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 问题分析

本问题要求我们根据已有的用户信贷数据与其违约情况，对另外一批用户进行违约预测。这实际上是一个机器学习创建模型的过程。上述问题的一个简单且直接的解决思路是: **利用已有用户数据构建决策树->根据模拟结果对决策树进行修正与优化->得出可行决策算法。**本实验中我们也将按照如上的步骤进行操作。

4.2 数据预处理

4.2.1 原始数据概况

赛题所附的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，获取榜单排名

其中，train_public.csv 包含了10000条记录(以loan_id为唯一区分特征)，每条记录包括39项不同特征；train_internet.csv包含了750000条记录(以loan_id为唯一区分特征)，每条记录包括42项不同特征。上述数据中特征'work_year'均有空缺，特征内容均包含数字、中文、英文部分。

注：以下处理过程均以train_public.csv中数据为例，train_internet.csv中数据处理方法同理。

4.2.2 对低相关性特征进行处理

为了更好地构建决策树，我们先对所有数据进行相关度分析，从中找出对结果相关度低的变量先行删除。操作如下：

```
temp = train.corr()
temp.to_excel('temp.xlsx', sheet_name='tb1', float_format='%.2f', na_rep='我是空值')
```

在得到的表格中，我们标记出与"isDefault"项相关度绝对值不超过0.01的项，并在程序中对其进行删除。之后我们再进行余下的操作。事实上经过验证删除前后决策树的预测效果相差不大，但删除部分特征后的决策树显然更加简洁。所以我们的删除操作是合理的。

4.2.3 对不规则格式项特征进行处理

I. 特征"work_year"处理

执行：

```
print(train['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 特征为离散型字符串特征，存在着若干缺失值。我们对已有的数据进行数值化。操作如下：

```
def work_year_trans(work_year):      #处理工作时间
    work_year = str(work_year)
    if work_year == '< 1 year' or work_year == '1 year':
        return 1
    elif work_year[0] >= '2' and work_year[0] < '5':
        return 2
    elif work_year[0] >= '5' and work_year[0] <= '9':
        return 3
    elif work_year == '10+ years':
        return 4

train['work_year'] = train['work_year'].apply(work_year_trans)
```

上述过程除了对字符串数值化外，我们还对其进行了归类整合。基本上以25%的数据为一组进行组划分。这样操作的目的是为了消除过拟合，同时降低最终决策树的复杂度。

```
工作时间        编号
  <=1           1
 >1 <=4         2
 >4 <=9         3
  >=10          4
```

划分后的数据描述如下：

```
4.0    3370
3.0    2386
2.0    2186
1.0    1436
Name: work_year, dtype: int64
```

之后可以很方便地对该特征进行数值补充。

II. 特征"industry"处理

industry 特征描述的是不同用户的工作行业类别。该特征可以从一定程度上反应客户所处的社会位置与其社会背景，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中文字符。对此我们先进行内容统计。执行：

```
print(train['industry'].value_counts())
```

得到:

```
金融业                  1629
电力、热力生产供应业          1248
公共服务、社会组织           1065
住宿和餐饮业                907
信息传输、软件和信息技术服务业      808
文化和体育业                793
建筑业                   704
房地产业                  554
采矿业                   506
交通运输、仓储和邮政业          492
农、林、牧、渔业             466
制造业                   302
批发和零售业                279
国际组织                  247
Name: industry, dtype: int64
```

我们发现上述结果内容较为分散，且短时间内难以得到不同类别间的联系，故我们对该特征先不做整合，直接进行数值化处理。操作如下：

```
def industry_transfer(industry):
    industrylist = ['金融业', '电力、热力生产供应业', '公共服务、社会组织', '住宿和餐
饮业', '信息传输、软件和信息技术服务业',
                    '文化和体育业', '建筑业', '房地产业', '采矿业', '交通运输、仓储和
邮政业', '农、林、牧、渔业', '制造业',
                    '批发和零售业', '国际组织']
    return industrylist.index(industry)
train['industry'] = train['industry'].apply(industry_transfer)
```

III. 特征"issue_date"处理

特征issue_date描述了不同用户贷款发放的时间，为时间字符串变量。其格式为"year/month/date"。我们将其转换成时间浮点数，操作如下：

