数据科学导论实验报告

1. 比赛名称

2021年第九届CCF大数据与计算智能大赛（CCF Big Data & Computing Intelligence Contest，简称CCF BDCI）子赛题：个贷违约预测

2. 参赛队伍

队伍名称： 摸着石头划水队
队伍成员： PB20111674 程千里　　PB20111699 吴骏东

3.问题描述

为进一步促进金融普惠的推广落地，金融机构需要服务许多新的客群。银行作为对风险控制要求很高的行业，因为缺乏对新客群的了解，对新的细分客群的风控处理往往成为金融普惠的重要阻碍。如何利用银行现有信贷行为数据来服务新场景、新客群成了一个很有价值的研究方向，迁移学习是其中一个重要手段。本赛题要求利用已有的与目标客群稍有差异的另一批信贷数据，辅助目标业务风控模型的创建，两者数据集之间存在大量相同的字段和极少的共同用户。此处希望大家可以利用迁移学习捕捉不同业务中用户基本信息与违约行为之间的关联，帮助实现对新业务的用户违约预测。

4.模型构建

4.1 业务分析

预测客户是否会违约应该考虑客户的违约收益和违约成本，违约收益大于违约成本就会导致客户违约。

假如一个人借了10000元，那么如果不还钱，他就获得了10000元，这就是他的违约收益。现在，如果不还钱就会被计入征信系统，下次再贷款时银行就不会通过，以后他就不会再获得金融信誉的好处，这就是违约成本之一。

对于违约收益是很好估计的，就是贷款数额。而违约成本是较难评估的，这需要我们构造模型来评估。

导致违约有两种情况，第一种就是违约收益高于成本导致还款意愿不足，第二种就是经济条件恶化导致客户还款能力不足。如何量化还款能力？主要考虑消费水平、收入水平和生活稳定程度，生活稳定的人违约成本更高。

4.2 数据预处理

4.2.1 原始数据概况

原始数据包括如下内容：

训练数据

train_public.csv 个人贷款违约记录数据

train_internet_public.csv 某网络信用贷产品违约记录数据

测试数据

test_public.csv 用于测试的数据

其中，train_public.csv 包含了10000条记录，每条记录包括39项不同特征；train_internet.csv包含了750000条记录，每条记录包括42项不同特征；test_public.csv包含5000条记录，每条记录包括39项不同特征。特征数据类型包含浮点数、整型数、字符串。

4.2.2 库与数据载入及基本设置

```
import matplotlib.pyplot as plt
import seaborn as sns
import gc
import re
import pandas as pd
import lightgbm as lgb
import numpy as np
from sklearn.metrics import roc_auc_score, precision_recall_curve, roc_curve,
average_precision_score
from sklearn.model_selection import KFold
from lightgbm import LGBMClassifier
import seaborn as sns
import gc
from sklearn.model_selection import StratifiedKFold
from dateutil.relativedelta import relativedelta
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

pd.set_option('max_columns', None)#便于查看所有的属性
pd.set_option('max_rows', 200)

train_public = pd.read_csv(r"D:\Data\train_public.csv")
test_public = pd.read_csv(r"D:\Data\test_public.csv")
train_inte = pd.read_csv(r"D:\Data\train_internet.csv")
```

4.2.3 特征数值化

I. 特征"work_year"

执行:

```
print(train_public['work_year'].value_counts())
```

得到:

```
10+ years    3370
2 years       848
3 years       776
< 1 year      765
1 year        671
5 years       623
4 years       562
6 years       476
8 years       458
7 years       436
9 years       393
Name: work_year, dtype: int64
```

可见 work_year 特征的数据为字符串。我们对数据进行数值化。操作如下:

```
def WorkYearTransform(x):
    if str(x)=='nan':
        return -1
        #顺带将缺失值处理了
    x = x.replace('< 1','0')
    return int(re.search('(\d+)', x).group())
        #search扫描整个字符串，返回第一个匹配成功的。group是获取字符串整体。

train_public['work_year'] = train_public['work_year'].map(WorkYearTransform)
test_public['work_year'] = test_public['work_year'].map(WorkYearTransform)
train_inte['work_year'] = train_inte['work_year'].map(WorkYearTransform)
```

