with open(*'studentID\_test.txt'*, *'r'*) as f:

studentIds = [line.strip() for line in f.readlines()]

f2 = open(*'results/results.csv'*, *'r'*)

sids = [line.split(*','*)[0] for line in f2.readlines()]

f2.close()

f2 = open(*'results/results.csv'*, *'a'*)

t = list(set(studentIds).difference(set(sids)))

print(len(t))

for i in t:

f2.write(i + *",0\n"*)

f2.close()

*'''*

*Created on 2017年7月22日*

**@author:** *zhenglongtian*

*'''*

from Classifiers.SingleModels import SingleClassifier

from sklearn import tree

class **DecesionTree**(SingleClassifier.SingleClassifier):

def **\_\_init\_\_**(*self*):

SingleClassifier.SingleClassifier.\_\_init\_\_(*self*)

# weak classifier

*self*.clf = tree.DecisionTreeClassifier()

*'''*

*Created on 2017年7月22日*

**@author:** *zhenglongtian*

*'''*

from Classifiers.SingleModels import SingleClassifier

from sklearn.ensemble import ExtraTreesClassifier

class **ExtraTrees**(SingleClassifier.SingleClassifier):

def **\_\_init\_\_**(*self*):

SingleClassifier.SingleClassifier.\_\_init\_\_(*self*)

# weak classifier

*self*.clf = ExtraTreesClassifier(n\_estimators=10, max\_depth=None, min\_samples\_split=2, random\_state=0)

*'''*

*Created on 2017年7月22日*

**@author:** *zhenglongtian*

*'''*

from Classifiers.SingleModels import SingleClassifier

from sklearn.naive\_bayes import GaussianNB

class **GaussianNB**(SingleClassifier.SingleClassifier):

def **\_\_init\_\_**(*self*):

SingleClassifier.SingleClassifier.\_\_init\_\_(*self*)

#weak classifier

*self*.clf=GaussianNB()

*'''*

*Created on 2017年7月22日*

*高斯过程*

**@author:** *zhenglongtian*

*'''*

from Classifiers.SingleModels import SingleClassifier

from sklearn import gaussian\_process

class **GaussianProcesses**(SingleClassifier.SingleClassifier):

def **\_\_init\_\_**(*self*):

SingleClassifier.SingleClassifier.\_\_init\_\_(*self*)

# weak classifier

*self*.clf = gaussian\_process.GaussianProcess(theta0=1e-2, thetaL=1e-4, thetaU=1e-1)

*'''*

*Created on 2017年7月22日*

**@author:** *zhenglongtian*

*'''*

from Classifiers.SingleModels import SingleClassifier

from sklearn.neural\_network import MLPClassifier

class **MLP**(SingleClassifier.SingleClassifier):

def **\_\_init\_\_**(*self*):

SingleClassifier.SingleClassifier.\_\_init\_\_(*self*)

# weak classifier

*self*.clf = MLPClassifier(activation=*'relu'*, alpha=1e-05, batch\_size=*'auto'*,

beta\_1=0.9, beta\_2=0.999, early\_stopping=False,

epsilon=1e-08, hidden\_layer\_sizes=(5, 2), learning\_rate=*'constant'*,

learning\_rate\_init=0.001, max\_iter=200, momentum=0.9,

nesterovs\_momentum=True, power\_t=0.5, random\_state=1, shuffle=True,

solver=*'lbfgs'*, tol=0.0001, validation\_fraction=0.1, verbose=False,

warm\_start=False)

*'''*

*Created on 2017年7月22日*

*随机梯度下降*

**@author:** *zhenglongtian*

*'''*

from Classifiers.SingleModels import SingleClassifier

from sklearn.neighbors import NearestNeighbors

class **NearestNeighbors**(SingleClassifier.SingleClassifier):

def **\_\_init\_\_**(*self*):

SingleClassifier.SingleClassifier.\_\_init\_\_(*self*)

# weak classifier

algorithms = [*'brute'*, *'ball\_tree'*, *'kd\_tree'*]

*self*.clf = NearestNeighbors(n\_neighbors=2, algorithm=*'ball\_tree'*)

*'''*

*Created on 2017年7月22日*

**@author:** *zhenglongtian*

*'''*

from Classifiers.SingleModels import SingleClassifier

from sklearn.ensemble import RandomForestClassifier

class **RandomForest**(SingleClassifier.SingleClassifier):

def **\_\_init\_\_**(*self*):

SingleClassifier.SingleClassifier.\_\_init\_\_(*self*)

#weak classifier

*self*.clf=RandomForestClassifier(random\_state=1)

*'''*

*Created on 2017年7月22日*

*随机梯度下降*

**@author:** *zhenglongtian*

*'''*

from Classifiers.SingleModels import SingleClassifier

from sklearn.linear\_model import SGDClassifier

class **SGD**(SingleClassifier.SingleClassifier):

def **\_\_init\_\_**(*self*):

SingleClassifier.SingleClassifier.\_\_init\_\_(*self*)

#weak classifier

*self*.clf=SGDClassifier(alpha=0.0001, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate=*'optimal'*, loss=*'modified\_huber'*, n\_iter=5, n\_jobs=1,

penalty=*'l2'*, power\_t=0.5, random\_state=None, shuffle=True,

verbose=0, warm\_start=False)

*'''*

*Created on 2017年7月22日*

**@author:** *zhenglongtian*

*'''*

from Tools import DataCarer

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import AdaBoostClassifier

class **SingleClassifier**():

def **\_\_init\_\_**(*self*):

# weak classifier

*self*.clf = None

# get train data

*self*.X, *self*.Y = DataCarer.createTrainDataSet()

def **getBestOne**(*self*, name):

# if the classifier has already generated

try:

from sklearn.externals import joblib

clf = joblib.load(name + *'.pkl'*)

return clf

except:

pass

# if the classifier is not exists

# search for the best loop time

bestAccuracyRate, n\_estimators = 0, 1

for loopTimes in range(2, 200):

sclf = AdaBoostClassifier(base\_estimator=*self*.clf, learning\_rate=1, n\_estimators=loopTimes, algorithm=*'SAMME'*)

# cross validation to get the score

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(*self*.X, *self*.Y, test\_size=0.1, random\_state=0)

sclf.fit(X\_train, Y\_train)

accuracyRate = sclf.score(X\_test, Y\_test)

if accuracyRate > bestAccuracyRate:

bestAccuracyRate = accuracyRate

n\_estimators = loopTimes

# save the classifier as a dump

joblib.dump(sclf, name + *'.pkl'*)

return AdaBoostClassifier(base\_estimator=*self*.clf, learning\_rate=1, n\_estimators=n\_estimators, algorithm=*'SAMME'*)