```
def time_transfer(issue_date):
    issue_date = issue_date + ' 00:00:00'
    a = time.strptime(issue_date,'%Y/%m/%d %H:%M:%S') #转换为时间组对象
    return time.mktime(a)

train['issue_date'] = train['issue_date'].apply(time_transfer)
```

这里我们得到了一个连续的时间变量。事实上我们也可以简单地按照年份-季度对数据进行划分。但是两种操作对结果精度影响不大，故我们采取了第一种方式。

4.2.4 对含缺失项特征进行处理

train_public.csv文件中特征"work_year"有数据缺失，以下是处理过程。

执行：

```
print(train.isnull().any()) #查找缺失值
```

得到：

```
total_loan                 False
...
work_year                   True
...
isDefault                  False
```

(略去了部分无关中间输出)

我们对其做一个简单的统计。执行：

```
print(train['work_year'].describe())
```

得到：

```
count    9378.000000
mean        2.820004
std         1.082901
min         1.000000
25%         2.000000
50%         3.000000
75%         4.000000
max         4.000000
Name: work_year, dtype: float64
```

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. 中位数填充

执行：

```
train['work_year'].fillna(train['work_year'].median(), inplace=True)
    #用中位数处理缺失的数据
```

此时的数据统计如下：

```
count    10000.000000
mean         2.831200
std          1.049579
min          1.000000
25%          2.000000
50%          3.000000
75%          4.000000
max          4.000000
Name: work_year, dtype: float64
```

补充前后数据的整体特征未发生改变。后期验证其对于决策结果的重要程度约为0.010。

注：重要程度介于0~1之间，越大代表特征对结果越重要

II. 临近值填充

执行：

```
train['work_year'].fillna(method ='ffill', inplace=True)
    #用前一个未空缺值处理缺失的数据
```

此时的数据统计如下：

```
count    10000.00000
mean         2.82070
std          1.08585
min          1.00000
25%          2.00000
50%          3.00000
75%          4.00000
max          4.00000
Name: work_year, dtype: float64
```

补充前后数据的整体特征未发生改变。后期验证其对于决策结果的重要程度约为0.009。

4.2.5 其他数据预处理

I. 信用分的引入

原始数据中存在"scoring_high"、"scoring_low"特征，用来表示用户的信用分上下界。我们可以将其取平均值作为用户的平均信用分进行统计。操作如下：

```
def score_numjudge(scoring_high, scoring_low):
    #利用上下界计算平均信用分
    return 1/2 *(scoring_high + scoring_low)
train['ave_credits'] = train.apply(lambda x: score_numjudge(x['scoring_high'],
x['scoring_low']), axis= 1)
train = train.drop(['scoring_low', 'scoring_high'], axis = 1)
#删除原始的两个特征
```

II. 全空项填充

最后，我们的原始数据应确保没有空缺项，故执行如下操作：

```
train = train.fillna(0)     #处理其他缺失值
```

将所有的空缺项用常数值0填充。事实上这是我们数据预处理的最后一步。

4.3 决策树构建

4.3.1 概述

在处理完成原始数据后，我们得到了一个没有空缺值、所有内容均为浮点数的数据表。本实验中我们选择**sklearn库**进行决策树构建，并调用**pydotplus库**进行决策树可视化。

4.3.2 决策树构建

对预处理后的数据表预览如下：

```
print(train.head())
```

得到：

```
    interest  ave_credits  ...  early_return_amount_3mon  isDefault
0    11.466   602.727273                       0.0          0
1    16.841   804.375000                       0.0          0
2     8.900   839.090909                       0.0          0
3     4.788   812.500000                       0.0          0
4    12.790   659.848485                       0.0          0
[5 rows x 31 columns]
```

该数据集的[0:29]列为特征列表，[30]列为最终结果。故作划分如下：

