**使用决策树预测德国人信贷风险**

**1. 实验目的**

* 熟练掌握使用决策树构建分类模型
* 熟练掌握决策树模型的调参过程

**2. 实验内容**

银行在给客户贷款之前，会提前预测贷款者信用风险，以决定是否贷款给他们。

本实验基于德国信贷风险数据，使用决策树模型预测信贷风险。预测结果为有风险/无风险。

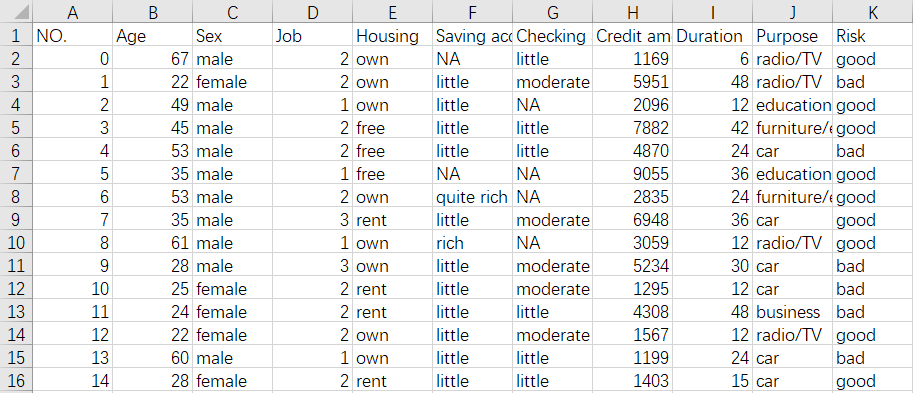
**3. 实验数据**

**数据集位置**

数据集存放在/home/dataset目录下。数据集名字是 german\_credit\_data.csv

**数据集格式**

德国人信贷风险数据集包含 1000 条德国人贷款数据。



**数据字段解释**

* NO.：数据序号
* Age：数值类型；年龄
* Sex：字符串类型；性别；有2个值，分别为male（男性）和female（女性）
* Job：数值类型；工作状态；有4个值，分别为0、1、2、3；0：unskilled and non-resident（无特长且非本地居民），1：unskilled and + + resident（无特长且是本地居民），2：skilled（有特长），3：highly skilled（技艺高超）
* Housing：字符串类型；有3个值，分别为own（有房子），rent（租房子），free（自己没房子也不租房，比如政府的救济房、比如住在别人家里）
* Saving accounts：字符串类型；活期储蓄账户存款多少（有固定利率）；有4个值，分别是little（几乎没有），moderate（中等），rich（富有），quite rich（非常富有）；如果有缺失数据，很大可能是这个人没有银行账户
* Checking account：字符串类型；普通支票账户存款多少（一般无息）；有3个值，分别是little（几乎没有），moderate（中等），rich（富有）；如果有缺失数据，很大可能是这个人没有银行账户
* Credit amount：数值类型；信贷额度（单位DM，即马克）
* Duration：数值类型；信贷时间（单位月）
* Purpose：字符串类型；贷款目的；有8个值，分别是car（买车），furniture/equipment（家具/装备），radio/TV（收音机/电视），+ + + + domestic appliances（家用电器），repairs（维修），education（教育），business（做生意），vacation/others（度假/其它）
* Risk：字符串类型；信用风险等级；有2个值，分别为good（无风险），bad（有风险）

**4. 实验知识点**

* 决策树模型做分类
* 树形结构

**5. 实验时长**

1学时

**6. 实验环境**

* Linux Ubuntu 操作系统
* Jupyter 代码编辑器
* Python 3.6.9
* numpy 1.18.5
* pandas 1.1.5
* scikit-learn 0.24.2

**7. 实验分析**

1. 导入包、导入数据、做数据预处理
2. 构建决策树模型
3. 使用不同参数，比较决策树模型性能
4. 画出决策树模型的树形结构

**8. 实验过程**

**8.1 导入包**

In [1]:

*# 导入包*

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**8.2 导入数据集**

In [2]:

