# purchasing feature analysis

import pandas as pd

import numpy as np

from numpy import array

import psycopg2

import math

import traceback

import matplotlib.pyplot as plt

import lightgbm as lgb

from collections import defaultdict

from sklearn.neural\_network import MLPClassifier

from sklearn import preprocessing

from sklearn.grid\_search import GridSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier

from sklearn import svm

from sklearn.metrics import f1\_score, precision\_score, recall\_score, confusion\_matrix, roc\_auc\_score, classification\_report, roc\_curve, auc

#import seaborn as sns

# output path

input\_path = "C:/Users/yawen/Google Drive/spatial temporal data analytics/urban computing/PurchasingBehavior/feature\_analysis\_output/"

output\_path = "C:/Users/yawen/Google Drive/spatial temporal data analytics/urban computing/PurchasingBehavior/feature\_analysis\_output/"

# connect to database and get cursor

try:

conn = psycopg2.connect(database = 'tmall', user = 'postgres', host = '128.138.157.86', port = '5432', password = '1234')

except psycopg2.Error as e:

print("I am unable to connect to the database")

print(e)

print(e.pgcode)

print(e.pgerror)

print(traceback.format\_exc())

cur = conn.cursor()

def features():

"""

get all features

"""

cur.execute("select \* from train\_ui\_up\_mp\_train\_ui\_up\_mp\_um\_double\_pca\_sim\_slope\_fillna")

train\_ui\_up\_mp\_um\_double\_pca\_sim\_slope\_fillna\_features = pd.DataFrame(cur.fetchall(), columns = [i[0] for i in cur.description])

return train\_ui\_up\_mp\_um\_double\_pca\_sim\_slope\_fillna\_features

# train\_data\_user\_merchant\_profile\_features\_stat = train\_data\_user\_merchant\_profile\_features.groupby('label').describe()

# train\_data\_user\_merchant\_profile\_features\_stat.to\_csv(output\_path + "train\_data\_user\_merchant\_profile\_features\_stat.csv")

# check missing value perc

# print(user\_info.isnull().sum())

# print(user\_label[user\_label['label'] == 1])

# plot the histgram for columns

# user\_label.hist(column = 'label', bins = 2, grid = False)

# plt.xticks(np.arange(-2, 2, 1))

# plt.show()

def classification(train\_ui\_up\_mp\_um\_double\_pca\_sim\_slope\_fillna\_features):

# ignore NaN, if any value is NaN

# train\_data\_user\_merchant\_profile\_features = train\_data\_user\_merchant\_profile\_features.dropna(how = 'any')

x = train\_ui\_up\_mp\_um\_double\_pca\_sim\_slope\_fillna\_features.filter(regex = 'mp')

x = x.apply(lambda x: (x - x.min()) / (x.max() - x.min()))

y = train\_ui\_up\_mp\_um\_double\_pca\_sim\_slope\_fillna\_features['label']

train\_size = [50000, 100000, 150000, 200000]

test\_size = [10000, 20000, 30000, 50000]

for i in range(3, 4):

train\_num = train\_size[i]

test\_num = test\_size[i]

x\_train\_o = x.iloc[:train\_num]

y\_train\_o = y.iloc[:train\_num]

# undersampling

repeat\_indices = y\_train\_o[y\_train\_o == 1].index

print(len(repeat\_indices))

repeat\_random\_indices = np.random.choice(repeat\_indices, int(train\_size[i] \* 0.05))

print(len(repeat\_random\_indices))

not\_repeat\_indices = y\_train\_o[y\_train\_o == 0].index

not\_repeat\_random\_indices = np.random.choice(not\_repeat\_indices, int(train\_size[1] \* 0.20))

x\_train = x\_train\_o.iloc[np.concatenate((repeat\_random\_indices, not\_repeat\_random\_indices))]

print(x\_train.shape)

y\_train = y\_train\_o.iloc[np.concatenate((repeat\_random\_indices, not\_repeat\_random\_indices))]

print(y\_train.shape)

x\_test = x.iloc[-test\_num:]

y\_test = y.iloc[-test\_num:]

### scale the data first

scaler = preprocessing.StandardScaler().fit(x\_train)

scaler.transform(x\_train)

scaler.transform(x\_test)

