from sklearn.tree import DecisionTreeClassifier

用上述的 DecisionTreeClassifier 构建决策树模型，对下述 4 个变量进行调优：

max_depth

min_samples_split

min_samples_leaf

max_features

比较决策树和随机森林模型的准确度（用代码中的 ROC_AUC 的结果衡量，越高越好）

改写代码后，经过两种模型的比较，随机森林的准确度还是要高不少的。

老师的代码稍作改动后如下，先看看随机森林的结果：

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn import metrics,cross_validation
from matplotlib import pyplot as plt

def Missingrate_Column(df, col):
    '''
    :param df:
    :param col:
    :return:
    '''
    missing_records = df[col].map(lambda x: int(x!=x))
    return missing_records.mean()


def Makeup_Missing(df,col, makeup_value):
    '''
    :param df:
    :param col:
    :return:
    '''
    raw_values = list(df[col])
    missing_position = [i for i in range(len(raw_values)) if raw_values[i] != raw_values[i]]
    for i in missing_position:
        raw_values[i] = makeup_value
    return raw_values
```

```
def Avg_Calc(numerator, denominator):
    if denominator == 0:
        return 0
    else:
        return numerator/denominator

def ROC_AUC(df, score, target, plot=True):
    df2 = df.copy()
    s = list(set(df2[score]))
    s.sort()
    tpr_list = [0]
    fpr_list = [0]
    for k in s:
        df2['label_temp'] = df[score].map(lambda x: int(x >= k))
        TP = df2[(df2.label_temp==1) & (df2[target]==1)].shape[0]
        FN = df2[(df2.label_temp == 1) & (df2[target] == 0)].shape[0]
        FP = df2[(df2.label_temp == 0) & (df2[target] == 1)].shape[0]
        TN = df2[(df2.label_temp == 0) & (df2[target] == 0)].shape[0]
        try:
            TPR = TP / (TP + FN)
        except:
            TPR =0
        try:
            FPR = FP / (FP + TN)
        except:
            FPR = 0
        tpr_list.append(TPR)
        fpr_list.append(FPR)
    tpr_list.append(1)
    fpr_list.append(1)
    ROC_df = pd.DataFrame({'tpr': tpr_list, 'fpr': fpr_list})
    ROC_df = ROC_df.sort_values(by='tpr')
    ROC_df = ROC_df.drop_duplicates()
    auc = 0
    ROC_mat = np.mat(ROC_df)
    for i in range(1, ROC_mat.shape[0]):
        auc = auc + (ROC_mat[i, 1] + ROC_mat[i - 1, 1]) * (ROC_mat[i, 0] - ROC_mat[i - 1, 0]) * 0.5
    if plot:
        plt.plot(ROC_df['fpr'], ROC_df['tpr'])
        plt.plot([0, 1], [0, 1])
        plt.title("AUC={}%".format(int(auc * 100)))
```

```
        plt.show()
    return auc


def KS(df, score, target, plot = True):
    '''
    :param df: 包含目标变量与预测值的数据集
    :param score: 得分或者概率
    :param target: 目标变量
    :return: KS 值
    :return: KS 值
    '''
    total = df.groupby([score])[target].count()
    bad = df.groupby([score])[target].sum()
    all = pd.DataFrame({'total':total, 'bad':bad})
    all['good'] = all['total'] - all['bad']
    all[score] = all.index
    all = all.sort_values(by=score, ascending = False)
    all.index = range(len(all))
    all['badCumRate'] = all['bad'].cumsum() / all['bad'].sum()
    all['goodCumRate'] = all['good'].cumsum() / all['good'].sum()
    KS_list = all.apply(lambda x: x.badCumRate - x.goodCumRate, axis=1)
    KS = max(KS_list)
    if plot:
        plt.plot(list(all.index), all['badCumRate'])
        plt.plot(list(all.index), all['goodCumRate'])
        plt.title('KS ={}%'.format(int(KS*100)))
        plt.show()
    return KS



########################
####  1，读取数据  #####
########################
#folderOfData = '/Users/Code/Data Collections/AF/'
#data = pd.read_csv(folderOfData + 'anti_fraud_data.csv', header = 0)
data = pd.read_csv('lesson5data/anti_fraud_data.csv', header = 0)
del data['ID']
train_data, test_data = train_test_split(data, test_size=0.3)

#########################
####  2，数据预处理  #####
#########################
```

```
all_columns = list(train_data.columns)
all_columns.remove('flag')

#查看每个字段的缺失率
column_missingrate = {col: Missingrate_Column(train_data, col) for col in all_columns}
column_MR_df = pd.DataFrame.from_dict(column_missingrate, orient='index')
column_MR_df.columns = ['missing_rate']
column_MR_df_sorted = column_MR_df.sort_values(by='missing_rate', ascending=False)
#由于变量 ip_desc_danger 在训练集中全部缺失，故将其删去。
all_columns.remove('ip_desc_danger')
del train_data['ip_desc_danger']
column_MR_df_sorted = column_MR_df_sorted.drop(index=['ip_desc_danger'])
columns_with_missing = column_MR_df_sorted[column_MR_df_sorted.missing_rate > 0].index


categorical_cols_withmissing =
['area1_danger','registered_channels','sex','is_email_acct','area8_danger','area6_danger','area11
_danger']
dummy_map = {}
dummy_columns = []
for raw_col in categorical_cols_withmissing:
    dummies = pd.get_dummies(train_data.loc[:, raw_col], prefix=raw_col)
    col_onehot = pd.concat([train_data[raw_col], dummies], axis=1)
    col_onehot = col_onehot.drop_duplicates()
    train_data = pd.concat([train_data, dummies], axis=1)
    del train_data[raw_col]
    dummy_map[raw_col] = col_onehot
    dummy_columns = dummy_columns + list(dummies)

#对于数值型变量，可以将原始变量与表示缺失状态的示性变量交互地使用.此外，由于这些
变量都是非负数，对于缺失，可以用 0 来填补
continuous_cols_withmissing = [i for i in columns_with_missing if i not in
categorical_cols_withmissing]
for col in continuous_cols_withmissing:
    train_data[col+'_ismissing'] = train_data[col].map(lambda x: int(x!=x))
    train_data[col] = Makeup_Missing(train_data, col, 0)

#注意到，原始数据中，年龄 age 没有缺失值，但是有 0.需要将 0 看成缺失
train_data['age'+'_ismissing'] = train_data['age'].map(lambda x: int(x==0))



########################
```

```
####  3，特征衍生  #####
########################
#（1）构造平均值型变量
periods = ['10m','30m','1h','12h','1d','7d','15d','30d','60d','90d']
for period in periods:
    amount = period+'_Sum_pay_amount'
    times = period+'_pay_times'
    avg_payment = period+'_Avg_pay_amount'
    train_data[avg_payment] = train_data[[amount,times]].apply(lambda x:
Avg_Calc(x[amount],x[times]),axis=1)

#（2）构造变量，检查平均每次支付金额上升量
for i in range(len(periods)-1):
    avg_payment_1 = periods[i]+'_Avg_pay_amount'
    avg_payment_2 = periods[i+1] + '_Avg_pay_amount'
    increase_payment = periods[i] + '_' + periods[i+1] + '_payment_increase'
    train_data[increase_payment] = train_data[[avg_payment_1,avg_payment_2]].apply(lambda
x: x[avg_payment_1] - x[avg_payment_2],axis=1)