*'''*

*Created on 2017年7月22日*

*支持向量机*

**@author:** *zhenglongtian*

*'''*

from Classifiers.SingleModels import SingleClassifier

from sklearn import svm

class **SVC**(SingleClassifier.SingleClassifier):

def **\_\_init\_\_**(*self*):

SingleClassifier.SingleClassifier.\_\_init\_\_(*self*)

# weak classifier

*self*.clf = svm.SVC(probability=True)

# 这边可以设权值来处理数据不平衡问题

*self*.weight = [1, 2, 3, ]

#

from sklearn import datasets

from sklearn import metrics

from sklearn.ensemble import ExtraTreesClassifier

import DataCarer

from sklearn import tree

from numpy import \*

from sklearn.linear\_model import SGDClassifier

from sklearn.ensemble import RandomForestClassifier

import DataCarer

from numpy import \*

from sklearn.pipeline import Pipeline

from sklearn.feature\_selection import SelectFromModel, SelectKBest, chi2

from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier, VotingClassifier, AdaBoostClassifier

from sklearn import tree

from sklearn import svm

from sklearn.svm import LinearSVC

from sklearn.neural\_network import MLPClassifier

from sklearn.linear\_model import SGDClassifier

from sklearn.neighbors.nearest\_centroid import NearestCentroid

from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

def **createTrainDataSet**():

dataSet = DataCarer.createTrainDataSet()

DataCarer.transform(dataSet)

return mat(dataSet)

def **createTestDataSet**():

students, dataSet = DataCarer.createTestDataSet()

DataCarer.transform(dataSet)

return students, dataSet

# get train data and test data

dataSet = mat(createTrainDataSet())

X\_train, Y\_train = dataSet[:, :-1], dataSet[:, -1]

students, dataSet = createTestDataSet()

X\_test = dataSet

# weak classifiers

clf0 = tree.DecisionTreeClassifier()

clf1 = ExtraTreesClassifier(n\_estimators=10, max\_depth=None, min\_samples\_split=2, random\_state=0)

clf2 = svm.SVC(probability=True)

clf4 = SGDClassifier(alpha=0.0001, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate=*'optimal'*, loss=*'modified\_huber'*, n\_iter=5, n\_jobs=1,

penalty=*'l2'*, power\_t=0.5, random\_state=None, shuffle=True,

verbose=0, warm\_start=False)

clf6 = RandomForestClassifier(random\_state=1)

clf7 = GaussianNB()

# stronger classifier

clf0 = AdaBoostClassifier(base\_estimator=clf0, learning\_rate=1, n\_estimators=110, algorithm=*'SAMME'*)

clf1 = AdaBoostClassifier(base\_estimator=clf1, learning\_rate=1, n\_estimators=50, algorithm=*'SAMME'*)

clf2 = AdaBoostClassifier(base\_estimator=clf2, learning\_rate=1, n\_estimators=1, algorithm=*'SAMME'*)

clf4 = AdaBoostClassifier(base\_estimator=clf4, learning\_rate=1, n\_estimators=1, algorithm=*'SAMME'*)

clf6 = AdaBoostClassifier(base\_estimator=clf6, learning\_rate=1, n\_estimators=150, algorithm=*'SAMME'*)

clf7 = AdaBoostClassifier(base\_estimator=clf7, learning\_rate=1, n\_estimators=150, algorithm=*'SAMME'*)

dataset = mat(createTrainDataSet())

clf0.fit(X\_train,Y\_train)

fetureSelection = SelectFromModel(clf0,prefit=True)

print(fetureSelection.get\_support(indices=True)) #display importance of each variables

# from sklearn import datasets

# from sklearn.feature\_selection import RFE

# from sklearn.linear\_model import LogisticRegression

#

# def createTrainDataSet():

# dataSet = DataCarer.createTrainDataSet()

# DataCarer.transform(dataSet)

# return mat(dataSet)

#

# dataset = mat(createTrainDataSet())

#

# X = dataset[:, :-1]

# Y = dataset[:, -1]

# model = LogisticRegression() # build logistic regression model

# rfe = RFE(model,20) # limit number of variables to three

# rfe = rfe.fit(X,Y)

# print(rfe.support\_)

# print(rfe.ranking\_)

from Tools import DataCarer

from sklearn.pipeline import Pipeline

from numpy import mat

from sklearn.feature\_selection import SelectFromModel, SelectKBest, chi2

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import VotingClassifier

from SingleModels import DecesionTree, ExtraTrees, GaussianNB, GaussianProcesses, MLP, NearestNeighbors, RandomForest, SGD, SVC

from sklearn.externals import joblib

# try:

# # 如果模型已经存在的话，则跑网上的验证集

# clf = joblib.load('final.pkl')

# students, X = DataCarer.createValidateDataSet()

# results = clf.predict(X)

# DataCarer.saveResult(students, results, 'results')

# except:

# 若果不存在的话则跑训练集

# single classifier

DecesionTree = DecesionTree.DecesionTree().getBestOne(*'DecesionTree'*)

ExtraTrees = ExtraTrees.ExtraTrees().getBestOne(*'ExtraTrees'*)

# GaussianNB = GaussianNB.GaussianNB().getBestOne('GaussianNB')

# GaussianProcesses = GaussianProcesses.GaussianProcesses().getBestOne('GaussianProcesses')

# MLP = MLP.MLP().getBestOne('MLP')

# NearestNeighbors = NearestNeighbors.NearestNeighbors().getBstOne('NearestNeighbors')

RandomForest = RandomForest.RandomForest().getBestOne(*'RandomForest'*)

SGD = SGD.SGD().getBestOne(*'SGD'*)

SVC = SVC.SVC().getBestOne(*'SVC'*)

# final classifier

# finalClassifier = VotingClassifier(estimators=[

# ('0', DecesionTree), ('1', ExtraTrees), ('2', GaussianNB), ('4', GaussianProcesses), ('6', MLP), ('7', NearestNeighbors), \

# ('8', RandomForest), ('9', SGD)],# ('5', SVC), ],

# voting='soft')

finalClassifier = VotingClassifier(estimators=[

(*'0'*, DecesionTree), (*'1'*, ExtraTrees), (*'2'*, SGD), (*'4'*, SVC), \

(*'8'*, RandomForest)],

voting=*'soft'*)

# feature selection

fetureSelection = SelectFromModel(DecesionTree)

clf = Pipeline([

(*'fetureSelection'*, fetureSelection),

(*'classification'*, finalClassifier)

])

# save the classifier as a dump

X, Y = DataCarer.createTrainDataSet()

accuracyRates = []

for i in range(1, 3):

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.1, random\_state=0)

clf.fit(X\_train, Y\_train)

accuracyRates.append(clf.score(X\_test, Y\_test))

if i == 1:

joblib.dump(clf, *'final.pkl'*)

print(mat(accuracyRates).mean())

