Decision Tree

1. 实验目的

了解决策树算法的原理，并且可以简单应用

2. 算法原理

1) 介绍

决策树(Decision Tree）是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。由于这种决策分支画成图形很像一棵树的枝干，故称决策树。在机器学习中，决策树是一个预测模型，他代表的是对象属性与对象值之间的一种映射关系。Entropy = 系统的凌乱程度，使用算法 ID3C4.5,和 C5.0 生成树算法使用熵。这一度量是基于信息学理论中熵的概念。

决策树是一种树形结构，其中每个内部节点表示一个属性上的测试，每个分支代表一个测试输出，每个叶节点代表一种类别。

决策树的根节点到叶节点的每一条路径构建一条规则；路径内部节点的特征对应着规则的条件，而叶节点的类对应着规则的结论。决策树学习算法通常是递归的选择最优特征，并根据该特征对训练数据进行分割，使得对各个子集数据有一个最好的分类结果。这一过程对应着特征空间的划分，也对应着决策树的构建。开始，构建根节点，将所有训练数据都放在根节点，选择一个最优特征，按照这一特征将训练数据集分割成子集，使得各个子集有一个当前条件下的最好分类。如果这些子集已经能够被基本正确分类，那么构建叶节点，并将这些子集分到对应的叶节点中去；如果还有子集不能被正确分类，那么就对这些子集继续选择最优特征，继续对其进行分割，构建相应的节点。如此递归下去，直到所有训练数据子集都被基本正确分类或者没有合适的特征为止。最后每个子集都有相应的类，这就生成了一颗决策树。

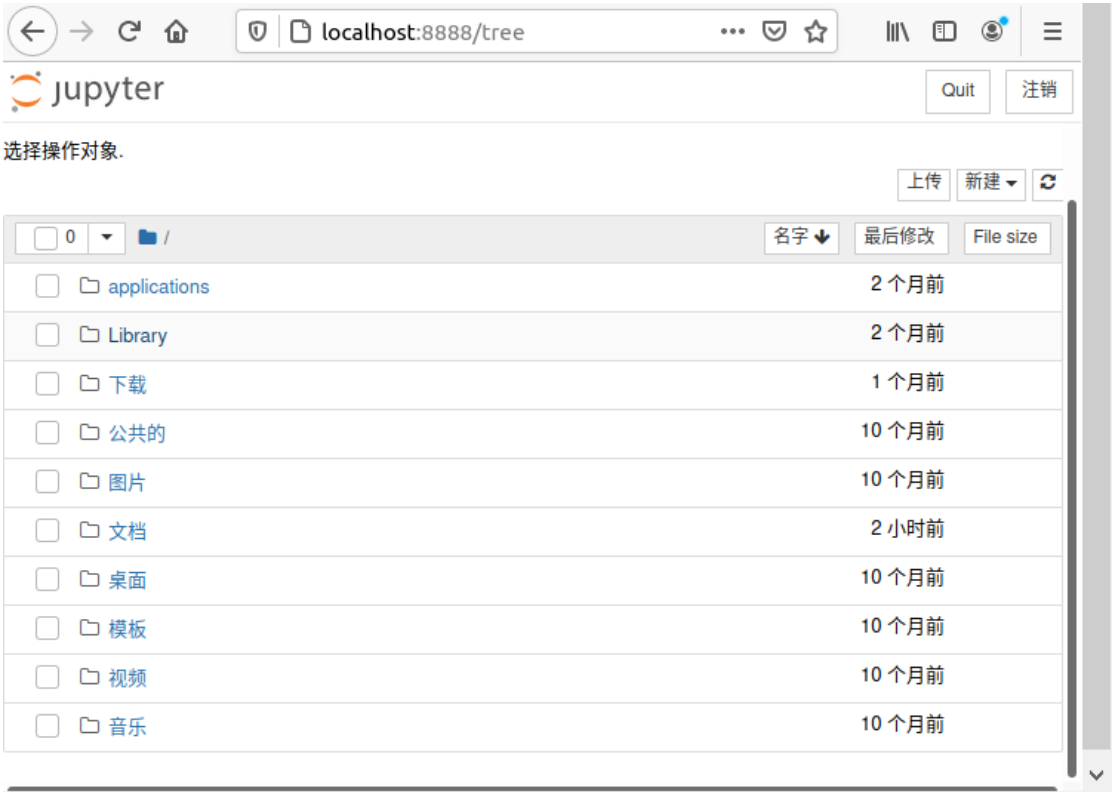
以上方法生成的决策树很有可能发生过拟合，所以我们需要对决策树进行剪枝处理，使决策树变的简单，从而具有更好的泛化能力。