#（3）在（1）的基础上求最大的平均支付金额值
avg_payments = [d+'_Avg_pay_amount' for d in periods]
train_data['max_Avg_pay_amount'] = train_data[avg_payments].apply(lambda x: max(x),axis=1)
features = list(train_data.columns)
features.remove('flag')
X,y = train_data[features], train_data['flag']

############################
####  4，构建随机森林  #####
############################
#使用默认参数进行建模
RFC = RandomForestClassifier(oob_score=True)
RFC.fit(X,y)
print(RFC.oob_score_)
y_predprob = RFC.predict_proba(X)[:,1]
result = pd.DataFrame({'real':y,'pred':y_predprob})
#print("AUC Score (Train): %f" % metrics.roc_auc_score(y, y_predprob))
ROC_AUC(result, 'pred', 'real')


#参数调整
#1，调整 n_estimators
param_test1 = {'n_estimators':range(10,101,10)}
gsearch1 = GridSearchCV(estimator = RandomForestClassifier(),param_grid = param_test1,
```

```
scoring='roc_auc',cv=5)
gsearch1.fit(X,y)
best_n_estimators_1 = gsearch1.best_params_['n_estimators']    #80

param_test1 = {'n_estimators':range(71,89)}
gsearch1 = GridSearchCV(estimator = RandomForestClassifier(),param_grid = param_test1,
scoring='roc_auc',cv=5)
gsearch1.fit(X,y)
best_n_estimators = gsearch1.best_params_['n_estimators']    #84


#2，对决策树最大深度 max_depth,内部节点再划分所需最小样本数 min_samples_split 和叶
子节点最少样本数 min_samples_leaf 进行网格搜索
param_test2 = {'max_depth':range(5,15), 'min_samples_split':range(20,81,10),
'min_samples_leaf':range(5,21,5)}
gsearch2 = GridSearchCV(estimator = RandomForestClassifier(n_estimators=
best_n_estimators),param_grid = param_test2, scoring='roc_auc',cv=5)
gsearch2.fit(X,y)
best_max_depth, best_min_samples_split, best_min_samples_leaf =
gsearch2.best_params_['max_depth'],gsearch2.best_params_['min_samples_leaf'],gsearch2.best
_params_['min_samples_split']
print('best_max_depth',best_max_depth)
print('best_min_samples_split',best_min_samples_split)
print('best_min_samples_leaf',best_min_samples_leaf)

#3，对 max_features 进行调优
param_test3 ={'max_features':['sqrt','log2']}
gsearch3 = GridSearchCV(estimator = RandomForestClassifier(n_estimators= best_n_estimators,
                                                           max_depth =
best_max_depth,
                                                           min_samples_split =
best_min_samples_split,
                                                           min_samples_leaf =
best_min_samples_leaf),
                        param_grid = param_test3, scoring='roc_auc',cv=5)
gsearch3.fit(X,y)
best_max_features = gsearch3.best_params_['max_features']
print('best_max_features',best_max_features)
RFC_2 = RandomForestClassifier(oob_score=True, n_estimators= best_n_estimators,
                               max_depth = best_max_depth,min_samples_split =
best_min_samples_split,
                               min_samples_leaf = best_min_samples_leaf,max_features =
best_max_features)
RFC_2.fit(X,y)
```

```
print(RFC_2.oob_score_)
y_predprob = RFC_2.predict_proba(X)[:,1]
result = pd.DataFrame({'real':y,'pred':y_predprob})
#print("AUC Score (Train): %f" % metrics.roc_auc_score(y, y_predprob))
ROC_AUC(result, 'pred', 'real')


#特征重要性评估
fi = RFC_2.feature_importances_
fi = sorted(fi, reverse=True)
plt.bar(list(range(len(fi))), fi)
plt.title('feature importance')
plt.show()


#################################
####  5，在测试集上进行测试  #####
#################################
#准备测试样本#
del test_data['ip_desc_danger']
#在对测试集进行哑变量编码或者独热编码的时候，要按照在训练集中的编码方式来进行
#例如，在训练集中，设备类型={Android， Apple， SDK}，但是在测试集中设备类型={Android，
Apple， SDK， PC}。多出来的值在编码中全部为 0
train_data, test_data = train_test_split(data, test_size=0.3)
test_data_cp = test_data.copy()

for raw_col in categorical_cols_withmissing:
    test_data = pd.merge(test_data, dummy_map[raw_col],on =raw_col, how='left')
    del test_data[raw_col]
dummy_columns  = test_data[dummy_columns]
dummy_columns.isnull().any()


for col in continuous_cols_withmissing:
    test_data[col+'_ismissing']    = test_data[col].map(lambda x: int(x!=x))
    test_data[col] = Makeup_Missing(test_data, col, 0)

#注意到，原始数据中，年龄 age 没有缺失值，但是有 0.需要将 0 看成缺失
test_data['age'+'_ismissing']    = test_data['age'].map(lambda x: int(x==0))
for period in periods:
    amount = period+'_Sum_pay_amount'
    times = period+'_pay_times'
    avg_payment = period+'_Avg_pay_amount'
    test_data[avg_payment]          =          test_data[[amount,times]].apply(lambda          x:
```

```
Avg_Calc(x[amount],x[times]),axis=1)

#（2）构造变量，检查平均每次支付金额上升量
for i in range(len(periods)-1):
    avg_payment_1 = periods[i]+'_Avg_pay_amount'
    avg_payment_2 = periods[i+1] + '_Avg_pay_amount'
    increase_payment = periods[i] + '_' + periods[i+1] + '_payment_increase'
    test_data[increase_payment] = test_data[[avg_payment_1,avg_payment_2]].apply(lambda x:
x[avg_payment_1] - x[avg_payment_2],axis=1)


#（3）在（1）的基础上求最大的平均支付金额值
avg_payments = [d+'_Avg_pay_amount' for d in periods]
test_data['max_Avg_pay_amount'] = test_data[avg_payments].apply(lambda x: max(x),axis=1)

# 用默认参数的随机森林进行建模
X_test,y_test = test_data[features], test_data['flag']
y_predprob = RFC.predict_proba(X_test)[:,1]
result = pd.DataFrame({'real':y_test,'pred':y_predprob})
ROC_AUC(result, 'pred', 'real')

# 用调优后的随机森林进行建模
y_predprob2 = RFC_2.predict_proba(X_test)[:,1]
result = pd.DataFrame({'real':y_test,'pred':y_predprob2})
ROC_AUC(result, 'pred', 'real')