II. 特征"industry"、“employer_type”

industry 特征描述的是不同用户的工作行业类别,employer_type表示就职企业的种类。二者可以从一定程度上反应客户的社会背景与经济情况，与最终违约判断有这一定的相关性，故我们需要对其处理。该特征下的内容均为字符串。操作如下：

```
IndustryTransform = {
    '金融业':1,
    '公共服务、社会组织':2,
    '文化和体育业':3,
    '信息传输、软件和信息技术服务业':4,
    '制造业':5,
    '住宿和餐饮业':6,
    '建筑业':7,
    '电力、热力生产供应业':8,
    '房地产业':9,
    '交通运输、仓储和邮政业':10,
    '批发和零售业':11,
    '农、林、牧、渔业':12,
    '采矿业':13,
    '国际组织':14,
}
train_public['industry'] = train_public['industry'].map(IndustryTransform)
test_public['industry'] = test_public['industry'].map(IndustryTransform)
train_inte['industry'] = train_inte['industry'].map(IndustryTransform)

EmployerTransform = {
    '政府机构':1,
    '世界五百强':2,
    '幼教与中小学校':3,
    '高等教育机构':4,
    '普通企业':5,
    '上市企业':6,
}
train_public['employer_type'] =
train_public['employer_type'].map(EmployerTransform)
test_public['employer_type'] =
test_public['employer_type'].map(EmployerTransform)
train_inte['employer_type'] = train_inte['employer_type'].map(EmployerTransform)
```

III. 特征"issue_date"、“earlies_credit_mon”

特征issue_date描述了不同用户贷款发放的时间，earlies_credit_mon表示借款人最早报告的信用额度开立的月份，二者均为时间字符串变量。其格式为"year/month/date"。我们将其转换成2001-1-1的形式，操作如下：

```
def DateTransform(x):
    a = re.search('(\d+-)', x)
    if a is None:
        return '1-'+x#无年份则默认2001
    return x + '-01'#无日则默认1日

train_public['issue_date'] = pd.to_datetime(train_public['issue_date'])
test_public['issue_date'] = pd.to_datetime(test_public['issue_date'])
train_inte['issue_date'] = pd.to_datetime(train_inte['issue_date'])

train_public['earlies_credit_mon'] =
pd.to_datetime(train_public['earlies_credit_mon'].map(DateTransform))
test_public['earlies_credit_mon'] =
pd.to_datetime(test_public['earlies_credit_mon'].map(DateTransform))
train_inte['earlies_credit_mon'] =
pd.to_datetime(train_inte['earlies_credit_mon'].map(DateTransform))
```

2001-1-1的形式不利于计算、比较大小，并且issue_data内的数据都是1日，每日的经济状况波动较小，因此可以将日期中的年份、月份提取出来成为新特征代替日期。操作如下：

```
train_public['issue_date_month'] = train_public['issue_date'].dt.month
test_public['issue_date_month'] = test_public['issue_date'].dt.month
train_inte['issue_date_month'] = train_inte['issue_date'].dt.month
train_public['issue_date_year'] = train_public['issue_date'].dt.year
test_public['issue_date_year'] = test_public['issue_date'].dt.year
train_inte['issue_date_year'] = train_inte['issue_date'].dt.year

train_public['earlies_credit_month'] =
train_public['earlies_credit_mon'].dt.month
test_public['earlies_credit_month'] = test_public['earlies_credit_mon'].dt.month
train_inte['earlies_credit_month'] = train_inte['earlies_credit_mon'].dt.month
train_public['earlies_credit_year'] = train_public['earlies_credit_mon'].dt.year
test_public['earlies_credit_year'] = test_public['earlies_credit_mon'].dt.year
train_inte['earlies_credit_year'] = train_inte['earlies_credit_mon'].dt.year

col_to_drop = ['issue_date', 'earlies_credit_mon']
train_public = train_public.drop(col_to_drop, axis=1)
test_public = test_public.drop(col_to_drop, axis=1 )
train_inte = train_inte.drop(col_to_drop, axis=1 )
```