```
from sklearn.tree import DecisionTreeClassifier as DTC, export_graphviz
X = train.iloc[:, 0:29]
y = train.iloc[:, 30]
dtc = DTC(criterion='gini')  # 基于基尼系数
dtc.fit(X, y)
```

我们选择以基尼系数作为决策标准构建决策树。事实上以信息熵为标准构建的决策树结果与基尼系数差异不大，我们不对二者在细节上的优劣程度加以区分。调用库对决策树可视化，操作如下：

```
dot_data = tree.export_graphviz(dtc, out_file=None,
                                feature_names=X.columns)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf('iris.pdf')
```

注：决策树可视化文件作为附件包含在文件夹中
对于原始模型，最终得到的决策树准确性评分均值为0.842

4.4 结果分析

注：下面的分析针对原始模型进行

4.4.1 特征重要性分析

得到决策树后，我们对所用到的特征贡献进行分析。操作如下：

```
print([*zip(X.columns),dtc.feature_importances_])
```

得到：

```
[('interest',), ('ave_credits',), ('loan_peryear',), ('monthly_payment',),
('class',), ('employer_type',), ('industry',), ('work_year',), ('house_exist',),
('issue_date',), ('use',), ('debt_loan_ratio',), ('del_in_18month',),
('known_outstanding_loan',), ('known_dero',), ('pub_dero_bankrup',),
('recircle_b',), ('recircle_u',), ('initial_list_status',), ('app_type',),
('title',), ('policy_code',), ('f0',), ('f1',), ('f2',), ('f3',), ('f4',),
('early_return',), ('early_return_amount',),
array([0.05859793, 0.06547005, 0.04241619, 0.04978228, 0.04492473,
    0.02106649, 0.0357894 , 0.01476012, 0.01037864, 0.0508633 ,
    0.00776959, 0.06131941, 0.00819608, 0.03156619, 0.00809274,
    0.00270712, 0.04204074, 0.04954528, 0.00850437, 0.00064824,
    0.01130115, 0.        , 0.02863468, 0.        , 0.02979758,
    0.04090912, 0.02173222, 0.00116533, 0.25202102])]
```

以上是决策树用到的特征重要性表。其中数值越大代表在决策树构建过程中该特征发挥的作用越大。我们去除重要性低于0.01的特征

```
'use', 'del_in_18month', 'known_dero',
'pub_dero_bankrup', 'initial_list_status', 'app_type'
```

重新构建决策树，结果如下:

```
[('interest',), ('ave_credits',), ('loan_peryear',), ('monthly_payment',),
('class',), ('employer_type',), ('industry',), ('work_year',), ('house_exist',),
('issue_date',), ('debt_loan_ratio',), ('known_outstanding_loan',),
('recircle_b',), ('recircle_u',), ('title',), ('policy_code',), ('f0',), ('f1',),
('f2',), ('f3',), ('f4',), ('early_return',), ('early_return_amount',),
array([0.05799417, 0.05102454, 0.04439337, 0.05781089,
0.04491889,   0.01036387, 0.03665764, 0.02018741, 0.01123159,
0.04589599,   0.06645536, 0.03531837, 0.07294249, 0.04606404,
0.02376239,   0.        , 0.02958151, 0.        , 0.03463284,
0.04332184,   0.03054502, 0.0014262 , 0.23547158])]

score = 0.8330550918196995
```

我们发现模型的预测效果没有发生较大变化。该操作可以较好地优化决策树的大小，使得生成的决策树更加直观、简洁。

4.4.2 模型准确度评估

我们将训练集的10000条数据划分成 [9:1] 的训练-测试集，用来检验原始模型构建的正确性。操作如下:

```
from sklearn.tree import DecisionTreeClassifier as DTC, export_graphviz
X = train.iloc[:, 0:29]
y = train.iloc[:, 30]
dtc = DTC(criterion='gini')  # 基于基尼系数
dtc.fit(X, y)
print(dtc.score(test.iloc[:, 0:29], test.iloc[:, 30]))
```

执行结果均值(10次评分取均值)为:

```
0.845
```

可以看到对于训练集内部数据，本次实验构建的模型预测准确度接近85%。该预测基本达到了目标精度，但仍然有一定的提升空间。

4.4.3 回归指标评估

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，操作如下：

```
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: 0.1686
MAE: 0.1686
ACCURACY SCORE: 0.8314
```

可见模型的整体准确率还是可以的。

4.4.4 ROC曲线评估

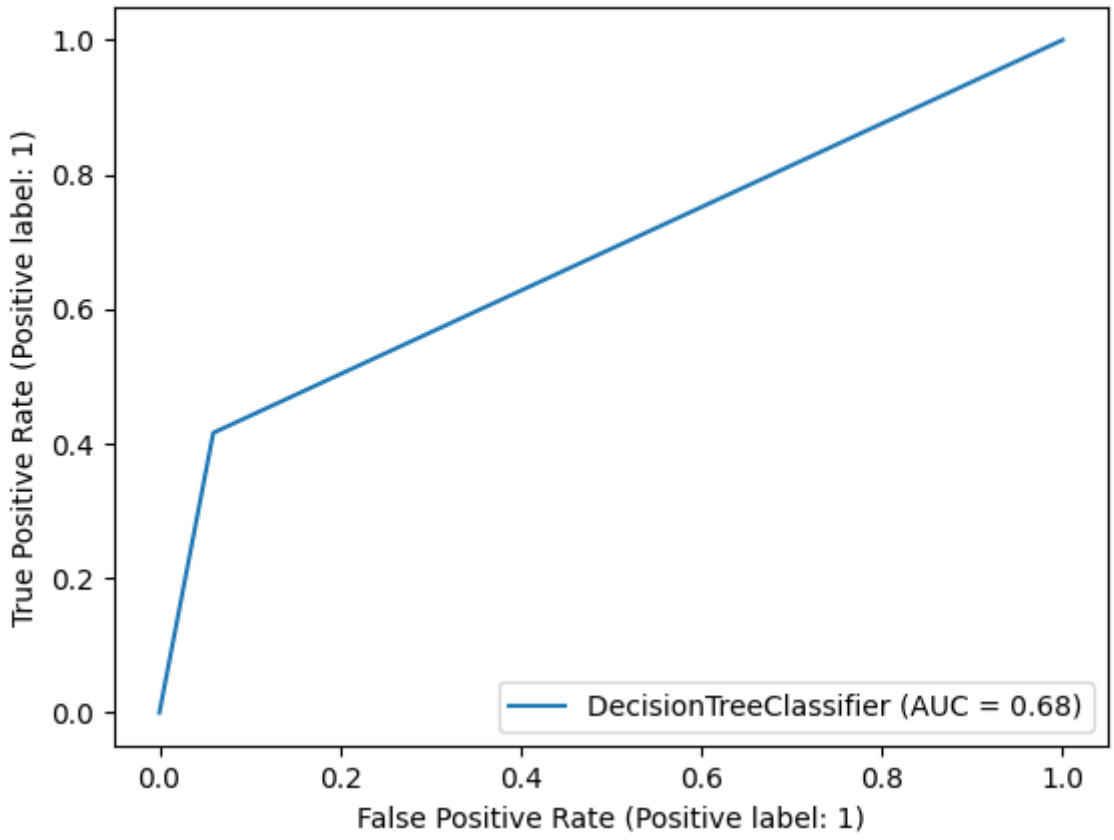
受试者工作特征曲线(Receiver Operating Characteristic Curve, ROC)，该曲线的横坐标为假阳性率（False Positive Rate, FPR），N是真实负样本的个数，FP是N个负样本中被分类器预测为正样本的个数。纵坐标为真阳性率（True Positive Rate, TPR）。

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：