KS(result, 'pred', 'real')
```

输出：

```
best_max_depth 14
best_min_samples_split  15
best_min_samples_leaf  20
best_max_features sqrt
```

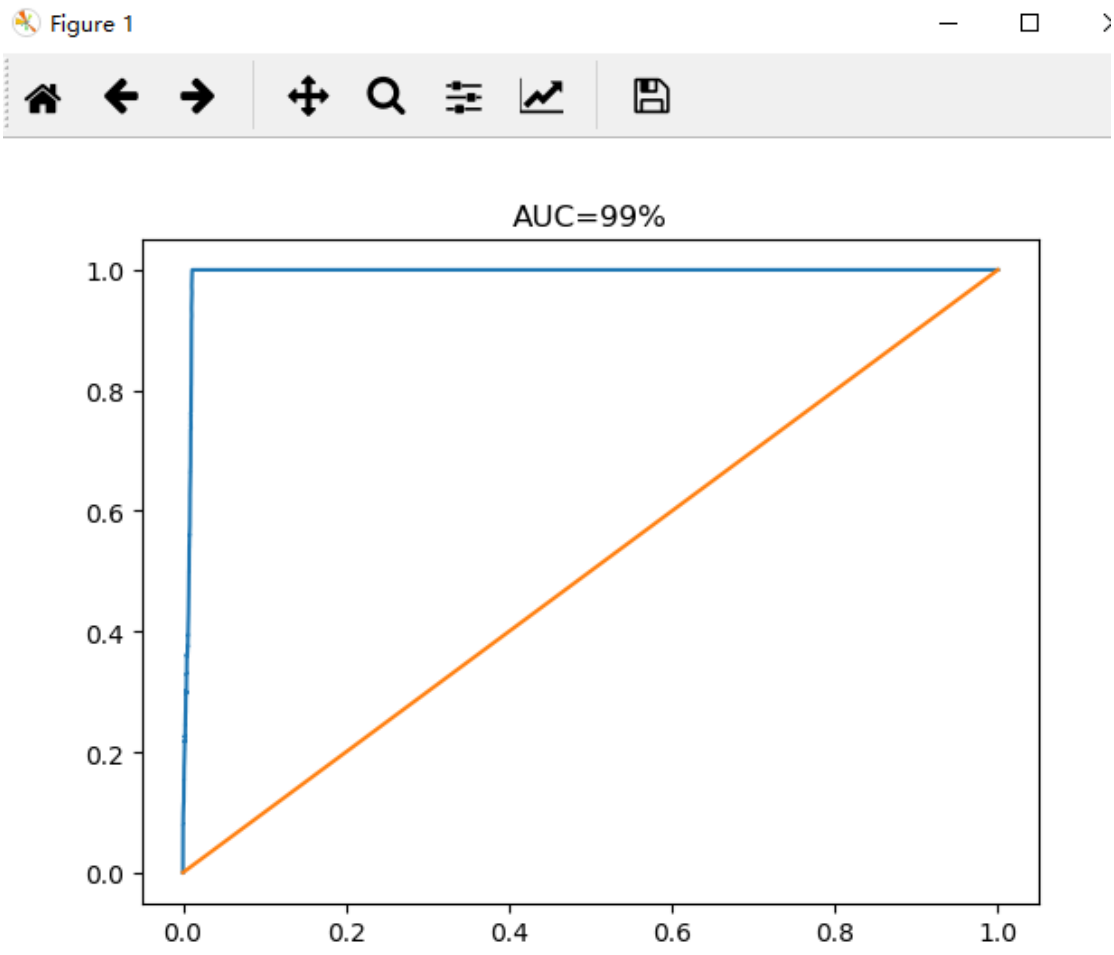


Figure 1
AUC=99%
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

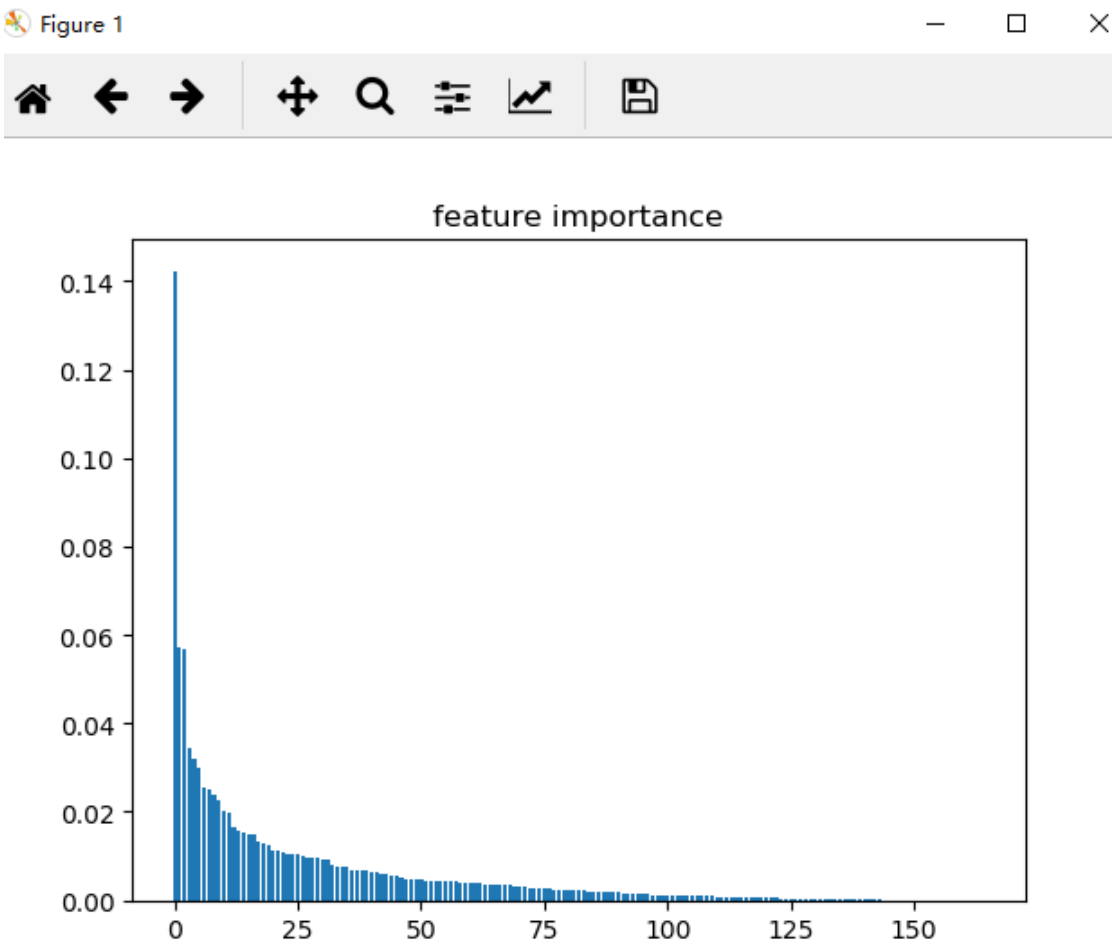


Figure 1
feature importance
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
25
50
75
100
125
150

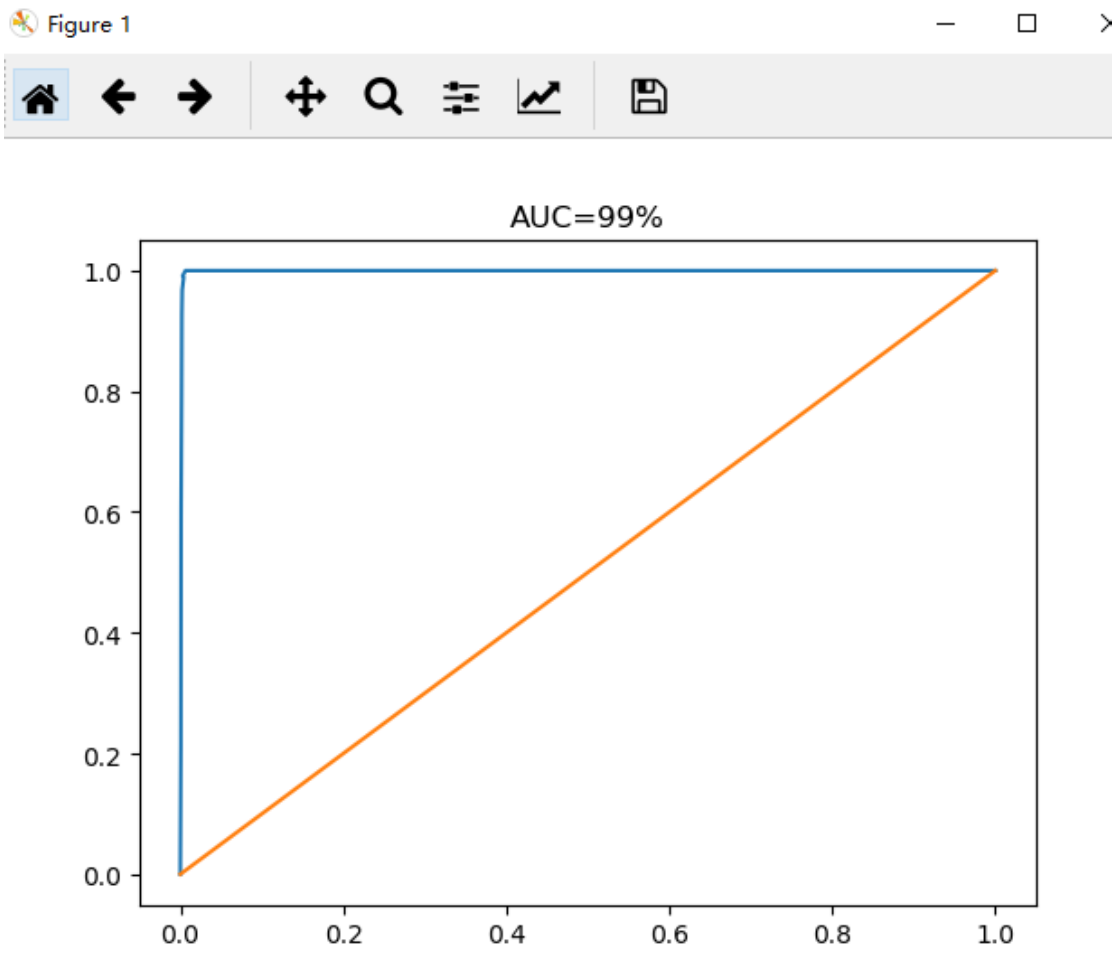


Figure 1
AUC=99%
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

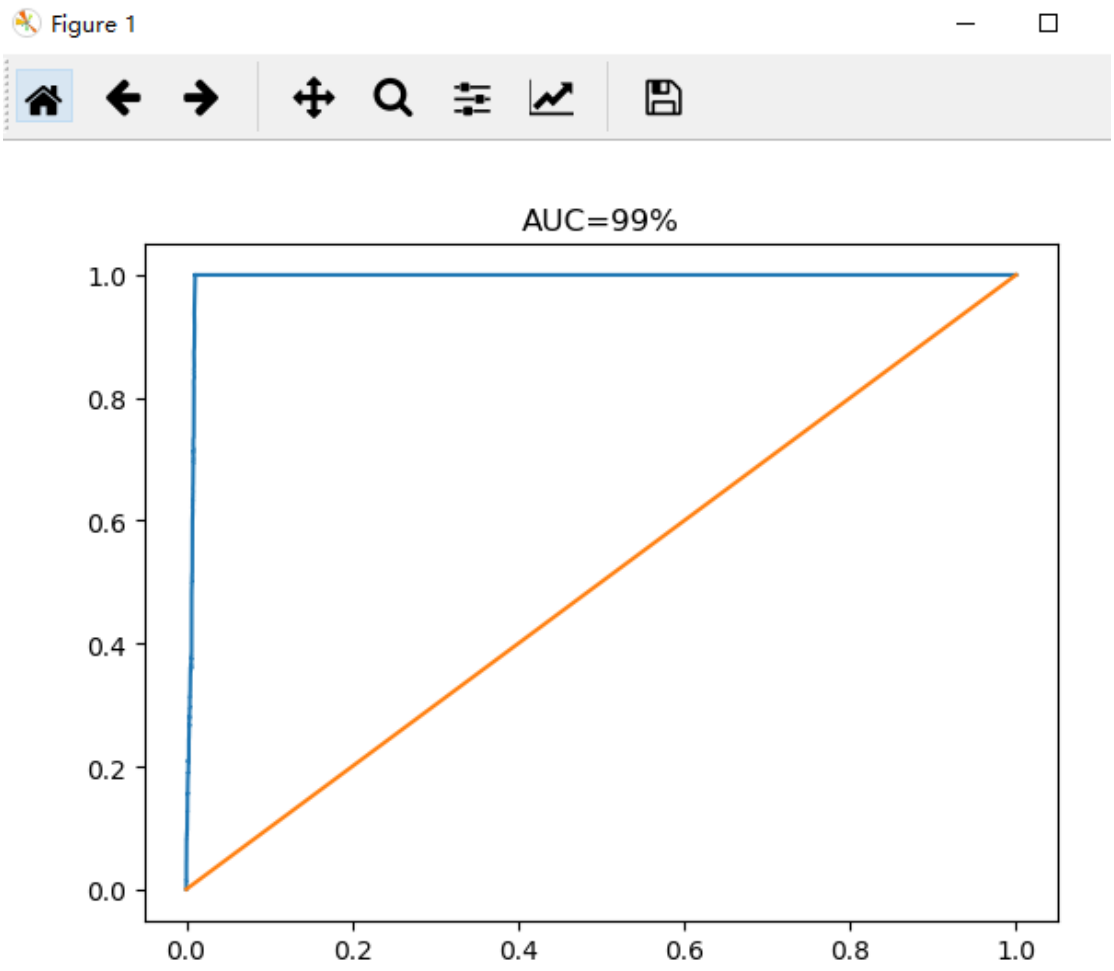
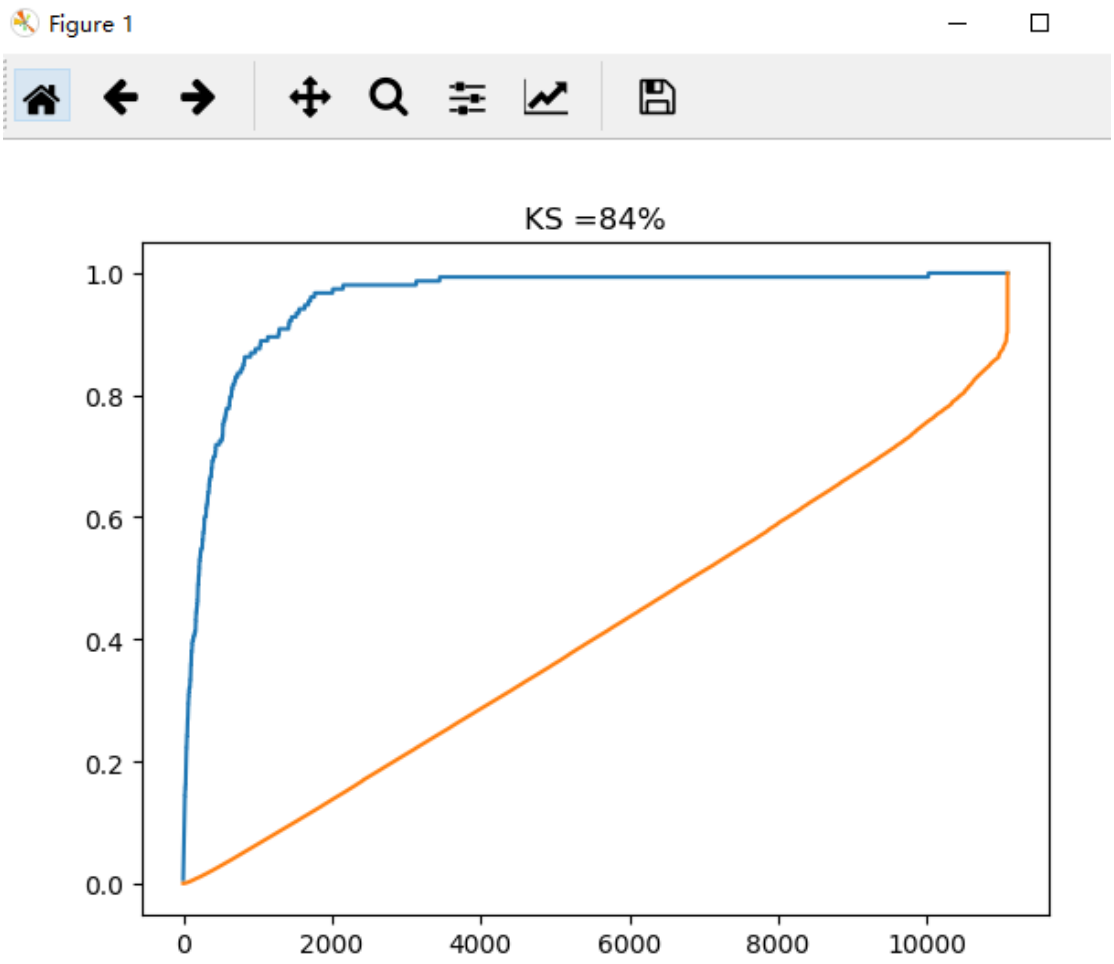


Figure 1
AUC=99%
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



用 DecisionTreeClassifier 的包替换 RandomForestClassifier 以后的代码如下：

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn import metrics,cross_validation
from matplotlib import pyplot as plt

def Missingrate_Column(df, col):
    '''
    :param df:
    :param col:
    :return:
```

```
    '''
    missing_records = df[col].map(lambda x: int(x!=x))
    return missing_records.mean()


def Makeup_Missing(df,col, makeup_value):
    '''
    :param df:
    :param col:
    :return:
    '''
    raw_values = list(df[col])
    missing_position = [i for i in range(len(raw_values)) if raw_values[i] != raw_values[i]]
    for i in missing_position:
        raw_values[i] = makeup_value
    return raw_values



def Avg_Calc(numerator, denominator):
    if denominator == 0:
        return 0
    else:
        return numerator/denominator

def ROC_AUC(df, score, target, plot=True):
    df2 = df.copy()
    s = list(set(df2[score]))
    s.sort()
    tpr_list = [0]
    fpr_list = [0]
    for k in s:
        df2['label_temp'] = df[score].map(lambda x: int(x >= k))
        TP = df2[(df2.label_temp==1) & (df2[target]==1)].shape[0]
        FN = df2[(df2.label_temp == 1) & (df2[target] == 0)].shape[0]
        FP = df2[(df2.label_temp == 0) & (df2[target] == 1)].shape[0]
        TN = df2[(df2.label_temp == 0) & (df2[target] == 0)].shape[0]
        try:
            TPR = TP / (TP + FN)
        except:
            TPR =0
        try:
            FPR = FP / (FP + TN)
        except:
```

```
                FPR = 0
            tpr_list.append(TPR)
            fpr_list.append(FPR)
        tpr_list.append(1)
        fpr_list.append(1)