IV. 特征"class"、“sub_class”

class与sub_class表示贷款类型，这与利息等等息息相关，是很重要的特征。操作如下：

```
ClassTransform = {
    'A': 1,
    'B': 2,
    'C': 3,
    'D': 4,
    'E': 5,
    'F': 6,
    'G': 7,
}
train_public['class'] = train_public['class'].map(ClassTransform)
test_public['class'] = test_public['class'].map(ClassTransform)
train_inte['class'] = train_inte['class'].map(ClassTransform)
```

train_public与test_public内无sub_class这一特征，由于这一特征与利率、还款压力等很相关，故而决定填充此特征。

采用kmeans聚类的方法。注意到只有class为A的元组的sub_class可能取值A1、A2等，所以在class为A的元组中进行聚类得到A1等。由于train_public、test_public的特征与train_inte有一定的不同，从经验上看二者也有所不同，故而决定单独在train_public、test_public内进行聚类。

```
def Kmeans(data, label):
    mms = MinMaxScaler()#归一化
    feats = [f for f in data.columns if f not in ['isDefault']]
    data = data[feats]
    mmsModel = mms.fit_transform(data.loc[data['class'] == label])
    clf = KMeans(5, random_state=1214)
    pre = clf.fit(mmsModel)
    test = pre.labels_
    final = pd.Series(test, index=data.loc[data['class'] == label].index)
    if label == 1:
        final = final.map({0: 'A1', 1: 'A2', 2: 'A3', 3: 'A4', 4: 'A5'})
    elif label == 2:
        final = final.map({0: 'B1', 1: 'B2', 2: 'B3', 3: 'B4', 4: 'B5'})
    elif label == 3:
        final = final.map({0: 'C1', 1: 'C2', 2: 'C3', 3: 'C4', 4: 'C5'})
    elif label == 4:
        final = final.map({0: 'D1', 1: 'D2', 2: 'D3', 3: 'D4', 4: 'D5'})
    elif label == 5:
        final = final.map({0: 'E1', 1: 'E2', 2: 'E3', 3: 'E4', 4: 'E5'})
    elif label == 6:
        final = final.map({0: 'F1', 1: 'F2', 2: 'F3', 3: 'F4', 4: 'F5'})
    elif label == 7:
        final = final.map({0: 'G1', 1: 'G2', 2: 'G3', 3: 'G4', 4: 'G5'})
    return final

train_public1 = Kmeans(train_public, 1)
train_public2 = Kmeans(train_public, 2)
train_public3 = Kmeans(train_public, 3)
train_public4 = Kmeans(train_public, 4)
train_public5 = Kmeans(train_public, 5)
train_public6 = Kmeans(train_public, 6)
train_public7 = Kmeans(train_public, 7)
train_all = pd.concat([train_public1, train_public2, train_public3,
train_public4, train_public5, train_public6,
train_public7]).reset_index(drop=True)
train_public['sub_class'] = train_all
```

```
test_public1 = Kmeans(test_public, 1)
test_public2 = Kmeans(test_public, 2)
test_public3 = Kmeans(test_public, 3)
test_public4 = Kmeans(test_public, 4)
test_public5 = Kmeans(test_public, 5)
test_public6 = Kmeans(test_public, 6)
test_public7 = Kmeans(test_public, 7)
test_all = pd.concat([test_public1, test_public2, test_public3, test_public4,
test_public5, test_public6, test_public7]).reset_index(drop=True)
test_public['sub_class'] = test_all
```

最后对sub_class进行数据化，操作如下：

```
col = ['sub_class']
lbl = LabelEncoder().fit(train_public[col])
train_public[col] = lbl.transform(train_public[col])
test_public[col] = lbl.transform(test_public[col])
train_inte[col] = lbl.transform(train_inte[col])
```

4.2.4 对含缺失项特征进行处理

通过isna()函数可知，train_public、train_inte、test_public在pub_dero_bankrup、f0、f1、f2、f3、f4有缺失，train_inte在debt_loan_ratio、recircle_u、title、post_code、known_outstanding_loan、known_dero、app_type、f5有缺失。

可见在10000条数据中共出现了622项缺失，整体占比不到10%。对于缺失数据的处理，我们通常有如下方式：用平均值填充、用中位数填充、临近值填充、聚类填充等。由于我们已经对该特征进行了数值化处理，且做好了划分，自然我们希望填充的内容也符合我们的划分标准。所以平均值填充不再适用。以下是对不同填充方式的探讨。