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **Avg\_ChargeCaculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql=*"select avg\_charge from students order by avg\_charge"*

*self*.executer.execute(sql)

avg\_charge = *self*.executer.fetchall()

A = avg\_charge[int(len(avg\_charge) \* 0.125)][0]

B = avg\_charge[int(len(avg\_charge) \* 0.25)][0]

C = avg\_charge[int(len(avg\_charge) \* 0.375)][0]

D = avg\_charge[int(len(avg\_charge) \* 0.5)][0]

E = avg\_charge[int(len(avg\_charge) \* 0.625)][0]

F = avg\_charge[int(len(avg\_charge) \* 0.75)][0]

G = avg\_charge[int(len(avg\_charge) \* 0.875)][0]

H = avg\_charge[len(avg\_charge) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*Avg\_ChargeCaculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"SELECT AVG(deal\_cost) FROM card where student\_id = "*+ studentId+*" and deal\_type = 'charge'"*

*self*.executer.execute(sql)

avg = *self*.executer.fetchone()[0]

sql = *"update students set avg\_charge ='"* + str(avg) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*Avg\_ChargeCaculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select avg\_charge from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set avg\_charge='"* + s + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **AvgDaysCostsCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select avgdayscosts from students order by avgdayscosts"*

*self*.executer.execute(sql)

avgDaysCosts = *self*.executer.fetchall()

A = avgDaysCosts[int(len(avgDaysCosts) \* 0.125)][0]

B = avgDaysCosts[int(len(avgDaysCosts) \* 0.25)][0]

C = avgDaysCosts[int(len(avgDaysCosts) \* 0.375)][0]

D = avgDaysCosts[int(len(avgDaysCosts) \* 0.5)][0]

E = avgDaysCosts[int(len(avgDaysCosts) \* 0.625)][0]

F = avgDaysCosts[int(len(avgDaysCosts) \* 0.75)][0]

G = avgDaysCosts[int(len(avgDaysCosts) \* 0.875)][0]

H = avgDaysCosts[len(avgDaysCosts) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*AvgDaysCostsCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select avg(deal\_cost) from card where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

sql = *"update students set avgdayscosts='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*AvgDaysCostsCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select avgdayscosts from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set avgdayscosts='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **BalanceRankCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select balance\_rank from students order by balance\_rank"*

*self*.executer.execute(sql)

BalanceRanks = *self*.executer.fetchall()

A = BalanceRanks[int(len(BalanceRanks) \* 0.125)][0]

B = BalanceRanks[int(len(BalanceRanks) \* 0.25)][0]

C = BalanceRanks[int(len(BalanceRanks) \* 0.375)][0]

D = BalanceRanks[int(len(BalanceRanks) \* 0.5)][0]

E = BalanceRanks[int(len(BalanceRanks) \* 0.625)][0]

F = BalanceRanks[int(len(BalanceRanks) \* 0.75)][0]

G = BalanceRanks[int(len(BalanceRanks) \* 0.875)][0]

H = BalanceRanks[len(BalanceRanks) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*BalanceRankCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select min(balance),max(balance) from card where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()

minBalance, maxBalance = s[0], s[1]

averageBalance = str((minBalance + maxBalance) / 2)

sql = *"update students set balance\_rank='"* + averageBalance + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*BalanceRankCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select balance\_rank from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

averageBalance = *self*.classify(s)

sql = *"update students\_rank set balance\_rank='"* + averageBalance + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **Below10\_RankCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select below10\_rank from students order by below10\_rank"*

*self*.executer.execute(sql)

avgDaysCosts = *self*.executer.fetchall()

A = avgDaysCosts[int(len(avgDaysCosts) \* 0.125)][0]

B = avgDaysCosts[int(len(avgDaysCosts) \* 0.25)][0]

C = avgDaysCosts[int(len(avgDaysCosts) \* 0.375)][0]

D = avgDaysCosts[int(len(avgDaysCosts) \* 0.5)][0]

E = avgDaysCosts[int(len(avgDaysCosts) \* 0.625)][0]

F = avgDaysCosts[int(len(avgDaysCosts) \* 0.75)][0]

G = avgDaysCosts[int(len(avgDaysCosts) \* 0.875)][0]

H = avgDaysCosts[len(avgDaysCosts) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*Below10\_CostCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"SELECT count(\*) from card\_2 where studentid = "* + studentId + *" and date\_cost<10.0"*

*self*.executer.execute(sql)

below10 = float(*self*.executer.fetchone()[0])

sql = *"SELECT count(\*) from card\_2 where studentid = "* + studentId

*self*.executer.execute(sql)

allday = float(*self*.executer.fetchone()[0])

rank = below10 / allday

sql = *"update students set below10\_rank ='"* + str(rank) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*Below10\_CostCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select below10\_rank from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set below10\_rank ='"* + s + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **Below2\_5\_RankCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select Below2\_5\_Rank from students order by Below2\_5\_Rank "*

*self*.executer.execute(sql)

Below2\_5\_Rank = *self*.executer.fetchall()

A = Below2\_5\_Rank[int(len(Below2\_5\_Rank) \* 0.125)][0]

B = Below2\_5\_Rank[int(len(Below2\_5\_Rank) \* 0.25)][0]

C = Below2\_5\_Rank[int(len(Below2\_5\_Rank) \* 0.375)][0]

D = Below2\_5\_Rank[int(len(Below2\_5\_Rank) \* 0.5)][0]

E = Below2\_5\_Rank[int(len(Below2\_5\_Rank) \* 0.625)][0]

F = Below2\_5\_Rank[int(len(Below2\_5\_Rank) \* 0.75)][0]

G = Below2\_5\_Rank[int(len(Below2\_5\_Rank) \* 0.875)][0]

H = Below2\_5\_Rank[len(Below2\_5\_Rank) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*Below25\_RankCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"SELECT count(\*) from card where student\_id = "*+ studentId

*self*.executer.execute(sql)

all = *self*.executer.fetchone()[0]

sql = *"SELECT count(\*) from card where student\_id = "*+ studentId+*" and deal\_cost > 0 and deal\_cost<2.5"*