        ROC_df = pd.DataFrame({'tpr': tpr_list, 'fpr': fpr_list})
        ROC_df = ROC_df.sort_values(by='tpr')
        ROC_df = ROC_df.drop_duplicates()
        auc = 0
        ROC_mat = np.mat(ROC_df)
        for i in range(1, ROC_mat.shape[0]):
            auc = auc + (ROC_mat[i, 1] + ROC_mat[i - 1, 1]) * (ROC_mat[i, 0] - ROC_mat[i - 1, 0]) * 0.5
        if plot:
            plt.plot(ROC_df['fpr'], ROC_df['tpr'])
            plt.plot([0, 1], [0, 1])
            plt.title("AUC={}%".format(int(auc * 100)))
            plt.show()
        return auc


def KS(df, score, target, plot = True):
    '''
    :param df: 包含目标变量与预测值的数据集
    :param score: 得分或者概率
    :param target: 目标变量
    :return: KS 值
    :return: KS 值
    '''
    total = df.groupby([score])[target].count()
    bad = df.groupby([score])[target].sum()
    all = pd.DataFrame({'total':total, 'bad':bad})
    all['good'] = all['total'] - all['bad']
    all[score] = all.index
    all = all.sort_values(by=score, ascending = False)
    all.index = range(len(all))
    all['badCumRate'] = all['bad'].cumsum() / all['bad'].sum()
    all['goodCumRate'] = all['good'].cumsum() / all['good'].sum()
    KS_list = all.apply(lambda x: x.badCumRate - x.goodCumRate, axis=1)
    KS = max(KS_list)
    if plot:
        plt.plot(list(all.index), all['badCumRate'])
        plt.plot(list(all.index), all['goodCumRate'])
        plt.title('KS ={}%'.format(int(KS*100)))
        plt.show()
```

```
    return KS


########################
####  1，读取数据  #####
########################
#folderOfData = '/Users/Code/Data Collections/AF/'
#data = pd.read_csv(folderOfData + 'anti_fraud_data.csv', header = 0)
data = pd.read_csv('lesson5data/anti_fraud_data.csv', header = 0)
del data['ID']
train_data, test_data = train_test_split(data, test_size=0.3)

#########################
####  2，数据预处理  #####
#########################

all_columns = list(train_data.columns)
all_columns.remove('flag')

#查看每个字段的缺失率
column_missingrate = {col: Missingrate_Column(train_data, col) for col in all_columns}
column_MR_df = pd.DataFrame.from_dict(column_missingrate, orient='index')
column_MR_df.columns = ['missing_rate']
column_MR_df_sorted = column_MR_df.sort_values(by='missing_rate', ascending=False)
#由于变量 ip_desc_danger 在训练集中全部缺失，故将其删去。
all_columns.remove('ip_desc_danger')