I. 平均值填充

```
train_public['pub_dero_bankrup']=train_public['pub_dero_bankrup'].fillna(train_p
ublic['pub_dero_bankrup'].mean())
train_inte['pub_dero_bankrup']=train_inte['pub_dero_bankrup'].fillna(train_inte[
'pub_dero_bankrup'].mean())
test_public['pub_dero_bankrup']=test_public['pub_dero_bankrup'].fillna(test_publ
ic['pub_dero_bankrup'].mean())

train_public['f0']=train_public['f0'].fillna(train_public['f0'].mean())
train_public['f1']=train_public['f1'].fillna(train_public['f1'].mean())
train_public['f2']=train_public['f2'].fillna(train_public['f2'].mean())
train_public['f3']=train_public['f3'].fillna(train_public['f3'].mean())
train_public['f4']=train_public['f4'].fillna(train_public['f4'].mean())

train_inte['f0']=train_inte['f0'].fillna(train_inte['f0'].mean())
train_inte['f1']=train_inte['f1'].fillna(train_inte['f1'].mean())
train_inte['f2']=train_inte['f2'].fillna(train_inte['f2'].mean())
train_inte['f3']=train_inte['f3'].fillna(train_inte['f3'].mean())
train_inte['f4']=train_inte['f4'].fillna(train_inte['f4'].mean())
train_inte['f5']=train_inte['f5'].fillna(train_inte['f5'].mean())

test_public['f0']=test_public['f0'].fillna(test_public['f0'].mean())
test_public['f1']=test_public['f1'].fillna(test_public['f1'].mean())
test_public['f2']=test_public['f2'].fillna(test_public['f2'].mean())
test_public['f3']=test_public['f3'].fillna(test_public['f3'].mean())
test_public['f4']=test_public['f4'].fillna(test_public['f4'].mean())

train_inte['debt_loan_ratio']=train_inte['debt_loan_ratio'].fillna(train_inte['d
ebt_loan_ratio'].mean())
train_inte['recircle_u']=train_inte['recircle_u'].fillna(train_inte['recircle_u'
].mean())
```

II. 临近值填充

train_inte的title、post_code缺失数量非常小，只有

尝试使用临近值填充

```
train_inte['title']=train_inte['title'].fillna(train_inte['title'].mode()[0])
train_inte['post_code'] =
train_inte['post_code'].fillna(train_inte['post_code'].mode()[0])
```

III. 特殊值填充

```
train_inte['known_outstanding_loan']=train['known_outstanding_loan'].fillna(-1)
train_inte['known_dero']=train['known_dero'].fillna(-1)
train_inte['app_type']=train['app_type'].fillna(-1)
    #用-1表示无记录
```

4.2.5 数据集成

将train_public、train_inte合并，由于test_public与train_public的特征种类相同，所以合并时train_inte只保留train_public中也有的特征。操作如下：

```
tr_cols = set(train_public.columns)
same_col = list(tr_cols.intersection(set(train_inte.columns)))
train_inteSame = train_inte[same_col].copy()
train = pd.concat([train_public,train_inteSame]).reset_index(drop=True)
```

4.2.6 数据规约

I. 相关性检测

对于线性相关性强的两个特征，保留其中一个即可，或者将两个特征进行处理得到新的特征。相关系数（部分）如下：

```
corr=train.corr()
corr
```

	loan_id	user_id	total_loan	year_of_loan	interest
loan_id	1.000000	-0.021876	-0.000259	-0.001195	0.000416
user_id	-0.021876	1.000000	0.001460	-0.000630	-0.002189
total_loan	-0.000259	0.001460	1.000000	0.380843	0.142067
year_of_loan	-0.001195	-0.000630	0.380843	1.000000	0.416494
interest	0.000416	-0.002189	0.142067	0.416494	1.000000