```
import matplotlib.pyplot as plt
display = metrics.plot_roc_curve(dtc, 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0.68，表明模型精度基本达到预期要求。但是相比较其他模型准确度还是偏低，表面模型出现了一定的过拟合现象。

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。造成这种现象的主要原因是决策树划分过于细致，导致决策结果严重依赖训练集的数据特征，也就是所谓的过拟合现象。为此我们需要对其进行更加深入的优化。

5.模型调参

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。这样得到的决策树往往会出现过拟合现象。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。

5.2 限制决策树深度

> **max_depth : int or None, optional (default=None)**
>
> The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
>
> (决策树深度，如果不设置，那么决策树会分裂到直到所有的叶子节点中只含有同一种样本或者叶子中的样本数小于min_samples_split参数)

> ————sklearn官方文档

决策树的深度限制可以很好地防止出现过拟合现象。事实上当决策树深度加深时，我们所需要的样本数据量也对应增加。对于10000条数据，我们可以选取最大深度为5~10进行比较。

max_depth	accuracy_score	AUC
5	0.8564	0.96
6	0.8781	0.95
7	0.8781	0.94
8	0.8798	0.91
9	0.8748	0.86
10	0.8581	0.81

综合准确性与拟合状况考虑，树深最大值为8时的模型效果最好。

5.3 限制决策树叶子结点中元素数目

> **min_samples_leaf : int, float, optional (default=1)**
>
> The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
>
> If int, then consider min_samples_leaf as the minimum number.
>
> If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
>
> (如果是整数，就是每个叶子节点最少容纳的样本数，如果是小数，那么每个叶子节点最少容纳的个数等于min_samples_leaf*样本总数。如果某个分裂条件下分裂出得某个子树含有的样本数小于这个数字，那么不能进行分裂) ————sklearn官方文档

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：

min_samples_leaf	accuracy_score	AUC
2	0.8781	0.91
3	0.8765	0.90
4	0.8815	0.89
5	0.8848	0.89
10	0.8815	0.92
15	0.8881	0.94
20	0.8898	0.94
25	0.8915	0.94
30	0.8781	0.95

综合准确性与拟合状况考虑，元素数目最小值为25时的模型效果最好。

5.4 限制分裂时考虑的特征数目

> max_features : int, float, string or None, optional (default=None)
>
> The number of features to consider when looking for the best split:
>
> If int, then consider max_features features at each split.
>
> If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.
>
> If “auto”, then max_features=sqrt(n_features).>
>
> If “sqrt”, then max_features=sqrt(n_features).
>
> If “log2”, then max_features=log2(n_features).
>
> If None, then max_features=n_features.
>
> Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features.
>
> (分裂时需要考虑的最多的特征数，如果是整数，那么分裂时就考虑这几个特征，如果是小数，则分裂时考虑的特征数=max_features*总特征数，如果是“auto”或者“sqrt”，考虑的特征数是总特征数的平方根，如果是“log2”，考虑的特征数是log2（总特征素），如果是None，考虑的特征数=总特征数。需要注意的是，如果在规定的考虑特征数之内无法找到满足分裂条件的特征，那么决策树会继续寻找特征，直到找到一个满足分裂条件的特征) ————sklearn官方文档

该参数限制分枝时考虑的特征个数，超过限制个数的特征都会被舍弃。但这个方法比较暴力，在不知道决策树中的各个特征的重要性的情况下，强行设定这个参数可能会导致模型学习不足。为此我们先假定所有特征重要性相同(事实上通过前期的相关性调查我们知道部分特征对预测结果贡献较大)，然后进行如下测试：

max_features	accuracy_score	AUC
5	0.8381	0.92
10	0.8848	0.95
15	0.8865	0.95
20	0.8715	0.94
25	0.8648	0.92