del train_data['ip_desc_danger']
column_MR_df_sorted = column_MR_df_sorted.drop(index=['ip_desc_danger'])
columns_with_missing = column_MR_df_sorted[column_MR_df_sorted.missing_rate > 0].index


categorical_cols_withmissing =
['area1_danger','registered_channels','sex','is_email_acct','area8_danger','area6_danger','area11
_danger']
dummy_map = {}
dummy_columns = []
for raw_col in categorical_cols_withmissing:
    dummies = pd.get_dummies(train_data.loc[:, raw_col], prefix=raw_col)
    col_onehot = pd.concat([train_data[raw_col], dummies], axis=1)
    col_onehot = col_onehot.drop_duplicates()
    train_data = pd.concat([train_data, dummies], axis=1)
    del train_data[raw_col]
    dummy_map[raw_col] = col_onehot
```

```
    dummy_columns = dummy_columns + list(dummies)

#对于数值型变量，可以将原始变量与表示缺失状态的示性变量交互地使用.此外，由于这些变量都是非负数，对于缺失，可以用 0 来填补
continuous_cols_withmissing = [i for i in columns_with_missing if i not in categorical_cols_withmissing]
for col in continuous_cols_withmissing:
    train_data[col+'_ismissing'] = train_data[col].map(lambda x: int(x!=x))
    train_data[col] = Makeup_Missing(train_data, col, 0)

#注意到，原始数据中，年龄 age 没有缺失值，但是有 0.需要将 0 看成缺失
train_data['age'+'_ismissing'] = train_data['age'].map(lambda x: int(x==0))



########################
####  3，特征衍生  #####
########################
#（1）构造平均值型变量
periods = ['10m','30m','1h','12h','1d','7d','15d','30d','60d','90d']
for period in periods:
    amount = period+'_Sum_pay_amount'
    times = period+'_pay_times'
    avg_payment = period+'_Avg_pay_amount'
    train_data[avg_payment] = train_data[[amount,times]].apply(lambda x: Avg_Calc(x[amount],x[times]),axis=1)

#（2）构造变量，检查平均每次支付金额上升量
for i in range(len(periods)-1):
    avg_payment_1 = periods[i]+'_Avg_pay_amount'
    avg_payment_2 = periods[i+1] + '_Avg_pay_amount'
    increase_payment = periods[i] + '_' + periods[i+1] + '_payment_increase'
    train_data[increase_payment] = train_data[[avg_payment_1,avg_payment_2]].apply(lambda x: x[avg_payment_1] - x[avg_payment_2],axis=1)


#（3）在（1）的基础上求最大的平均支付金额值
avg_payments = [d+'_Avg_pay_amount' for d in periods]
train_data['max_Avg_pay_amount'] = train_data[avg_payments].apply(lambda x: max(x),axis=1)
features = list(train_data.columns)
features.remove('flag')
X,y = train_data[features], train_data['flag']

#########################
```