其中相关系数大于0.8的如下：

interest, class : 0.952875；

total_loan, monthly_payment : 0.952974；

scoring_low, scoring_high : 0.924619；

f3, f4 : 0.838122；

early_return_amount, early_return_amount_3mon : 0.851380

对这些特征的处理如下：

```
train['total_payment']=train['year_of_loan']*12*train['monthly_payment']
#保留total_loan可以反应违约收益，total_payment直接反应了客户的总还款额。
del train['year_of_loan']
del train['monthly_payment']
del train['interest']
train['scoring']=(train['scoring_high']+train['scoring_low'])/2
#scoring_high与scoring_low的平均值，反应了客户的贷款评分的水平。
del train['scoring_high']
del train['scoring_low']
del train['f4']
del train['early_return_amount']
#保留early_return_amount_3mon,因为近三个月的情况更有时效性。
```

II. 其他

```
del train['loan_id']
del train['user_id']
del train['policy_code']#数据中此特征的值只有1
```

III. 测试集的对应处理

到此，训练集train的数据预处理算是完成了，下面是对测试集的对应处理

```
test_public['total_payment']=test_public['year_of_loan']*12*test_public['monthly
_payment']
del test_public['year_of_loan']
del test_public['monthly_payment']
del test_public['interest']
test_public['scoring']=
(test_public['scoring_high']+test_public['scoring_low'])/2
del test_public['scoring_high']
del test_public['scoring_low']
del test_public['f4']
del test_public['early_return_amount']
del test_public['policy_code']
test = test_public.drop(['loan_id','user_id'],axis=1,inplace=False)
```

4.3 决策树构建

4.3.1 概述

本实验中我们选择**sklearn库**进行决策树构建。

4.3.2 决策树构建

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier, export_graphviz
df_x=train.drop('isDefault',axis=1)
df_y=train.loc[:,'isDefault']
model=DecisionTreeClassifier(
    #默认基尼系数
    random_state=2021
)
dtc=model.fit(df_x,df_y)
```

4.4 结果分析

4.4.1 模型准确度评估

使用交叉验证，我们将训练集分成10折，用来检验原始模型构建的正确性。操作如下：

```
from sklearn.model_selection import cross_val_score
clf=DecisionTreeClassifier(criterion="gini")
cross_val_score(clf, df_x, df_y, cv=10, scoring='accuracy')
```

执行结果为：

```
array([0.73744737, 0.73219737, 0.72953947, 0.73185526, 0.72815789,
       0.73101316, 0.72956579, 0.72977632, 0.72953947, 0.72988158])
均值为0.730897368
```

可以看到构建的模型预测准确度约为0.731，还有很大的提升空间。

4.4.2 回归指标评估

先从df_x,df_y中分一部分出来作为测试集。操作如下：

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(df_x,df_y,test_size=0.3)
clf2=clf.fit(x_train,y_train)
y_pre = clf2.predict(x_test)
```

对测试集的真实结果y_test和预测结果y_pre进行回归统计分析，

```
from sklearn import  metrics
mse = metrics.mean_squared_error(y_test, y_pre)
print("MSE: %.4f" % mse)

mae = metrics.mean_absolute_error(y_test, y_pre)
print("MAE: %.4f" % mae)

ascore = metrics.accuracy_score(y_test, y_pre)
print("ACCURACY SCORE: %.4f" %ascore)
```

上述内容中MSE代表均方差(Mean squared error)，表示预测数据和原始数据对应点误差的平方和的均值；MAE表示平均绝对误差(Mean absolute error)，表示预测数据和原始数对应点误差绝对值的均值；Accuracy Score表示预测准确值，其数值与score接近，表示预测数据与真实数据之间的契合程度。相关结果如下：

```
MSE: 0.2682
MAE: 0.2682
ACCURACY SCORE: 0.7318
```

可见模型的整体准确率还是可以接受。

4.4.3 ROC曲线评估

受试者工作特征曲线(Receiver Operating Characteristic Curve, ROC)，该曲线的横坐标为假阳性率（False Positive Rate, FPR），N是真实负样本的个数，FP是N个负样本中被分类器预测为正样本的个数。纵坐标为真阳性率（True Positive Rate, TPR）。