3. 实验环境

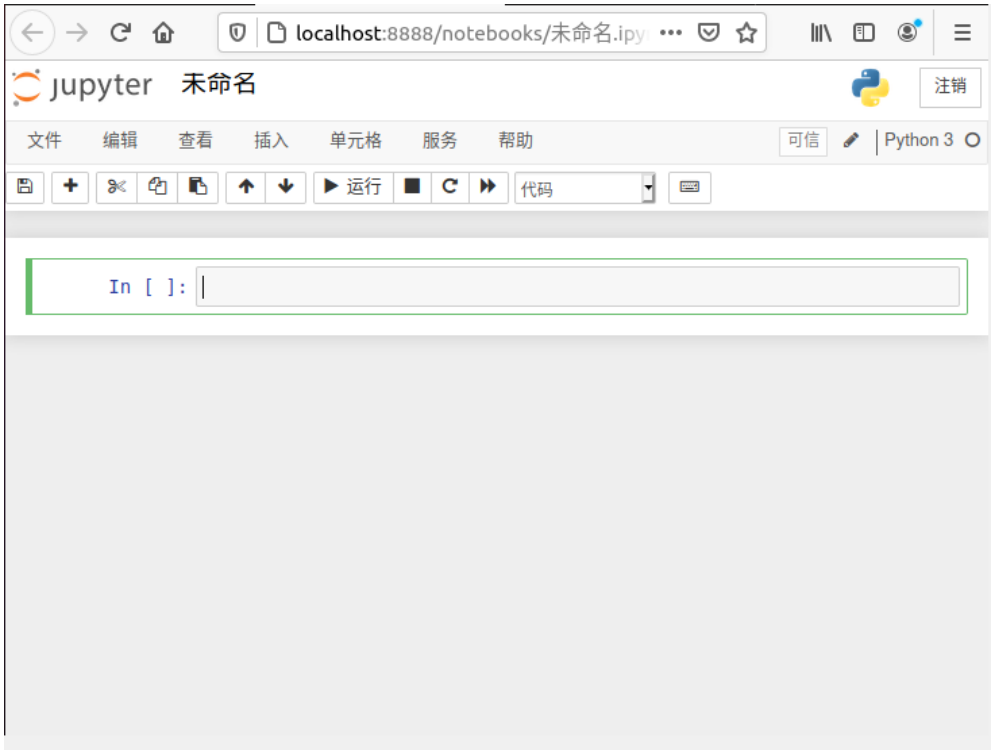
Ubuntu 20.04
Python 3.6
Jupyter notebook

4. 实验步骤

1)打开终端，然后输入 jupyter notebook，出现如下界面



2)选定特定文件夹，新建 ipynb 文件，在未命名出可重命名文件



5. 实操

Step 1:数据预处理

1. 导入库
2. 导入数据集
3. 删除缺失值
4. 分割数据集

```
# 导入工具包 numpy pandas 用于数据处理
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

#读取数据，并删除前 3 列
df = pd.read_excel('data.xlsx',engine='openpyxl')
df=df.iloc[:,1:]
df.tail()


#删除数据的缺失值
df1=df.dropna()
df1.tail()


#描述一下数据的统计信息，可以清楚的看到有没有异常值
df1.describe()

#划分特征 与 标签 展现下数据形状
df2=df1
y=df2['故障类别']
x=df2.drop(columns = ['故障类别'])
x.shape,y.shape

#划分训练集测试集，取 10%的测试集。
from sklearn.model_selection import train_test_split
train_data,test_data, train_labels, test_labels = train_test_split(x,
                                                    y,
                                                    test_size = 0.1,
                                                    random_state = 0)
len(test_data)#看一下测试集长度
```