*self*.executer.execute(sql)

below25 = *self*.executer.fetchone()[0]

rank = below25/all

sql = *"update students set Below2\_5\_Rank ='"* + str(rank) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*Below25\_RankCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select Below2\_5\_Rank from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set Below2\_5\_Rank='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import Student

from Tools import MyDataBase

from tqdm import tqdm

from FeatureCalculate.AvgDaysCostsCalculater import AvgDaysCostsCalculater

from FeatureCalculate.BalanceRankCalculater import BalanceRankCalculater

from FeatureCalculate.CardDaysCalculater import CardDaysCalculater

from FeatureCalculate.CardRechargeCalculater import CardRechargeCalculater

from FeatureCalculate.ConsumeTimes11\_12Calculater import ConsumeTimes11\_12Calculater

from FeatureCalculate.CostAmountCalculater import CostAmountCalculater

from FeatureCalculate.CostAverageDayDinnerHallCalculater import CostAverageDayDinnerHallCalculater

from FeatureCalculate.CostAverageDayLaundryRoomCalculater import CostAverageDayLaundryRoomCalculater

from FeatureCalculate.CostAverageDaySupermarketCalculater import CostAverageDaySupermarketCalculater

from FeatureCalculate.CostTimesDayDinnerHallCalculater import CostTimesDayDinnerHallCalculater

from FeatureCalculate.CostTimesDayLaundryRoomCalculater import CostTimesDayLaundryRoomCalculater

from FeatureCalculate.CostTimesDaySupermarketCalculater import CostTimesDaySupermarketCalculater

from FeatureCalculate.CostRateDinnerHallCalculater import CostRateDinnerHallCalculater

from FeatureCalculate.CostRateLaundryRoomCalculater import CostRateLaundryRoomCalculater

from FeatureCalculate.CostRateSupermarketCalculater import CostRateSupermarketCalculater

from FeatureCalculate.CostVarianceCalculater import CostVarianceCalculater

from FeatureCalculate.CosumeTimes0\_25Calculater import CosumeTimes0\_25Calculater

from FeatureCalculate.CountCost0\_10Calculater import CountCost0\_10Calculater

from FeatureCalculate.LibraryBorrowCalculater import LibraryBorrowCalculater

from FeatureCalculate.LibraryTimesCalculater import LibraryTimesCalculater

from FeatureCalculate.LibraryTimeSpandCalculater import LibraryTimeSpandCalculater

from FeatureCalculate.MaxCost7\_8Calculater import MaxCost7\_8Calculater

from FeatureCalculate.ScoreRankCalculater import ScoreRankCalculater

from FeatureCalculate.SubsidyCalculater import SubsidyCalculater

from FeatureCalculate.Time6\_7CostsCalculater import Time6\_7CostsCalculater

from FeatureCalculate.Time7\_8CostsCalculater import Time7\_8CostsCalculater

from FeatureCalculate.TotalDinnerCostsCalculater import TotalDinnerCostsCalculater

from FeatureCalculate.Avg\_ChargeCaculater import Avg\_ChargeCaculater

from FeatureCalculate.Below2\_5\_RankCalculater import Below2\_5\_RankCalculater

from FeatureCalculate.Below10\_RankCalculater import Below10\_RankCalculater

from FeatureCalculate.Num\_Of\_1000Calculater import Num\_Of\_1000Calculater

from FeatureCalculate.Num\_Of\_2000Calculater import Num\_Of\_2000Calculater

from FeatureCalculate.Num\_Of\_1500Calculater import Num\_Of\_1500Calculater

from FeatureCalculate.PropotionCalculater1000 import PropotionCalculater1000

from FeatureCalculate.PropotionCalculater2000 import PropotionCalculater2000

from FeatureCalculate.PropotionCalculater1500 import PropotionCalculater1500

from FeatureCalculate.scorerank\_divided\_by\_stunum import scorerank\_divided\_by\_stunum

from FeatureCalculate.Stu\_Num\_Calculater import Stu\_Num\_Calculater

from FeatureCalculate.Time7\_8Consume\_Avg import Time7\_8Consume\_Avg

Student = Student.Student

AvgDaysCostsCalculater = AvgDaysCostsCalculater()

Below10\_RankCalculater=Below10\_RankCalculater()

BalanceRankCalculater = BalanceRankCalculater()

CardDaysCalculater = CardDaysCalculater()

CardRechargeCalculater = CardRechargeCalculater()

ConsumeTimes11\_12Calculater = ConsumeTimes11\_12Calculater()

CostAmountCalculater = CostAmountCalculater()

CostAverageDayDinnerHallCalculater = CostAverageDayDinnerHallCalculater()

CostAverageDayLaundryRoomCalculater = CostAverageDayLaundryRoomCalculater()

CostAverageDaySupermarketCalculater = CostAverageDaySupermarketCalculater()

CostTimesDayDinnerHallCalculater = CostTimesDayDinnerHallCalculater()

CostTimesDayLaundryRoomCalculater = CostTimesDayLaundryRoomCalculater()

CostTimesDaySupermarketCalculater = CostTimesDaySupermarketCalculater()

CostRateDinnerHallCalculater = CostRateDinnerHallCalculater()

CostRateLaundryRoomCalculater = CostRateLaundryRoomCalculater()

CostRateSupermarketCalculater = CostRateSupermarketCalculater()

CostVarianceCalculater = CostVarianceCalculater()

CosumeTimes0\_25Calculater = CosumeTimes0\_25Calculater()

CountCost0\_10Calculater = CountCost0\_10Calculater()

LibraryBorrowCalculater = LibraryBorrowCalculater()

LibraryTimesCalculater = LibraryTimesCalculater()

LibraryTimeSpandCalculater = LibraryTimeSpandCalculater()

MaxCost7\_8Calculater = MaxCost7\_8Calculater()

ScoreRankCalculater = ScoreRankCalculater()

SubsidyCalculater = SubsidyCalculater()

Time6\_7CostsCalculater = Time6\_7CostsCalculater()

Time7\_8CostsCalculater = Time7\_8CostsCalculater()

TotalDinnerCostsCalculater = TotalDinnerCostsCalculater()

Avg\_ChargeCaculater = Avg\_ChargeCaculater()

Below2\_5\_RankCalculater = Below2\_5\_RankCalculater()

Num\_Of\_1000Calculater = Num\_Of\_1000Calculater()

Num\_Of\_2000Calculater = Num\_Of\_2000Calculater()

Num\_Of\_1500Calculater = Num\_Of\_1500Calculater()

PropotionCalculater1000 = PropotionCalculater1000()

PropotionCalculater2000 = PropotionCalculater2000()

PropotionCalculater1500 = PropotionCalculater1500()

scorerank\_divided\_by\_stunum = scorerank\_divided\_by\_stunum()

Stu\_Num\_Calculater = Stu\_Num\_Calculater()