*# 导入数据集*

data = pd.read\_csv("/home/dataset/german\_credit\_data.csv")

data

Out[2]:

|  | **NO.** | **Age** | **Sex** | **Job** | **Housing** | **Saving accounts** | **Checking account** | **Credit amount** | **Duration** | **Purpose** | **Risk** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 67 | male | 2 | own | NaN | little | 1169 | 6 | radio/TV | good |
| **1** | 1 | 22 | female | 2 | own | little | moderate | 5951 | 48 | radio/TV | bad |
| **2** | 2 | 49 | male | 1 | own | little | NaN | 2096 | 12 | education | good |
| **3** | 3 | 45 | male | 2 | free | little | little | 7882 | 42 | furniture/equipment | good |
| **4** | 4 | 53 | male | 2 | free | little | little | 4870 | 24 | car | bad |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 995 | 31 | female | 1 | own | little | NaN | 1736 | 12 | furniture/equipment | good |
| **996** | 996 | 40 | male | 3 | own | little | little | 3857 | 30 | car | good |
| **997** | 997 | 38 | male | 2 | own | little | NaN | 804 | 12 | radio/TV | good |
| **998** | 998 | 23 | male | 2 | free | little | little | 1845 | 45 | radio/TV | bad |
| **999** | 999 | 27 | male | 2 | own | moderate | moderate | 4576 | 45 | car | good |

1000 rows × 11 columns

去掉NO.列，因为该列无用处。

In [3]:

data = data.drop(['NO.'], axis = 1)

data

Out[3]:

|  | **Age** | **Sex** | **Job** | **Housing** | **Saving accounts** | **Checking account** | **Credit amount** | **Duration** | **Purpose** | **Risk** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 67 | male | 2 | own | NaN | little | 1169 | 6 | radio/TV | good |
| **1** | 22 | female | 2 | own | little | moderate | 5951 | 48 | radio/TV | bad |
| **2** | 49 | male | 1 | own | little | NaN | 2096 | 12 | education | good |
| **3** | 45 | male | 2 | free | little | little | 7882 | 42 | furniture/equipment | good |
| **4** | 53 | male | 2 | free | little | little | 4870 | 24 | car | bad |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 31 | female | 1 | own | little | NaN | 1736 | 12 | furniture/equipment | good |
| **996** | 40 | male | 3 | own | little | little | 3857 | 30 | car | good |
| **997** | 38 | male | 2 | own | little | NaN | 804 | 12 | radio/TV | good |
| **998** | 23 | male | 2 | free | little | little | 1845 | 45 | radio/TV | bad |
| **999** | 27 | male | 2 | own | moderate | moderate | 4576 | 45 | car | good |

1000 rows × 10 columns

**8.3 数据预处理**

**8.3.1 检测并处理缺失值**

In [4]:

*# 检测缺失值*

null\_df = data.isnull().sum() *# 检测缺失值*

null\_df

Out[4]:

Age 0

Sex 0

Job 0

Housing 0

Saving accounts 183

Checking account 394

Credit amount 0

Duration 0

Purpose 0

Risk 0

dtype: int64

Saving accounts和Checking account 这2个字段有缺失值，需要填补

In [5]:

*# 处理Saving accounts 和 Checking account 这2个字段*

**for** col **in** ['Saving accounts', 'Checking account']: *# 处理缺失值*

data[col].fillna('none', inplace=**True**) *# none说明这些人没有银行账户*

填补完毕后，需要再次检查缺失值，确保缺失值缺失填补了

In [6]:

*# 检测缺失值*

null\_df = data.isnull().sum()

null\_df

Out[6]:

Age 0

Sex 0

Job 0

Housing 0

Saving accounts 0

Checking account 0

Credit amount 0

Duration 0

Purpose 0

Risk 0

dtype: int64

由输出结果得出结论，无字段有缺失值

**8.3.2 处理因变量**

对因变量做字符编码。

In [7]:

**from** **sklearn.preprocessing** **import** LabelEncoder

le = LabelEncoder()

data['Risk'] = le.fit\_transform(data['Risk'])

data['Risk']

Out[7]:

0 1

1 0

2 1

3 1

4 0

..

995 1

996 1

997 1

998 0

999 1

Name: Risk, Length: 1000, dtype: int64

In [8]:

le.classes\_

Out[8]:

array(['bad', 'good'], dtype=object)

bad被转换为了0，good被转换为了1。这和Risk字段的含义不符合。应该把bad转换为1，把good转换为0。

In [9]:

data['Risk'] = data['Risk'] - 1

data.loc[data['Risk']==-1, 'Risk'] = 1

data['Risk']

Out[9]:

0 0

1 1

2 0

3 0

4 1

..