Step 2:决策树模型

```
#决策树方法
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
model=DecisionTreeClassifier()
model.fit(train_data,train_labels)
predict_target=model.predict(train_data)
```

```
print("训练集:")
print("预测正确数量,训练集样本量:")
print(sum(predict_target == train_labels),len(train_labels))
print("精确度等指标：")
print(metrics.classification_report(train_labels,predict_target))
print("混淆矩阵：")
print(metrics.confusion_matrix(train_labels,predict_target))
print("测试集:")
predict_target2=model.predict(test_data)
print("预测正确数量,测试集样本量:")
print(sum(predict_target2 == test_labels),len(test_labels))
print("精确度等指标：")
print(metrics.classification_report(test_labels,predict_target2))
print("混淆矩阵：")
print(metrics.confusion_matrix(test_labels,predict_target2))

#在训练集上的预测效果,
print("在训练集上的预测效果:")
plt.plot(train_labels.values[20:50])
plt.plot(predict_target[20:50],'o')
plt.legend(('y_train','y_predict'))
plt.title('train')
plt.show()

#在训练集上的预测效果
print("在测试集上的预测效果:")
plt.plot(test_labels.values)
plt.plot(predict_target2,'o')
plt.legend(('y_test','y_predict'))
plt.title('test')
plt.show()
```

Step 3:图表展示

```
训练集:
预测正确数量,训练集样本量:
326 328
精确度等指标:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       192
           1       1.00      0.98      0.99       105
           2       1.00      1.00      1.00        31

    accuracy                           0.99       328
   macro avg       1.00      0.99      1.00       328
weighted avg       0.99      0.99      0.99       328

混淆矩阵:
[[192   0   0]
 [  2 103   0]
 [  0   0  31]]
```

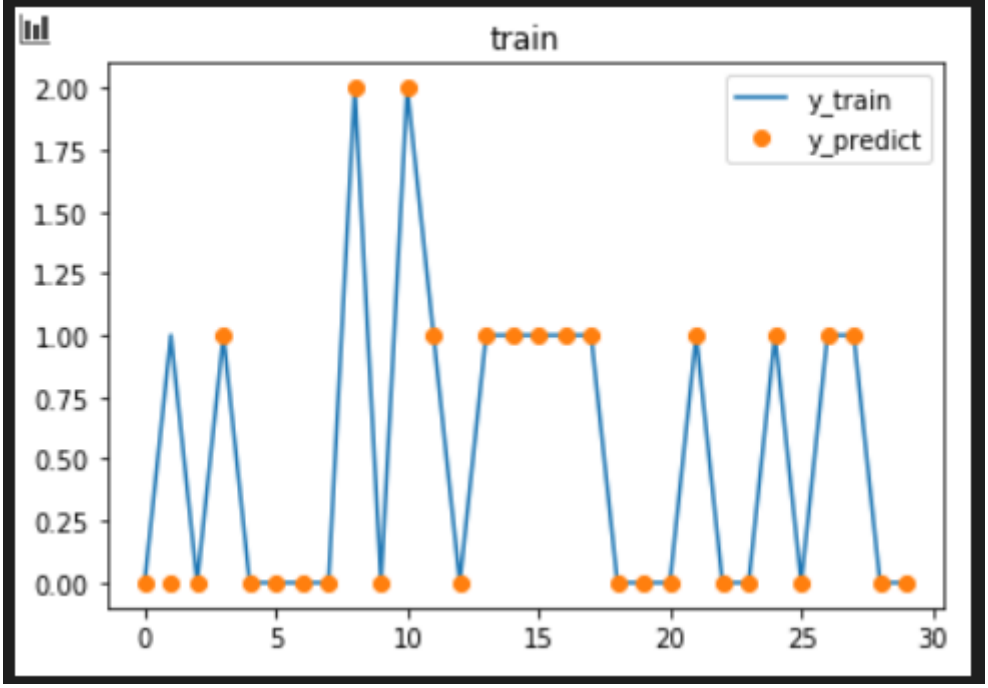
```
测试集:
预测正确数量,测试集样本量:
28 37
精确度等指标:
              precision    recall  f1-score   support

           0       0.95      0.82      0.88        22
           1       0.60      0.75      0.67        12
           2       0.33      0.33      0.33         3

    accuracy                           0.76        37
   macro avg       0.63      0.63      0.63        37
weighted avg       0.78      0.76      0.77        37

混淆矩阵:
[[18  4  0]
 [ 1  9  2]
 [ 0  2  1]]
```

在训练集上的预测效果：



在测试集上的预测效果：