Time7\_8Consume\_Avg = Time7\_8Consume\_Avg()

calculater = [

# Stu\_Num\_Calculater,

# Num\_Of\_1000Calculater,

# Num\_Of\_2000Calculater,

# Num\_Of\_1500Calculater,

ScoreRankCalculater,

Below10\_RankCalculater,

Time6\_7CostsCalculater,

Time7\_8CostsCalculater,

TotalDinnerCostsCalculater,

AvgDaysCostsCalculater,

BalanceRankCalculater,

CardDaysCalculater,

CardRechargeCalculater,

ConsumeTimes11\_12Calculater,

CostAmountCalculater,

CostAverageDayDinnerHallCalculater,

CostAverageDayLaundryRoomCalculater,

CostAverageDaySupermarketCalculater,

CostRateDinnerHallCalculater,

CostRateLaundryRoomCalculater,

CostRateSupermarketCalculater,

CostTimesDayDinnerHallCalculater,

CostTimesDayLaundryRoomCalculater,

CostTimesDaySupermarketCalculater,

CostVarianceCalculater,

CosumeTimes0\_25Calculater,

CountCost0\_10Calculater,

LibraryBorrowCalculater,

LibraryTimesCalculater,

LibraryTimeSpandCalculater,

MaxCost7\_8Calculater,

Avg\_ChargeCaculater,

Below2\_5\_RankCalculater,

PropotionCalculater1000,

PropotionCalculater2000,

PropotionCalculater1500,

scorerank\_divided\_by\_stunum,

SubsidyCalculater,

Time7\_8Consume\_Avg

]

# calculater = [SubsidyCalculater]

def **calculate**():

db = MyDataBase.MyDataBase(*"train"*)

conn = db.getConn()

executer = db.getExcuter()

sql = *"select student\_id from score"*

executer.execute(sql)

studentIds = executer.fetchall()

db.close()

# for i in tqdm(studentIds):

# i = i[0]

# student = Student(studentId=i)

# student.calculate(calculater)

for i in calculater:

i.setLevel()

for i in tqdm(studentIds):

i = i[0]

student = Student(studentId=i)

student.rankit(calculater)

for i in calculater:

i.afterCalculate()

if \_\_name\_\_ == *'\_\_main\_\_'*:

calculate()

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **CardDaysCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select card\_days from students order by card\_days"*

*self*.executer.execute(sql)

card\_days = *self*.executer.fetchall()

A = card\_days[int(len(card\_days) \* 0.125)][0]

B = card\_days[int(len(card\_days) \* 0.25)][0]

C = card\_days[int(len(card\_days) \* 0.375)][0]

D = card\_days[int(len(card\_days) \* 0.5)][0]

E = card\_days[int(len(card\_days) \* 0.625)][0]

F = card\_days[int(len(card\_days) \* 0.75)][0]

G = card\_days[int(len(card\_days) \* 0.875)][0]

H = card\_days[len(card\_days) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*CardDaysCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select count(\*) from card\_2 where studentid="* + str(studentId)

*self*.executer.execute(sql)

card\_days = *self*.executer.fetchone()[0]

sql = *"update students set card\_days ='"* + str(card\_days) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*CardDaysCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select card\_days from students where student\_id="* + studentId

*self*.executer.execute(sql)

card\_days = *self*.executer.fetchone()[0]

card\_days = *self*.classify(card\_days)

sql = *"update students set card\_days='"* + card\_days + *"' where student\_id="* + str(studentId)

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **CardDaysCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select card\_days from students order by card\_days"*

*self*.executer.execute(sql)

card\_days = *self*.executer.fetchall()

A = card\_days[int(len(card\_days) \* 0.125)][0]

B = card\_days[int(len(card\_days) \* 0.25)][0]

C = card\_days[int(len(card\_days) \* 0.375)][0]

D = card\_days[int(len(card\_days) \* 0.5)][0]

E = card\_days[int(len(card\_days) \* 0.625)][0]

F = card\_days[int(len(card\_days) \* 0.75)][0]

G = card\_days[int(len(card\_days) \* 0.875)][0]

H = card\_days[len(card\_days) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*CardDaysCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select count(\*) from card\_2 where studentid="* + str(studentId)

*self*.executer.execute(sql)

card\_days = *self*.executer.fetchone()[0]

sql = *"update students set card\_days ='"* + str(card\_days) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*CardDaysCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select card\_days from students where student\_id="* + studentId

*self*.executer.execute(sql)

card\_days = *self*.executer.fetchone()[0]

card\_days = *self*.classify(card\_days)

sql = *"update students set card\_days='"* + card\_days + *"' where student\_id="* + str(studentId)

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **ConsumeTimes11\_12Calculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select consumetimes11\_12 from students order by consumetimes11\_12"*

*self*.executer.execute(sql)

ConsumeTimes11\_12Ranks = *self*.executer.fetchall()

A = ConsumeTimes11\_12Ranks[int(len(ConsumeTimes11\_12Ranks) \* 0.125)][0]

B = ConsumeTimes11\_12Ranks[int(len(ConsumeTimes11\_12Ranks) \* 0.25)][0]

C = ConsumeTimes11\_12Ranks[int(len(ConsumeTimes11\_12Ranks) \* 0.375)][0]

D = ConsumeTimes11\_12Ranks[int(len(ConsumeTimes11\_12Ranks) \* 0.5)][0]

E = ConsumeTimes11\_12Ranks[int(len(ConsumeTimes11\_12Ranks) \* 0.625)][0]

F = ConsumeTimes11\_12Ranks[int(len(ConsumeTimes11\_12Ranks) \* 0.75)][0]

G = ConsumeTimes11\_12Ranks[int(len(ConsumeTimes11\_12Ranks) \* 0.875)][0]

H = ConsumeTimes11\_12Ranks[len(ConsumeTimes11\_12Ranks) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*ConsumeTimes11\_12Calculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select count(\*) from card where student\_id="* + studentId + *" and hour(deal\_date)=11"*

*self*.executer.execute(sql)

s = str(*self*.executer.fetchone()[0])

sql = *"update students set consumetimes11\_12='"* + s + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*ConsumeTimes11\_12Calculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select consumetimes11\_12 from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set consumetimes11\_12='"* + s + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from Tools import MyLog

from Tools import MyDataBase

class **XXCalculater**:

def **\_\_init\_\_**(*self*):

*self*.db = MyDataBase.MyDataBase(*"train"*)

*self*.conn = *self*.db.getConn()

*self*.executer = *self*.db.getExcuter()

*self*.level = None

def **setStudent**(*self*, student):

*self*.student = student

def **setLevel**(*self*):

pass

*@MyLog.myException*

def **calculate**(*self*):

pass

def **classify**(*self*, param):

if param is None:

return 0

for i in range(len(*self*.level)):

if float(param) <= float(*self*.level[i]):

return str(i + 1)

return str(len(*self*.level) + 1)

def **afterCalculate**(*self*):

*self*.db.close()

if \_\_name\_\_ == *'\_\_main\_\_'*:

print(*'a father'*)

# from \_overlapped import NULL

class **Student**:

def **\_\_init\_\_**(*self*, studentId, attributes=None, subsidy=*"1"*):

*self*.studentId = studentId

try:

*self*.attributes = list(attributes)

except:

pass

*self*.subsidy = subsidy

def **calculate**(*self*, calculater):

for i in calculater:

i.setStudent(*self*)

i.calculate()

def **rankit**(*self*, calculater):

for i in calculater:

i.setStudent(*self*)

i.rankit()

def **getAll**(*self*):

(*self*.attributes).append(*self*.subsidy)

return *self*.attributes

def **setStudentId**(*self*, studentId):

*self*.studentId = studentId

def **getStudentId**(*self*):

return *self*.studentId

*'''*

*Created on 2017年7月22日*

**@author:** *zhenglongtian*

*'''*

from FeatureCalculate import Student

from Tools import MyDataBase

import random

from sklearn.neighbors import NearestNeighbors

from numpy import mat

*'''*

*used to deal with the data*

*'''*

Student = Student.Student

def **createTrainDataSet**():

*'''*

*get train data*

*'''*

db = MyDataBase.MyDataBase(*"train"*)

conn, executer = db.getConn(), db.getExcuter()

# get all the students

executer.execute(*"select \* from students\_rank"*)

dataSet = []

for i in executer.fetchall():

student = Student(studentId=i[0], attributes=list(i[1:-1]), subsidy=i[-1])

dataSet.append(student.getAll())

conn.close();executer.close()

dataSet = mat(dataSet)

return dataSet[:, :-1], dataSet[:, -1]

def **createValidateDataSet**():

*'''*

*get validate data*

*'''*

db = MyDataBase.MyDataBase(*"validate"*)

conn, executer = db.getConn(), db.getExcuter()

# get all the students

executer.execute(*"select \* from students\_rank"*)

students,dataSet = [],[]

for i in executer.fetchall():

student = Student(studentId=i[0], attributes=list(i[1:-1]), subsidy=i[-1])

dataSet.append(student.getAll())

students.append(student)

conn.close();executer.close()

dataSet = mat(dataSet)

return students,dataSet[:, :-1]

def **DataImbalanceProcessing**(dataSet):

# 处理数据不平衡问题

# 统计每种类别的个数

classCount = {*'A'*:0, *'B'*:0, *'C'*:0, *'D'*:0}

for i in dataSet:

classCount[(isinstance(i[-1], int) and str(chr(ord(*'A'*) + i[-1] - 1))) or str(i[-1])] += 1

classCount\_result = sorted(classCount.items(), key=lambda asd:asd[1], reverse=False)

flag = True

while flag:

flag = False

for i in classCount\_result[:-1]:

if i[1] > 0 and 20 \* i[1] < classCount\_result[-1][1]:

flag = True

type = i[0]

temp = []

for student in dataSet :

if ((isinstance(student[-1], int) and str(chr(ord(*'A'*) + student[-1] - 1))) or str(student[-1])) == type:

t = student[:]

temp.append(t)

classCount[type] += 1

dataSet.extend(temp)

classCount\_result = sorted(classCount.items(), key=lambda asd:asd[1], reverse=False)

def **saveResult**(students, results, filename):

*'''*

*save the result to a file*

*'''*

with open(*'../AccuracyValidation/results/'* + filename + *'.csv'*, *'w'*)as f:

# 提交到网上要求的第一行

f.write(*"studentid,subsidy\n"*)

temp = *""*

for student, result in zip(students, results):

if result == 1:

temp = 0

elif result == 2:

temp = 1000

elif result == 3:

temp = 1500

elif result == 4:

temp = 2000

else:

print(*"it is weird"*)

f.write(str(student.getStudentId()) + *","* + str(temp) + *"\n"*)

# class Smote:

# def \_\_init\_\_(self, samples, N=10, k=5):

# self.n\_samples, self.n\_attrs = samples.shape

# self.N = N

# self.k = k

# self.samples = samples

# self.newindex = 0

# # self.synthetic=np.zeros((self.n\_samples\*N,self.n\_attrs))

#

# def over\_sampling(self):

# N = int(self.N / 100)

# self.synthetic = np.zeros((self.n\_samples \* N, self.n\_attrs))

# neighbors = NearestNeighbors(n\_neighbors=self.k).fit(self.samples)

# print 'neighbors', neighbors

# for i in range(len(self.samples)):

# nnarray = neighbors.kneighbors(self.samples[i].reshape(1, -1), return\_distance=False)[0]

# # print nnarray

# self.\_populate(N, i, nnarray)

# return self.synthetic

#

#

# # for each minority class samples,choose N of the k nearest neighbors and generate N synthetic samples.

# def \_populate(self, N, i, nnarray):

# for j in range(N):

# nn = random.randint(0, self.k - 1)

# dif = self.samples[nnarray[nn]] - self.samples[i]

# gap = random.random()

# self.synthetic[self.newindex] = self.samples[i] + gap \* dif

# self.newindex += 1

# a = np.array([[1, 2, 3], [4, 5, 6], [2, 3, 1], [2, 1, 2], [2, 3, 4], [2, 3, 4]])

# s = Smote(a, N=100)

# print s.over\_sampling()

import pymysql

class **MyDataBase**:

def **\_\_init\_\_**(*self*, database=*"train"*):

# print("connect to data base......")

*self*.db = pymysql.connect(*"localhost"*, *"root"*, *"root"*, *"intelligentCampus"* + database, charset=*'utf8'*)

*self*.cursor = *self*.db.cursor()

*self*.db.autocommit(True)

# print("connect success!")

def **getConn**(*self*):

return *self*.db

def **getExcuter**(*self*):

return *self*.cursor

def **close**(*self*):

*self*.cursor.close()

*self*.db.close()

if \_\_name\_\_ == *'\_\_main\_\_'*:

print(*"a module used to connect db"*)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **MaxCost7\_8Calculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select maxcost7\_8 from students order by maxcost7\_8"*

*self*.executer.execute(sql)

MaxCost7\_8Ranks = *self*.executer.fetchall()

A = MaxCost7\_8Ranks[int(len(MaxCost7\_8Ranks) \* 0.125)][0]

B = MaxCost7\_8Ranks[int(len(MaxCost7\_8Ranks) \* 0.25)][0]

C = MaxCost7\_8Ranks[int(len(MaxCost7\_8Ranks) \* 0.375)][0]

D = MaxCost7\_8Ranks[int(len(MaxCost7\_8Ranks) \* 0.5)][0]

E = MaxCost7\_8Ranks[int(len(MaxCost7\_8Ranks) \* 0.625)][0]

F = MaxCost7\_8Ranks[int(len(MaxCost7\_8Ranks) \* 0.75)][0]

G = MaxCost7\_8Ranks[int(len(MaxCost7\_8Ranks) \* 0.875)][0]

H = MaxCost7\_8Ranks[len(MaxCost7\_8Ranks) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*MaxCost7\_8Calculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select sum(deal\_cost) as a from card where student\_id="* + studentId + *" and hour(deal\_date)=7 group by date(deal\_date) order by a limit 1"*