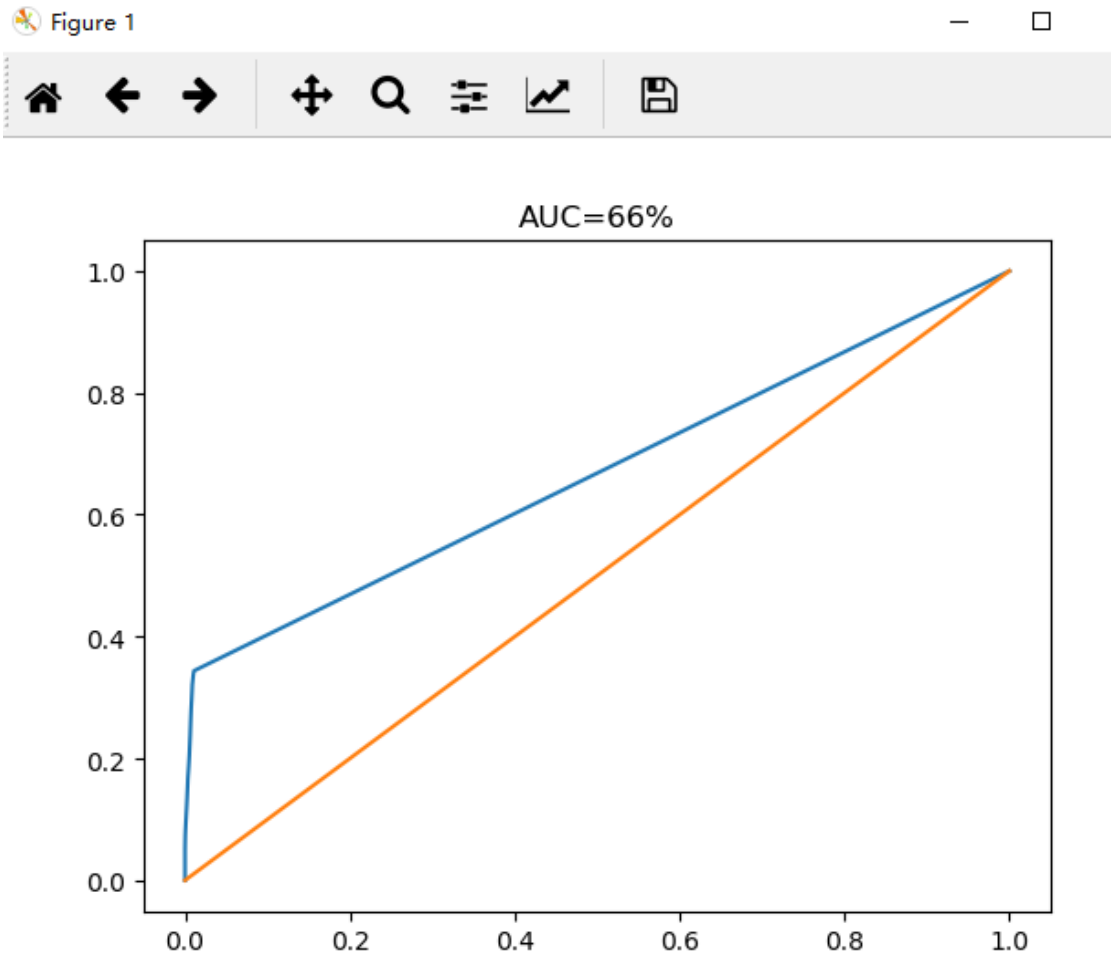
```
####  4，构建随机森林  #####
############################
#使用默认参数进行建模
# RFC = DecisionTreeClassifier(oob_score=True)
# RFC.fit(X,y)
# print(RFC.oob_score_)
# y_predprob = RFC.predict_proba(X)[:,1]
# result = pd.DataFrame({'real':y,'pred':y_predprob})
# #print("AUC Score (Train): %f" % metrics.roc_auc_score(y, y_predprob))
# ROC_AUC(result, 'pred', 'real')


#参数调整
#1，调整 n_estimators
# param_test1 = {'n_estimators':range(10,101,10)}
# gsearch1 = GridSearchCV(estimator = DecisionTreeClassifier(),param_grid = param_test1, scoring='roc_auc',cv=5)
# gsearch1.fit(X,y)
# best_n_estimators_1 = gsearch1.best_params_['n_estimators']  #80
#
# param_test1 = {'n_estimators':range(71,89)}
# gsearch1 = GridSearchCV(estimator = DecisionTreeClassifier(),param_grid = param_test1, scoring='roc_auc',cv=5)
# gsearch1.fit(X,y)
# best_n_estimators = gsearch1.best_params_['n_estimators']  #84
#
#
##2，对决策树最大深度 max_depth,内部节点再划分所需最小样本数 min_samples_split 和叶子节点最少样本数 min_samples_leaf 进行网格搜索
# param_test2 = {'max_depth':range(5,15), 'min_samples_split':range(20,81,10), 'min_samples_leaf':range(5,21,5)}
# gsearch2 = GridSearchCV(estimator = DecisionTreeClassifier(n_estimators= best_n_estimators),param_grid = param_test2, scoring='roc_auc',cv=5)
# gsearch2.fit(X,y)
# best_max_depth, best_min_samples_split, best_min_samples_leaf = gsearch2.best_params_['max_depth'],gsearch2.best_params_['min_samples_leaf'],gsearch2.best_params_['min_samples_split']
# print('best_max_depth=',best_max_depth)
# print('best_min_samples_split',best_min_samples_split)
# print('best_min_samples_leaf',best_min_samples_leaf)
#
## #3，对 max_features 进行调优
# param_test3 ={'max_features':['sqrt','log2']}
# gsearch3 = GridSearchCV(estimator = DecisionTreeClassifier(n_estimators= best_n_estimators,
```

```
#                                                                                 max_depth =
best_max_depth,
#                                                                             min_samples_split  =
best_min_samples_split,
#                                                                              min_samples_leaf  =
best_min_samples_leaf),
#                                      param_grid = param_test3, scoring='roc_auc',cv=5)
# gsearch3.fit(X,y)
# best_max_features = gsearch3.best_params_['max_features']
# print('best_max_features',best_max_features)
# RFC_2 = DecisionTreeClassifier(oob_score=True,  n_estimators= best_n_estimators,
#                                               max_depth  =  best_max_depth,min_samples_split  =
best_min_samples_split,
#                                         min_samples_leaf  = best_min_samples_leaf,max_features  =
best_max_features)

RFC_2 = DecisionTreeClassifier(
                                      max_depth = 14,min_samples_split  = 15,
                                      min_samples_leaf  = 20,max_features = 'sqrt')
RFC_2.fit(X,y)
#print(RFC_2.oob_score_)
y_predprob = RFC_2.predict_proba(X)[:,1]
result = pd.DataFrame({'real':y,'pred':y_predprob})
#print("AUC Score (Train): %f" % metrics.roc_auc_score(y, y_predprob))
ROC_AUC(result, 'pred', 'real')


#特征重要性评估
fi = RFC_2.feature_importances_
fi = sorted(fi, reverse=True)
plt.bar(list(range(len(fi))), fi)
plt.title('feature importance')
plt.show()


#################################
####  5，在测试集上进行测试  #####
#################################
#准备测试样本#
del test_data['ip_desc_danger']
#在对测试集进行哑变量编码或者独热编码的时候，要按照在训练集中的编码方式来进行
#例如，在训练集中，设备类型={Android， Apple， SDK}，但是在测试集中设备类型={Android，
Apple， SDK， PC}。多出来的值在编码中全部为 0
train_data, test_data = train_test_split(data, test_size=0.3)
```

```
test_data_cp = test_data.copy()

for raw_col in categorical_cols_withmissing:
    test_data = pd.merge(test_data, dummy_map[raw_col],on =raw_col, how='left')
    del test_data[raw_col]
dummy_columns = test_data[dummy_columns]
dummy_columns.isnull().any()



for col in continuous_cols_withmissing:
    test_data[col+'_ismissing'] = test_data[col].map(lambda x: int(x!=x))
    test_data[col] = Makeup_Missing(test_data, col, 0)

#注意到，原始数据中，年龄 age 没有缺失值，但是有 0.需要将 0 看成缺失
test_data['age'+'_ismissing'] = test_data['age'].map(lambda x: int(x==0))
for period in periods:
    amount = period+'_Sum_pay_amount'
    times = period+'_pay_times'
    avg_payment = period+'_Avg_pay_amount'
    test_data[avg_payment] = test_data[[amount,times]].apply(lambda x:
Avg_Calc(x[amount],x[times]),axis=1)

#（2）构造变量，检查平均每次支付金额上升量
for i in range(len(periods)-1):
    avg_payment_1 = periods[i]+'_Avg_pay_amount'
    avg_payment_2 = periods[i+1] + '_Avg_pay_amount'
    increase_payment = periods[i] + '_' + periods[i+1] + '_payment_increase'
    test_data[increase_payment] = test_data[[avg_payment_1,avg_payment_2]].apply(lambda x:
x[avg_payment_1] - x[avg_payment_2],axis=1)



#（3）在（1）的基础上求最大的平均支付金额值
avg_payments = [d+'_Avg_pay_amount' for d in periods]
test_data['max_Avg_pay_amount'] = test_data[avg_payments].apply(lambda x: max(x),axis=1)

# 用默认参数的随机森林进行建模
X_test,y_test = test_data[features], test_data['flag']
# y_predprob = RFC.predict_proba(X_test)[:,1]
# result = pd.DataFrame({'real':y_test,'pred':y_predprob})
# ROC_AUC(result, 'pred', 'real')

# 用调优后的随机森林进行建模
y_predprob2 = RFC_2.predict_proba(X_test)[:,1]
result = pd.DataFrame({'real':y_test,'pred':y_predprob2})
```