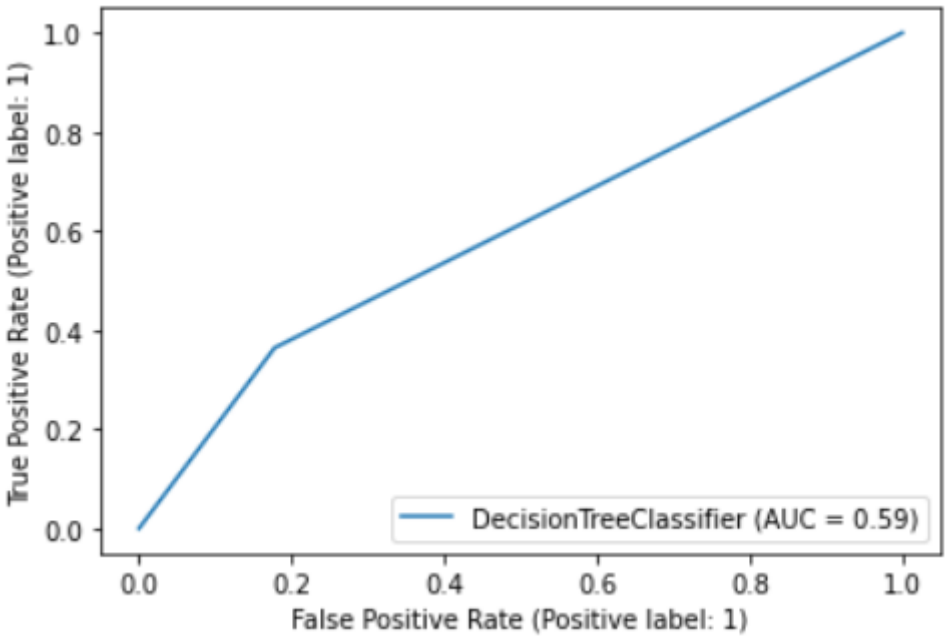
Area Under roc Curve，AUC，表示处于 ROC 曲线下方的图形面积大小。通常 AUC 的值介于0.5到1.0之间，较大的AUC代表了较好的性能。AUC是基于ROC曲线的一种用来度量分类模型好坏的一个标准。

一般情况下，ROC曲线都处于(0, 0)和(1, 1)连线的上方，表示模型实际预测效果优于均值预测。对应的AUC值介于0.5~1之间。

对于本实验，我们进行如下操作：

```
display = metrics.plot_roc_curve(clf2, x_test, y_test)
print('type(display):',type(display))
plt.show()
```

结果如下：



可以看到实验得到的AUC值为0.59，表明模型精度的确是有待提升。

4.5 小结

至此我们已经通过原始数据构建了最初版本的决策树。通过评估该决策树在测试集上的训练效果，我们发现其准确性基本达标，但是AUC值偏低。需要进行调参。

5.调整参数

5.1 调参概述

上面的决策树构建中我们没有进行调参，均选取默认值进行构建。接下来我们选取若干参数对其进行调整，分别评估其对于决策树预测效果的影响。下面使用GridSearchCV方法调参。

5.2 决策树深度

操作如下：

```
from sklearn.model_selection import GridSearchCV
tree_params={'max_depth':range(5,10)}
locally_best_tree=
GridSearchCV(DecisionTreeClassifier(random_state=2021),tree_params,cv=5)
#5折
locally_best_tree.fit(df_x,df_y)
print(format(locally_best_tree.best_params_))
```

结果如下：

```
{'max_depth': 9}。
```

5.3 决策树叶子结点中元素数目

通过增加叶子结点中最少的元素数目，我们可以很好地防止过拟合现象的发生。**在5.2的基础上**我们进行如下测试：

```
tree_params={'min_samples_leaf':range(10,100)}
locally_best_tree=
GridSearchCV(DecisionTreeClassifier(random_state=2021),tree_params,cv=5)
locally_best_tree.fit(df_x,df_y)
print(format(locally_best_tree.best_params_))
```

