. 使用训练的OCSVM预测离职的测试用户，自动遍历参数选取最高Recall参数，输出结果；

上述工作还需要进行以下几个实验：

1. 计算ATF时使用的Leave Users还是LaidOff Users计算LCE特征；
2. 根据上述分析，OCEAN数据与JS或者CPB是重复特征，可以去掉不用；否则后续计算群簇公式时较难筛选；

首先使用去掉了OCEAN的新特征CPB\_ATF进行实验，后续全部步骤一致有：

此时的ATF文件为CERT5.2\_Static\_CPB\_ATF-0.1.csv

**PCA=1 Leave\_ATF**

**[0.05, 1.0, 0.8695652173913043, 0.8695652173913043, 0.0832016850974197, 0.10905452726363181, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',**

**max\_iter=-1, nu=0.05, random\_state=10, shrinking=True, tol=0.01,**

**verbose=False), 0.7931034482758621, 0.9, 1.0,]**

结果同使用OCEAN一样，也可以理解，因为OCEAN与后续的JS、CPB都相关，因而结果无影响。

DF函数结果保存在CERT5.2\_KMeans\_OCSVM\_CPB\_ATF\_Predictor\_Risk-0.1.csv

中

如果采用LaidOff来计算ATF，重新计算整个过程：结果略微比Leave ATF差，放弃！

[0.05, 1.0, 0.8405797101449275, 0.8405797101449275, 0.08109531332280147, 0.10605302651325663, OneClassSVM(cache\_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, nu=0.05, random\_state=10, shrinking=True, tol=0.01,

verbose=False), 0.7586206896551724, 0.8666666666666667, 1.0,

接下来，开始实验如何显示计算CPB来筛选群簇：

直接计算两个群簇的各个变量的均值/中位数，然后分别输入一个群簇筛选的标记计算：

我们先使用第一种计算公式看看：

上述计算中的结果来自于MinMax后的CPB\_ATF中的群簇中心值

计算

0 197

1 1564

0 ['2.5541490577132886', '3.4844853898564727']

1 ['2.6281383448135647', '3.622423493187438']

0 1564

1 197

0 ['2.5515736824051607', '3.5109122722209904']

1 ['2.4859452312892065', '3.3809321436702398']

上述结果竟然大群集的结果更高，不符合我们的预期

**0 1564**

**1 197**

**0 ['3.2753099575151476', '3.2186032810077254']**

**1 ['3.3508221785941816', '3.2531961210848963']**

即当放弃了缺勤率的系数之后，得到的大群集的CPB低于小群集

LED：0 ['0.7790327375125565', '1.0908185836192104']

LED：1 ['0.7418911236680935', '1.0392647777234978']

**0 1564**

**1 197**

**0 ['4.054342695027704', '4.309421864626936']**

**1 ['4.0927133022622755', '4.292460898808394']**

**如果改成CPB+缺勤比例和，结果也是大群簇CPB低**