ROC_AUC(result, 'pred', 'real')

KS(result, 'pred', 'real')



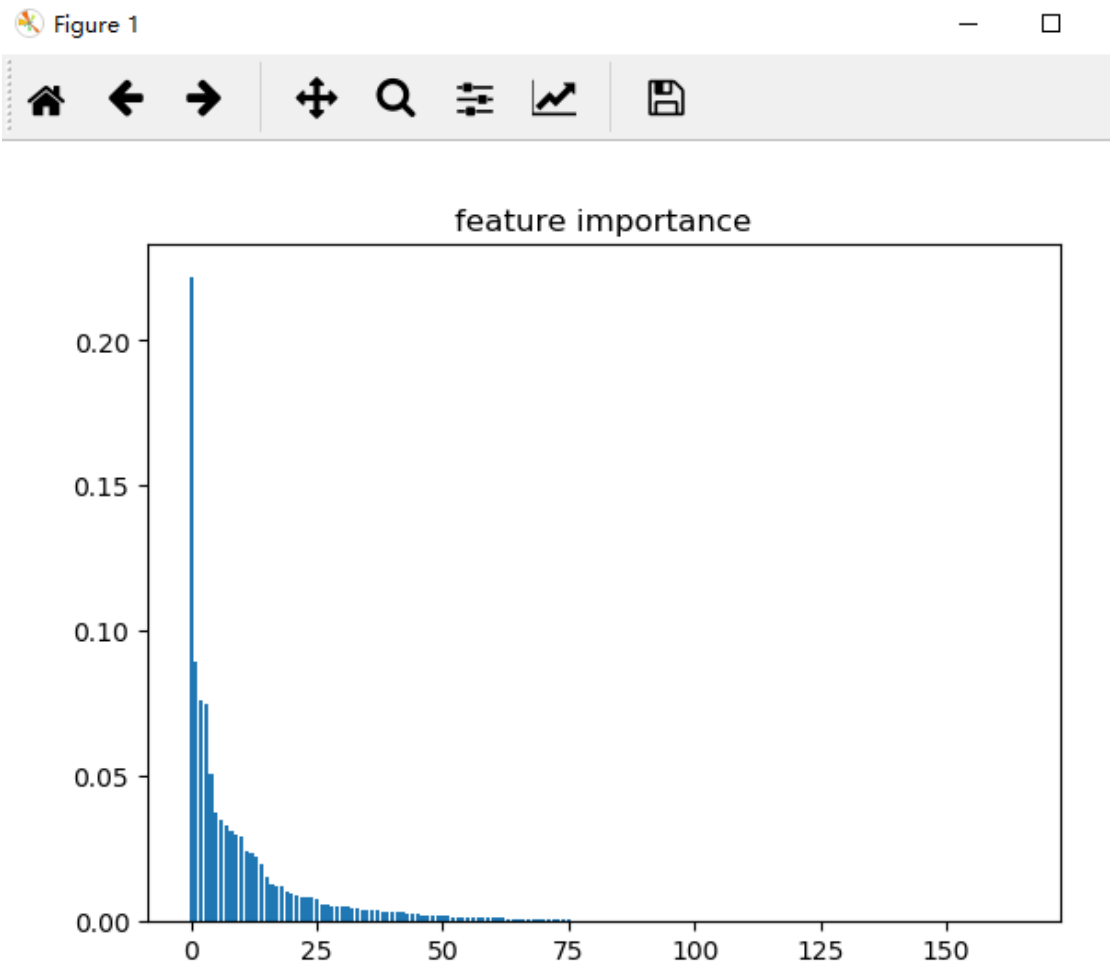


Figure 1
feature importance
0.20
0.15
0.10
0.05
0.00
0
25
50
75
100
125
150

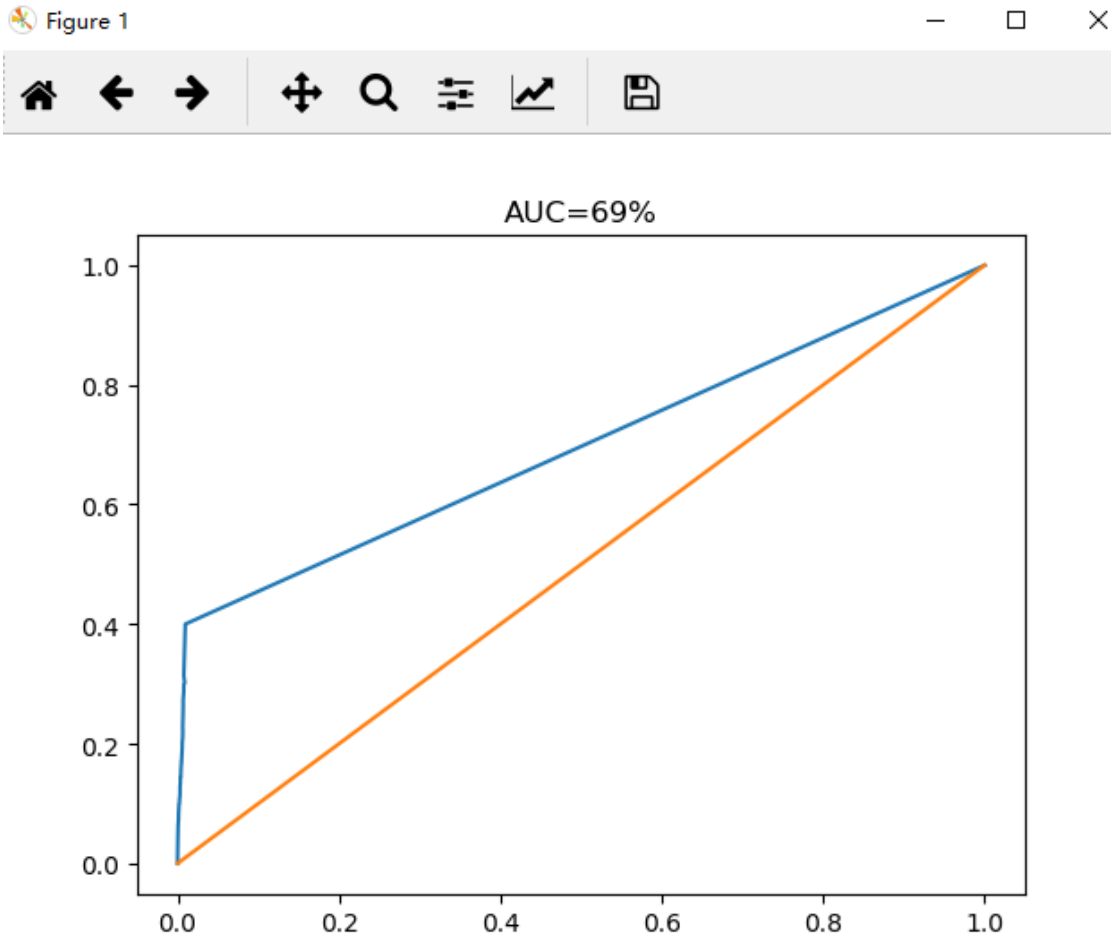