结果如下：

```
{'min_samples_leaf': 99}
```

5.5 小结

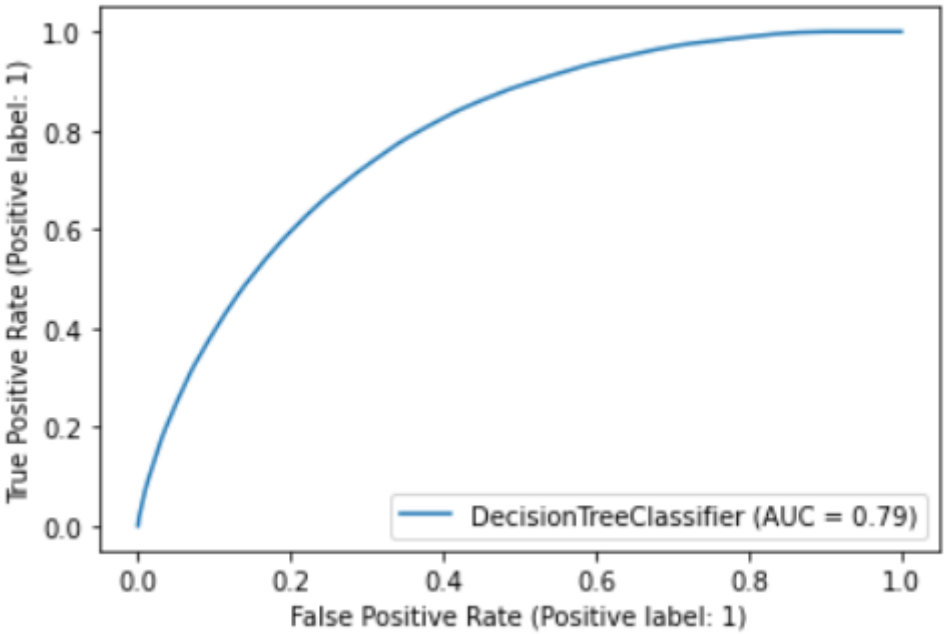
在简单的参数调整后，我们最终得到的决策树参数为：

```
criterion='gini',  max_depth = 8,
min_samples_leaf= 99
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import  metrics
df_x=train.drop('isDefault',axis=1)
df_y=train.loc[:,'isDefault']
model=DecisionTreeClassifier(#默认基尼系数
    random_state=2021,
    max_depth=9,
    min_sample_leaf=99,
    #min_impurity_decrease=0.001
)
dtc=model.fit(df_x,df_y)
display = metrics.plot_roc_curve(dtc, df_x, df_y)
print('type(display):',type(display))
plt.show()
```

评估结果为：

```
MSE: 0.1893
MAE: 0.1893
ACCURACY SCORE: 0.8107
```



AUC=0.79,终于达到了可以接受的程度。

再次通过交叉验证评估模型：

```
cross_val_score(model, df_x, df_y, cv=10, scoring='accuracy')
cross_val_score(model, df_x, df_y, cv=10, scoring='roc_auc')
```

结果如下：

```
array([0.81056579, 0.81057895, 0.80948684, 0.81090789, 0.80927632,
       0.81117105, 0.81043421, 0.80894737, 0.81026316, 0.80946053])
array([0.77847115, 0.78382971, 0.78264   , 0.78820023, 0.78448676,
       0.78645478, 0.78784762, 0.77926009, 0.78255691, 0.78528236])
```

可以认为最终模型的效果基本达到了预期要求。

6.团队分工

代码框架构建： 程千里、吴骏东
数据预处理： 程千里、吴骏东
决策树构建： 吴骏东
模型优化： 程千里
实验报告撰写： 吴骏东 、程千里

7.总结与感悟

这是我们第一次完整地独立参与数据分析的项目。在选课之前，我们还是对数据科学、统计学原理、python编程了解尚浅的“萌新”。无论是知识体系还是内容拓展，这门选修课都毫不逊色与我们的核心专业课。当然，知识内容的极大丰富也造成了我们对其难以全盘消化吸收。而这一次的课程实验很好地帮助我们巩固并强化了已经学过的知识。

在实验的过程之中，我们遇到了很多困难，包括：python环境配置、库调用不兼容问题；数据格式难以处理、频繁报错问题；相关前置知识不够问题；代码整体框架问题等。这极大地提高了我们应对这些问题的能力。

做实验的最大感悟就是要从业务理解出发，不然就是胡乱进行数据分析。应当先通过经验分析特征，再用数值化的方法量化特征，量化目标，这样才能抓住问题核心，应对海量数据的干扰。

最后，由于时间较短，我们没能将这些问题深入研究下去。期待后续的研究交流，也欢迎大家的批评指正。