综合准确性与拟合状况考虑，特征数目最大值为15时的模型效果最好。

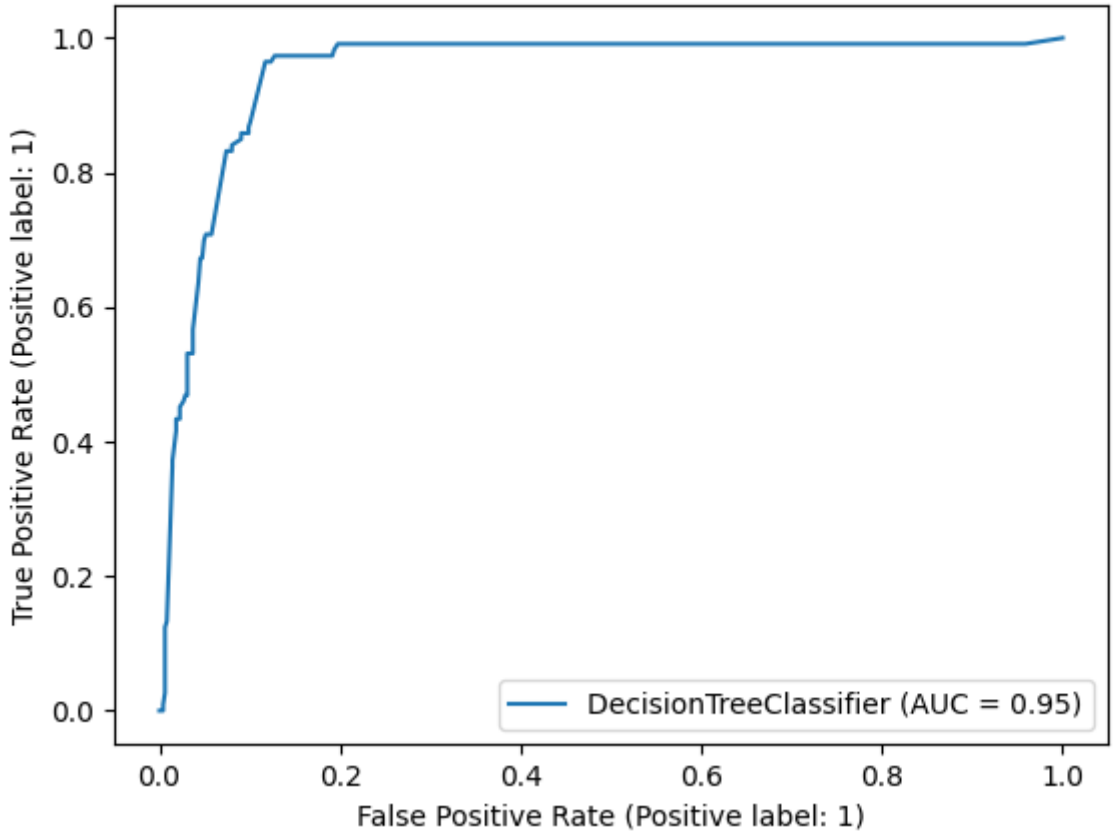
5.5 小结

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= 10,  max_features= 15
```

评估结果为：

```
MSE: 0.1269
MAE: 0.1269
ACCURACY SCORE: 0.8731
```



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

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”，怀揣着对于新知识的渴求开始了本学期最大的一次尝试。不得不说，这门课程的干货数量远远超出了我的预期。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。如果说课堂内容是这门课程的硬核部分，那么课程实验就是衔接理论与实践很好的跳板。

在实验的过程之中，我们也遇到了很多困难，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。事实上这些问题也困扰了我们不少时间，我们也为了一些重要问题的解决方案而彻夜长谈。但现在回视自己走过的路程，内心还是颇为高兴的。先前看似难以逾越的困难如今也被我们克服(或者绕开)，之后对于类似的需求我们也不用手足无措。

回到课程本身，尽管这只是一门公选课，但我们对待它就像是自己的专业课一样。在科大一年半的时间里，这门选修课是我们所经历过的可谓高质量选修课之一。作为计算机专业的学生，我的未来方向也会偏向计算机应用部分。这门课程也将作为我新阶段学习规划的起点。

最后，由于时间较短，我们没有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。