995 0

996 0

997 0

998 1

999 0

Name: Risk, Length: 1000, dtype: int64

此时bad被转换为1，good被转换为0。符合要求。

**8.3.3 处理类别型变量**

In [10]:

*# 查看所有字段的类型*

print(data.dtypes)

Age int64

Sex object

Job int64

Housing object

Saving accounts object

Checking account object

Credit amount int64

Duration int64

Purpose object

Risk int64

dtype: object

结合输出的字段类型结果和字段解释，只有Job字段的类型有误。Job字段的类型应该是类别型，在数据集中却是整型。

In [11]:

*# 处理Job字段的类型*

data['Job'] = data['Job'].astype('object')

字段类型无误后，处理类别型变量。

In [12]:

*# 处理类别型变量*

data = pd.get\_dummies(data, drop\_first = **True**)

data

Out[12]:

|  | **Age** | **Credit amount** | **Duration** | **Risk** | **Sex\_male** | **Job\_1** | **Job\_2** | **Job\_3** | **Housing\_own** | **Housing\_rent** | **...** | **Checking account\_moderate** | **Checking account\_none** | **Checking account\_rich** | **Purpose\_car** | **Purpose\_domestic appliances** | **Purpose\_education** | **Purpose\_furniture/equipment** | **Purpose\_radio/TV** | **Purpose\_repairs** | **Purpose\_vacation/others** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 67 | 1169 | 6 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **1** | 22 | 5951 | 48 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **2** | 49 | 2096 | 12 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **3** | 45 | 7882 | 42 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **4** | 53 | 4870 | 24 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 31 | 1736 | 12 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **996** | 40 | 3857 | 30 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **997** | 38 | 804 | 12 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **998** | 23 | 1845 | 45 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **999** | 27 | 4576 | 45 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | ... | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

1000 rows × 24 columns

**8.3.4 得到自变量和因变量**

In [13]:

*# 得到自变量和因变量*

y = data['Risk'].values

data = data.drop(['Risk'], axis = 1)

x = data.values

**8.3.5 拆分训练集和测试集**

In [14]:

*# 拆分训练集和测试集*

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2, random\_state = 1)

print(x\_train.shape)

print(x\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

(800, 23)

(200, 23)

(800,)

(200,)

**8.3.6 特征缩放**

In [15]:

*# 特征缩放*

**from** **sklearn.preprocessing** **import** StandardScaler

sc\_x = StandardScaler()

x\_train = sc\_x.fit\_transform(x\_train)

x\_test = sc\_x.transform(x\_test)

In [16]:

x\_train[:5]

Out[16]:

array([[-0.91656217, -0.69772497, 0.09776381, -1.4535613 , -0.4843221 ,

0.75611456, -0.42214126, -1.52752523, 2.0647416 , -0.32634444,

2.081666 , -0.25819889, -0.23241869, -0.61009236, 1.24736143,

-0.25543363, 1.35531525, -0.11250879, -0.22941573, -0.47050403,

-0.62554324, -0.12852402, -0.11250879],

[ 1.27224301, -0.29775188, -0.75420842, 0.68796548, -0.4843221 ,

0.75611456, -0.42214126, 0.65465367, -0.4843221 , -0.32634444,

2.081666 , -0.25819889, -0.23241869, -0.61009236, 1.24736143,

-0.25543363, 1.35531525, -0.11250879, -0.22941573, -0.47050403,

-0.62554324, -0.12852402, -0.11250879],

[-0.65390555, -0.10517224, 0.01256659, -1.4535613 , -0.4843221 ,

-1.3225509 , 2.36887531, 0.65465367, -0.4843221 , 3.06424712,

-0.48038446, -0.25819889, -0.23241869, -0.61009236, -0.80169226,

3.91491132, 1.35531525, -0.11250879, -0.22941573, -0.47050403,

-0.62554324, -0.12852402, -0.11250879],

[-1.09166658, -0.6727944 , -1.26539175, -1.4535613 , -0.4843221 ,

-1.3225509 , -0.42214126, -1.52752523, 2.0647416 , -0.32634444,

-0.48038446, 3.87298335, -0.23241869, -0.61009236, -0.80169226,

-0.25543363, 1.35531525, -0.11250879, -0.22941573, -0.47050403,

-0.62554324, -0.12852402, -0.11250879],

[-1.00411438, -0.00870176, -0.24302508, -1.4535613 , -0.4843221 ,

0.75611456, -0.42214126, 0.65465367, -0.4843221 , -0.32634444,

-0.48038446, -0.25819889, -0.23241869, -0.61009236, -0.80169226,

-0.25543363, -0.73783572, -0.11250879, -0.22941573, -0.47050403,

1.59861051, -0.12852402, -0.11250879]])