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

sql = *"update students set maxcost7\_8='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*MaxCost7\_8Calculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select maxcost7\_8 from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set maxcost7\_8='"* + s + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **Num\_Of\_1500Calculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

pass

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*正在计算各个学院获得1500助学金数量*

*'''*

i = 1

while i <= 19:

sql =*"SELECT count(\*) from score LEFT JOIN subsidy ON score.student\_id = subsidy.student\_id WHERE college\_id = "*+ str(i)+ *" and stipend = 1000"*

*self*.executer.execute(sql)

e = *self*.executer.fetchone()[0]

sql=*"update college\_info set 1500\_num = "*+str(e)+*" where college\_id = "*+str(i)

*self*.executer.execute(sql)

i = i + 1

*@MyLog.myException*

def **rankit**(*self*):

pass

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **PropotionCalculater2000**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select propotion\_of\_2000 from students order by propotion\_of\_2000 "*

*self*.executer.execute(sql)

propotion\_of\_2000 = *self*.executer.fetchall()

A = propotion\_of\_2000[int(len(propotion\_of\_2000) \* 0.125)][0]

B = propotion\_of\_2000[int(len(propotion\_of\_2000) \* 0.25)][0]

C = propotion\_of\_2000[int(len(propotion\_of\_2000) \* 0.375)][0]

D = propotion\_of\_2000[int(len(propotion\_of\_2000) \* 0.5)][0]

E = propotion\_of\_2000[int(len(propotion\_of\_2000) \* 0.625)][0]

F = propotion\_of\_2000[int(len(propotion\_of\_2000) \* 0.75)][0]

G = propotion\_of\_2000[int(len(propotion\_of\_2000) \* 0.875)][0]

H = propotion\_of\_2000[len(propotion\_of\_2000) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*PropotionCalculater2000.rankit*

*'''*

sql = *"SELECT SUM(2000\_num) FROM college\_info"*

*self*.executer.execute(sql)

sum2000 = *self*.executer.fetchone()[0]

studentId = str(*self*.student.getStudentId())

sql = *"select college\_id from score where student\_id ="* + studentId

*self*.executer.execute(sql)

collegeId = *self*.executer.fetchone()[0]

sql = *"select 2000\_num from college\_info where college\_id = "*+str(collegeId)

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]/sum2000

sql = *"update students set propotion\_of\_2000 ='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*PropotionCalculater2000.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select propotion\_of\_2000 from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set propotion\_of\_2000 = '"* + str(s) + *"' where student\_id= "* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **scorerank\_divided\_by\_stunum**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select scorerank\_divided\_by\_stunum from students order by scorerank\_divided\_by\_stunum "*

*self*.executer.execute(sql)

scorerank\_divided\_by\_stunum = *self*.executer.fetchall()

A = scorerank\_divided\_by\_stunum[int(len(scorerank\_divided\_by\_stunum) \* 0.125)][0]

B = scorerank\_divided\_by\_stunum[int(len(scorerank\_divided\_by\_stunum) \* 0.25)][0]

C = scorerank\_divided\_by\_stunum[int(len(scorerank\_divided\_by\_stunum) \* 0.375)][0]

D = scorerank\_divided\_by\_stunum[int(len(scorerank\_divided\_by\_stunum) \* 0.5)][0]

E = scorerank\_divided\_by\_stunum[int(len(scorerank\_divided\_by\_stunum) \* 0.625)][0]

F = scorerank\_divided\_by\_stunum[int(len(scorerank\_divided\_by\_stunum) \* 0.75)][0]

G = scorerank\_divided\_by\_stunum[int(len(scorerank\_divided\_by\_stunum) \* 0.875)][0]

H = scorerank\_divided\_by\_stunum[len(scorerank\_divided\_by\_stunum) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*scorerank\_divided\_by\_stunum.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"SELECT rank FROM score where student\_id = "*+ studentId

*self*.executer.execute(sql)

rank = *self*.executer.fetchone()[0]

sql = *"select college\_id from score where student\_id ="* + studentId

*self*.executer.execute(sql)

collegeId = *self*.executer.fetchone()[0]

sql = *"select stu\_num from college\_info where college\_id = "*+str(collegeId)

*self*.executer.execute(sql)

stu\_num = *self*.executer.fetchone()[0]

s = rank/stu\_num

sql = *"update students set scorerank\_divided\_by\_stunum ='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*scorerank\_divided\_by\_stunum.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select scorerank\_divided\_by\_stunum from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set scorerank\_divided\_by\_stunum = '"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **ScoreRankCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select score from students order by score"*

*self*.executer.execute(sql)

scoreRank = *self*.executer.fetchall()

A = scoreRank[int(len(scoreRank) \* 0.125)][0]

B = scoreRank[int(len(scoreRank) \* 0.25)][0]

C = scoreRank[int(len(scoreRank) \* 0.375)][0]

D = scoreRank[int(len(scoreRank) \* 0.5)][0]

E = scoreRank[int(len(scoreRank) \* 0.625)][0]

F = scoreRank[int(len(scoreRank) \* 0.75)][0]

G = scoreRank[int(len(scoreRank) \* 0.875)][0]

H = scoreRank[len(scoreRank) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*ScoreRankCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select rank from score where student\_id="* + studentId

*self*.executer.execute(sql)

score = *self*.executer.fetchone()[0]

sql = *"insert into students(student\_id,score) values("* + studentId + *",'"* + str(score) + *"')"*

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*ScoreRankCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select score from students where student\_id="* + studentId

*self*.executer.execute(sql)

score = *self*.executer.fetchone()[0]

score = *self*.classify(score)

sql = *"insert into students\_rank(student\_id,score) values("* + studentId + *",'"* + str(score) + *"')"*

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **Stu\_Num\_Calculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

pass

*@MyLog.myException*

def **calculate**(*self*):

i = 1

while i <= 19:

sql =*"SELECT count(\*) from score WHERE college\_id = "*+ str(i)

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

sql=*"update college\_info set stu\_num = "*+str(s)+*" where college\_id ="*+str(i)

print(sql)

*self*.executer.execute(sql)

i = i + 1

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **SubsidyCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

*self*.level = [0, 1000, 1500, 2000]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*SubsidyCalculater.calculate*

*'''*

studentId = *self*.student.getStudentId()

sql = *"select stipend from subsidy where student\_id="* + str(studentId)

*self*.executer.execute(sql)

subsidy = *self*.executer.fetchone()[0]

sql = *"update students set subsidy= '"* + str(subsidy) + *"' where student\_id = "* + str(studentId)

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*ScoreRankCalculater.rankit*

*'''*

studentId = *self*.student.getStudentId()

sql = *"select subsidy from students where student\_id="* + str(studentId)