Figure 1
AUC=69%
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

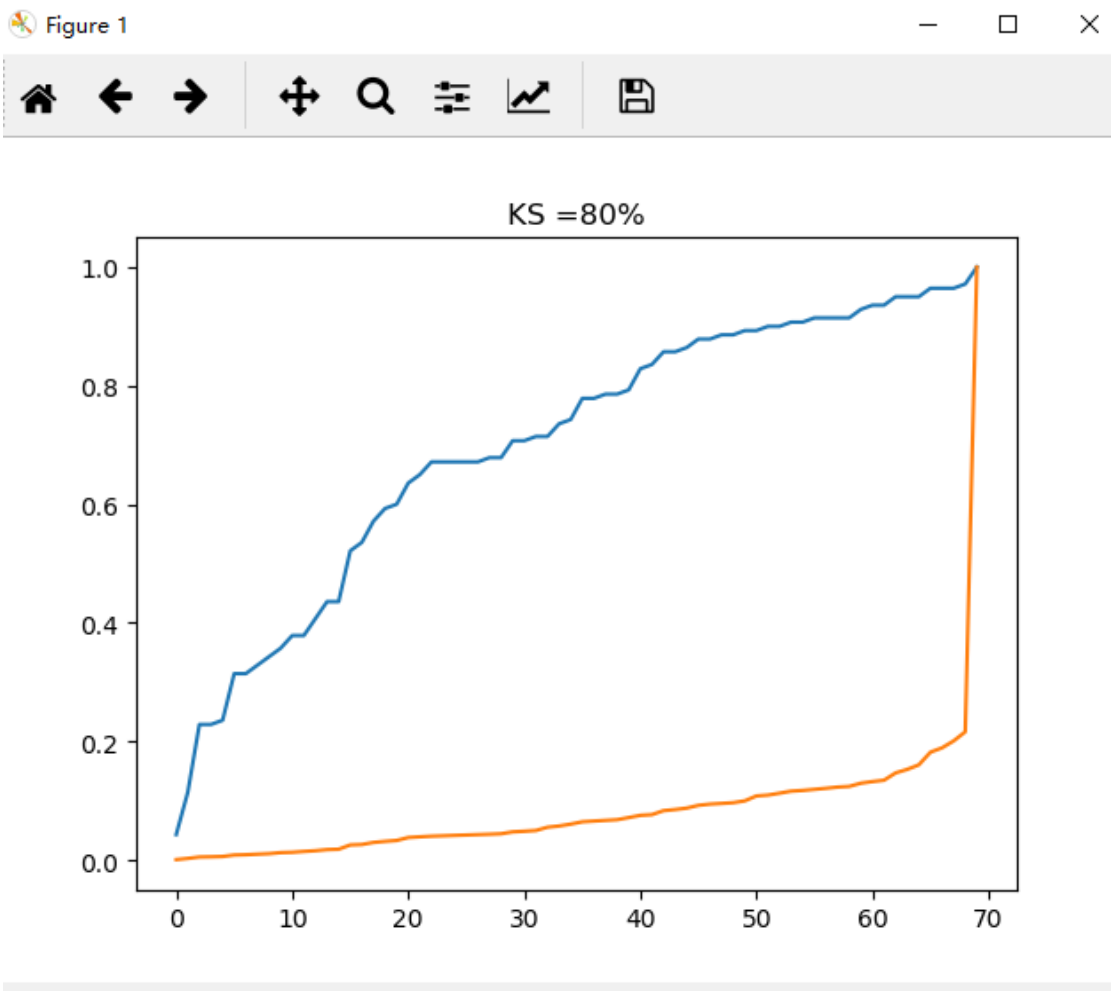


Figure 1
KS =80%
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70