### Ada models-----------------------------------------------------------------------------------------

model\_Ada = AdaBoostClassifier( base\_estimator = RandomForestClassifier(n\_estimators=70, min\_samples\_split=10, min\_samples\_leaf=10,max\_depth =22,max\_features='sqrt' ,random\_state=10,class\_weight = 'balanced'),

algorithm="SAMME",

n\_estimators=200, learning\_rate=0.1)

x\_train\_Ada = x\_train

y\_train\_Ada = y\_train

x\_test\_Ada = x\_test

model\_Ada.fit(x\_train\_Ada,y\_train\_Ada)

pre\_Ada = model\_Ada.predict\_proba(x\_test\_Ada)

###SVM returns only label

# model = svm.SVC(kernel = 'linear', decision\_function\_shape = 'ovr')

### RandomForest could return the proba

# model = RandomForestClassifier(n\_estimators=70, min\_samples\_split=10, min\_samples\_leaf=10,max\_depth =22,max\_features='sqrt' ,random\_state=10,class\_weight = 'balanced')

###GradientBoosting return the proba

# model = GradientBoostingClassifier()

###adaboost could return the proba

# model = AdaBoostClassifier( base\_estimator = RandomForestClassifier(n\_estimators=70, min\_samples\_split=10, min\_samples\_leaf=10,max\_depth =22,max\_features='sqrt' ,random\_state=10,class\_weight = 'balanced'),

#algorithm="SAMME",

#n\_estimators=50, learning\_rate=0.1)

###grid search for Ada

#find n\_estimators

'''print('find n\_estimators')

param\_test1 = { 'n\_estimators':[50,80,100,150,200,250,300,400,500]}

gsearch1 = GridSearchCV(estimator = AdaBoostClassifier( base\_estimator = RandomForestClassifier(n\_estimators=70, min\_samples\_split=10, min\_samples\_leaf=10,max\_depth =22,max\_features='sqrt' ,random\_state=10,class\_weight = 'balanced'),

algorithm="SAMME",

learning\_rate=0.1),

param\_grid = param\_test1, scoring='roc\_auc',cv=5)

gsearch1.fit(x\_train, y\_train)

print(gsearch1.grid\_scores\_, gsearch1.best\_params\_, gsearch1.best\_score\_)

'''

#find learning rate

print('find learning rate')

param\_test2 = { 'learning\_rate':[0.001,0.01,0.1,0.3,0.5,0.8,1.0,5.0,10.0]}

gsearch2 = GridSearchCV(estimator = AdaBoostClassifier( base\_estimator = RandomForestClassifier(n\_estimators=70, min\_samples\_split=10, min\_samples\_leaf=10,max\_depth =22,max\_features='sqrt' ,random\_state=10,class\_weight = 'balanced'),

algorithm="SAMME",

n\_estimators = 200),

param\_grid = param\_test2, scoring='roc\_auc',cv=5)

gsearch2.fit(x\_train, y\_train)

print(gsearch2.grid\_scores\_, gsearch2.best\_params\_, gsearch2.best\_score\_)

# confusion matrix

'''confusion = confusion\_matrix(y\_test, predictions)

print("\t" + "\t".join(str(x) for x in range(0, 2)))

print("".join(["-"] \* 50))

for ii in range(0, 2):

jj = ii

print("%i:\t" % jj + "\t".join(str(confusion[ii][x]) for x in range(0, 2)))

print(pre)

'''

#print(f1\_score(y\_test, predictions))

#print(precision\_score(y\_test, predictions))

#print(recall\_score(y\_test, predictions))

# print(roc\_auc\_score(y\_test, predictions))

# print(classification\_report(y\_test, predictions))

fpr, tpr, thresholds = roc\_curve(y\_test, pre\_Ada[:,1])

roc\_auc = auc(fpr, tpr)

print(roc\_auc)

print('Finish!')

if \_\_name\_\_ == "\_\_main\_\_":

train\_ui\_up\_mp\_um\_double\_pca\_sim\_slope\_fillna\_features = features()

classification(train\_ui\_up\_mp\_um\_double\_pca\_sim\_slope\_fillna\_features)