*self*.executer.execute(sql)

subsidy = *self*.executer.fetchone()[0]

subsidy = *self*.classify(subsidy)

sql = *"update students\_rank set subsidy= '"* + subsidy + *"' where student\_id = "* + str(studentId)

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **Time6\_7CostsCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select time6\_7costs from students order by time6\_7costs "*

*self*.executer.execute(sql)

Time6\_7CostsRanks = *self*.executer.fetchall()

A = Time6\_7CostsRanks[int(len(Time6\_7CostsRanks) \* 0.125)][0]

B = Time6\_7CostsRanks[int(len(Time6\_7CostsRanks) \* 0.25)][0]

C = Time6\_7CostsRanks[int(len(Time6\_7CostsRanks) \* 0.375)][0]

D = Time6\_7CostsRanks[int(len(Time6\_7CostsRanks) \* 0.5)][0]

E = Time6\_7CostsRanks[int(len(Time6\_7CostsRanks) \* 0.625)][0]

F = Time6\_7CostsRanks[int(len(Time6\_7CostsRanks) \* 0.75)][0]

G = Time6\_7CostsRanks[int(len(Time6\_7CostsRanks) \* 0.875)][0]

H = Time6\_7CostsRanks[len(Time6\_7CostsRanks) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*Time6\_7CostsCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select sum(deal\_cost) from card where student\_id="* + studentId + *" and hour(deal\_date)=6"*

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

sql = *"update students set time6\_7costs='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*Time6\_7CostsCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select time6\_7costs from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set time6\_7costs='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **Time7\_8Consume\_Avg**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select time7\_8costs from students order by time7\_8costs"*

*self*.executer.execute(sql)

Time7\_8Consume\_Avg = *self*.executer.fetchall()

A = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.125)][0]

B = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.25)][0]

C = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.375)][0]

D = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.5)][0]

E = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.625)][0]

F = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.75)][0]

G = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.875)][0]

H = Time7\_8Consume\_Avg[len(Time7\_8Consume\_Avg) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*Time7\_8Consume\_Avg.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select sum(deal\_cost) as a from card where student\_id="* + studentId + *" and hour(deal\_date)=7 group by date(deal\_date) order by a limit 1"*

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

sql = *"update students set time7\_8consume\_avg ='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*Time7\_8Consume\_Avg.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select time7\_8consume\_avg from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set time7\_8consume\_avg = '"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **Time7\_8Consume\_Avg**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select time7\_8costs from students order by time7\_8costs"*

*self*.executer.execute(sql)

Time7\_8Consume\_Avg = *self*.executer.fetchall()

A = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.125)][0]

B = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.25)][0]

C = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.375)][0]

D = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.5)][0]

E = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.625)][0]

F = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.75)][0]

G = Time7\_8Consume\_Avg[int(len(Time7\_8Consume\_Avg) \* 0.875)][0]

H = Time7\_8Consume\_Avg[len(Time7\_8Consume\_Avg) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*Time7\_8Consume\_Avg.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select sum(deal\_cost) as a from card where student\_id="* + studentId + *" and hour(deal\_date)=7 group by date(deal\_date) order by a limit 1"*

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

sql = *"update students set time7\_8consume\_avg ='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*Time7\_8Consume\_Avg.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select time7\_8consume\_avg from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set time7\_8consume\_avg = '"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **Time7\_8CostsCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select time7\_8costs from students order by time6\_7costs "*

*self*.executer.execute(sql)

Time7\_8CostsRanks = *self*.executer.fetchall()

A = Time7\_8CostsRanks[int(len(Time7\_8CostsRanks) \* 0.125)][0]

B = Time7\_8CostsRanks[int(len(Time7\_8CostsRanks) \* 0.25)][0]

C = Time7\_8CostsRanks[int(len(Time7\_8CostsRanks) \* 0.375)][0]

D = Time7\_8CostsRanks[int(len(Time7\_8CostsRanks) \* 0.5)][0]

E = Time7\_8CostsRanks[int(len(Time7\_8CostsRanks) \* 0.625)][0]

F = Time7\_8CostsRanks[int(len(Time7\_8CostsRanks) \* 0.75)][0]

G = Time7\_8CostsRanks[int(len(Time7\_8CostsRanks) \* 0.875)][0]

H = Time7\_8CostsRanks[len(Time7\_8CostsRanks) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*Time7\_8CostsCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select sum(deal\_cost) from card where student\_id="* + studentId +*" and hour(deal\_date)=7"*

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

sql = *"update students set time7\_8costs='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*Time7\_8CostsCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select time7\_8consume\_avg from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set time7\_8consume\_avg = '"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

from FeatureCalculate import XXCalculater

from Tools import MyLog

class **TotalDinnerCostsCalculater**(XXCalculater.XXCalculater):

def **setLevel**(*self*):

sql = *"select totaldinnercosts from students order by totaldinnercosts"*

*self*.executer.execute(sql)

TotalDinnerCostsRanks = *self*.executer.fetchall()

A = TotalDinnerCostsRanks[int(len(TotalDinnerCostsRanks) \* 0.125)][0]

B = TotalDinnerCostsRanks[int(len(TotalDinnerCostsRanks) \* 0.25)][0]

C = TotalDinnerCostsRanks[int(len(TotalDinnerCostsRanks) \* 0.375)][0]

D = TotalDinnerCostsRanks[int(len(TotalDinnerCostsRanks) \* 0.5)][0]

E = TotalDinnerCostsRanks[int(len(TotalDinnerCostsRanks) \* 0.625)][0]

F = TotalDinnerCostsRanks[int(len(TotalDinnerCostsRanks) \* 0.75)][0]

G = TotalDinnerCostsRanks[int(len(TotalDinnerCostsRanks) \* 0.875)][0]

H = TotalDinnerCostsRanks[len(TotalDinnerCostsRanks) - 1][0]

*self*.level = [A, B, C, D, E, F, G]

*@MyLog.myException*

def **calculate**(*self*):

*'''*

*TotalDinnerCostsCalculater.calculate*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select sum(deal\_cost) from card where student\_id="* + studentId + *" and deal\_way = 'dinnerhall'"*

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

sql = *"update students set totaldinnercosts='"* + str(s) + *"' where student\_id="* + studentId

*self*.executer.execute(sql)

*@MyLog.myException*

def **rankit**(*self*):

*'''*

*TotalDinnerCostsCalculater.rankit*

*'''*

studentId = str(*self*.student.getStudentId())

sql = *"select totaldinnercosts from students where student\_id="* + studentId

*self*.executer.execute(sql)

s = *self*.executer.fetchone()[0]

s = *self*.classify(s)

sql = *"update students\_rank set totaldinnercosts='"* + s + *"' where student\_id="* + studentId

*self*.executer.execute(sql)