In [17]:

y\_train[:5]

Out[17]:

array([0, 0, 0, 0, 1])

**8.4 使用不同的参数构建决策树模型**

**8.4.1 模型1**

参数：criterion = 'entropy', max\_depth = 10, min\_samples\_leaf = 10

**8.4.1.1 构建模型**

In [18]:

*# 模型1：构建决策树模型*

**from** **sklearn.tree** **import** DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy', max\_depth = 10, min\_samples\_leaf = 10, random\_state = 0)

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

Out[18]:

DecisionTreeClassifier(criterion='entropy', max\_depth=10, min\_samples\_leaf=10,

random\_state=0)

**8.4.1.2 测试集做预测**

In [19]:

*# 在测试集做预测*

y\_pred = classifier.predict(x\_test)

**8.4.1.3 评估模型性能**

In [20]:

*# 输出准确率*

**from** **sklearn.metrics** **import** accuracy\_score

print(accuracy\_score(y\_test, y\_pred))

0.73

准确率是0.73，非常低，表明模型性能非常差。

In [21]:

*# 输出混淆矩阵*

**from** **sklearn.metrics** **import** confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

[[117 24]

[ 30 29]]

由混淆矩阵可知，真实值是0（实际没风险），预测值也是0（预测没风险），这样的样本有117条；真实是0，预测是1，这样的样本有24条；真实值是1，预测是0，这样的样本有30条；真实值是1，预测值也是1，这样的样本有29条。

In [22]:

**from** **sklearn.metrics** **import** classification\_report

print(classification\_report(y\_test, y\_pred))

precision recall f1-score support

0 0.80 0.83 0.81 141

1 0.55 0.49 0.52 59

accuracy 0.73 200

macro avg 0.67 0.66 0.67 200

weighted avg 0.72 0.73 0.73 200

1. 根据业务场景，银行想尽可能预测出来有风险的人，以避免贷出去的钱收不回来，所以Recall指标更加重要。
2. Recall指标的weighted avg只有0.73，相对比较低。

**8.4.1.4 画出树形结构**

在Jupyter Notebook中运行会报错，但是.dot文件却成功生成了。所以尽管忽略这个错误。

In [23]:

*# 生成.dot文件*

**from** **sklearn** **import** tree

**import** **graphviz**

graphviz.Source(tree.export\_graphviz(classifier, out\_file='output.dot'))

---------------------------------------------------------------------------

AttributeError Traceback (most recent call last)

~/miniconda3/lib/python3.9/site-packages/IPython/core/formatters.py in \_\_call\_\_(self, obj)

**343** method = get\_real\_method(obj, self.print\_method)

**344** if method is not None:

--> 345 return method()

**346** return None

**347** else:

~/miniconda3/lib/python3.9/site-packages/graphviz/files.py in \_repr\_svg\_(self)

**139**

**140** def \_repr\_svg\_(self):

--> 141 return self.pipe(format='svg').decode(self.\_encoding)

**142**

**143** def pipe(self, format=None, renderer=None, formatter=None, quiet=False):

~/miniconda3/lib/python3.9/site-packages/graphviz/files.py in pipe(self, format, renderer, formatter, quiet)

**162** format = self.\_format

**163**

--> 164 data = self.source.encode(self.\_encoding)

**165**

**166** out = backend.pipe(self.\_engine, format, data,

AttributeError: 'NoneType' object has no attribute 'encode'

Out[23]:

<graphviz.files.Source at 0x7ff519481b50>

将 dot 文件转换成图片文件

In [24]:

!dot -Tpng output.dot -o output.png

会生成output.png文件，保存到当前目录下。

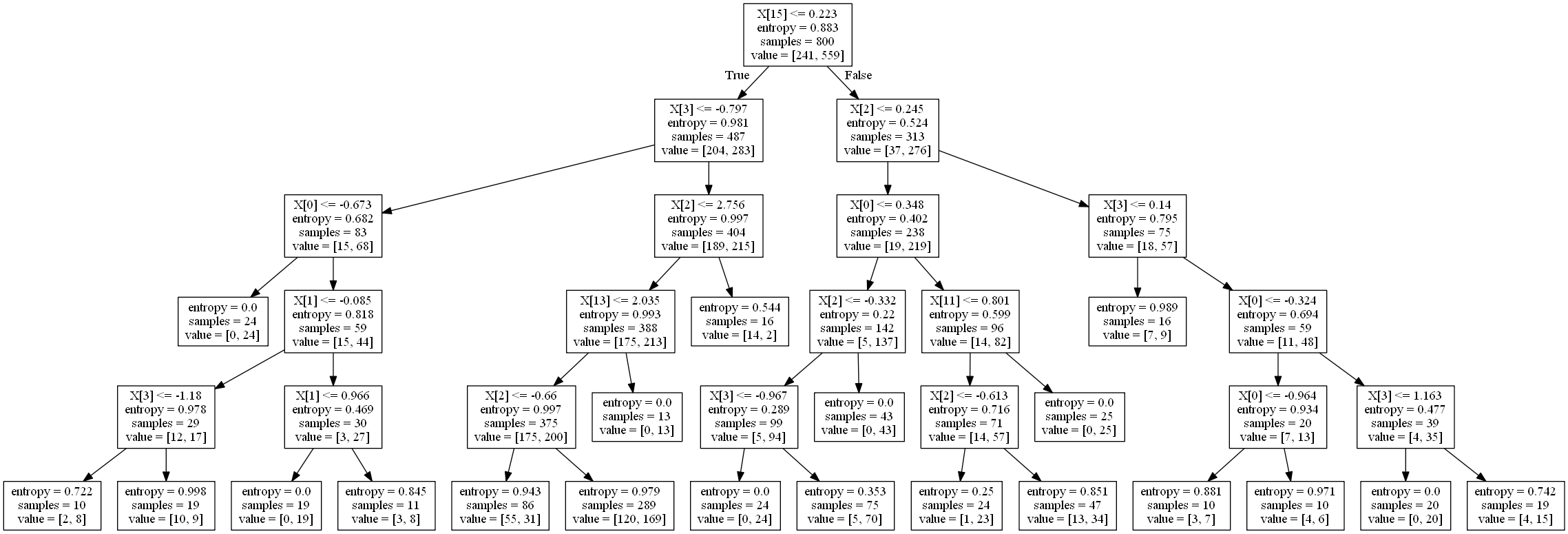
将 dot 文件转换成 pdf 文件

In [25]:

!dot -Tpdf output.dot -o output.pdf

会生成output.pdf文件，保存到当前目录下。

生成的树结构如下图所示：



通过树形结构，可以清晰地看到分类的逻辑。

**8.4.2 模型2**

参数：criterion = 'gini', max\_depth = 5, min\_samples\_leaf = 10, min\_samples\_split=10

In [26]:

*# 模型2：构建决策树模型*

classifier = DecisionTreeClassifier(criterion = 'gini', max\_depth = 5, min\_samples\_leaf = 10, min\_samples\_split=10, random\_state = 0)

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

Out[26]:

DecisionTreeClassifier(max\_depth=5, min\_samples\_leaf=10, min\_samples\_split=10,

random\_state=0)

In [27]:

*# 在测试集做预测*

y\_pred = classifier.predict(x\_test)

In [28]:

*# 评估模型性能*

print(accuracy\_score(y\_test, y\_pred))

0.755

In [29]:

print(confusion\_matrix(y\_test, y\_pred))

[[126 15]

[ 34 25]]

In [30]:

print(classification\_report(y\_test, y\_pred))

precision recall f1-score support

0 0.79 0.89 0.84 141

1 0.62 0.42 0.51 59

accuracy 0.76 200

macro avg 0.71 0.66 0.67 200

weighted avg 0.74 0.76 0.74 200

由准确率、分类报告可见，使用这批参数，模型的性能有很大的提升。

可以尝试着继续调整参数，继续优化模型性能。

**9. 实验结果（结论）**

1. 不同超参数对模型性能的影响不同
2. 通过树形结构，可以清晰地看到分类的逻辑