机器学习初步实验报告

姓名：卜梦煜　　学号：1120192419　　班级：07111905

1. 实验名称

决策树的构建与应用

2. 实验目的

熟练掌握决策树原理，编程实现决策树完成二分类任务。

3. 实验内容

该实验需要自选编程语言，分别构造一个基于多数投票算法的二分类器和一个基于决策树的二分类器，在四个数据集上进行分类预测，并衡量模型的预测性能。

4. 实验环境

PyCharm Community Edition 2021.3.2 + sklearn 0.24.2

5. 实验过程与步骤

本实验首先阅读和理解实验要求，整理对决策树生成的特殊要求，并查找 sklearn.tree 库中的函数及参数说明，尝试使用自动工具生成决策树。查找文档确认自动工具可满足要求后，按要求设计和编写 inspection.py 和 decisionTree.py 程序。最后在四个数据集上测试，观察决策树结构，并衡量决策树的性能。

5.1 整理实验要求，查找相关文档

对实验说明中关于 inspection.py 程序的输入、程序、输出的要求整理如下：

inspection.py要求及处理

1. 信息熵取以2为底的对数

```
np.log2()    # 计算以2为底的对数值
```

2. 输出的结果文件包括：entropy和error的值

整理实验说明中关于 decisionTree.py 程序的输入、决策树构建、输出的要求，查找 sklearn.tree 库中相关的函数及参数说明，整理结果如下：

decisionTree.py要求及处理

1. 子结点划分原则：计算互信息（即信息增益entropy），选互信息最大的属性分裂
2. 停止划分条件：`互信息<=0` 或 `节点深度>max_depth`，节点深度从0开始计数

```
sklearn.tree.DecisionTreeClassifier(
    criterion="entropy",          # 互信息最大的属性分裂
    max_depth=3,                  # 生成决策树的最大深度
    min_impurity_decrease=1e-9    # 互信息<=0时停止分裂
)
```

3. 叶结点label根据Majority Vote选出，若Majority Vote值相同则选字典序大的label

```
sklearn.tree.predict_proba()      # 输出概率值，若最大概率值相同则选字典序大的作为label
```

4. 特殊情况：`max_depth=0`，`max_depth>number of attributes`
   - `max_depth=0`时，手写Majority Vote Classifier进行预测
   - `max_depth>number of attributes`时，sklearn.tree可以自动建立层数较小的决策树进行预测
5. 输出结果包括：
   - 终端输出决策树结构
   - 输出train.labels、test.labels、metrics.txt文件，分别包括训练集预测的标签、测试集预测的标签、训练集和测试集上的错误率（召回率）

由以上可知，对本实验，决策树的自动生成工具是可用的，因此将使用自动生成工具构建决策树。

5.2 设计程序，编写代码

分别编写 inspection.py 和 decisionTree.py 文件，两个文件的代码流程相似，模块包括：加载数据、数据预处理与分割、配置与训练模型、模型预测、结果评估。

5.2.1 设计并实现 inspection.py

首先利用 pandas 库读入.tsv 数据，并封装成 pandas.DataFrame 格式。

```
def preprocessTSV(self, inTSVPath: str):
    data = pd.read_csv(inTSVPath, sep='\t', header=0)
    data = pd.DataFrame(data)
    return data
```

按照实验要求，编写基于 Majority Vote 的分类器，并计算采用该分类器进行预测时的信息熵和错误率。这里用到了 pandas.value_counts()函数，该函数对表中数据值进行计数并排序。

```
def calculateRootEntropy(self, data: pd.DataFrame):
    label = data.keys()[-1]
    rate = pd.value_counts(data[label]) / len(data[label])
    return sum(-1*rate*np.log2(rate))

def calculateMajorityVoteErrorRate(self, data: pd.DataFrame):
    label = data.keys()[-1]
    rate = pd.value_counts(data[label]) / len(data[label])
    return 1 - rate.max()
```

最后输出结果文件，包括该 Majority Vote 分类器的信息熵和错误率的值。

```
def writeResultFile(self, outTXTFile, rootEntropy, majorityVoteErrorRate):
    result = "entropy: " + str(rootEntropy) + "\nerror: " + str(majorityVoteErrorRate)
    f = open(outTXTFile, "w")
    f.writelines(result)
```

主控程序如下：

```
def run(self):
    data = self.preprocessTSV(self.inTSVPath)
    rootEntropy = self.calculateRootEntropy(data)
    majorityVoteErrorRate = self.calculateMajorityVoteErrorRate(data)
    self.writeResultFile(self.outTXTPath, rootEntropy, majorityVoteErrorRate)
```

5.2.2 设计并实现 decisionTree.py

首先加载数据并对数据做预处理。将字符串替换为数字，记录映射关系，以匹配决策树输入数据的格式要求。

```
# 载入数据并做预处理
def loadData(self, inPath):
    data = pd.read_csv(inPath, sep='\t', header=0)
    data = pd.DataFrame(data)

    x = data[data.keys()[: -1]]
    y = data[data.keys()[-1: ]]

    self.features = x.keys()

    # 手动构建数据集字母替换为数字的映射
    if len(self.numToFeature) == 0:
        if "n" in x.values and "democrat" in y.values:    # politicians
            self.numToFeature.append("n")
            self.numToFeature.append("y")
            self.numToLabel.append("democrat")
            self.numToLabel.append("republican")
        elif "notA" in x.values and "notA" in y.values:   # education
            self.numToFeature.append("notA")
            self.numToFeature.append("A")
            self.numToLabel.append("A")
            self.numToLabel.append("notA")
        else:                                             # mushroom
            self.numToFeature.append(0)
            self.numToFeature.append(1)
            self.numToLabel.append(0)
            self.numToLabel.append(1)

    # 数据集字母替换为数字
    for i in range(len(self.numToFeature)):
        x = x.replace(self.numToFeature[i], i)
    for i in range(len(self.numToLabel)):
        y = y.replace(self.numToLabel[i], i)

    return x, y
```

构建决策树，包括利用 sklearn.tree.DecisionTreeClassifier()构建决策树，在决策树深度大于 0 时使用；手动构建基于 Majority Vote 分类器，在决策树深度等于 0 时使用。

决策树深度>0 时，决策树预测需输出属于每个标签的概率，选最大概率值对应的标签为预测标签，存在相同最大概率时选字典序大的标签作为该条数据的预测标签。sklearn.tree.DecisionTreeClassifier()的构建与预测如下：

```python
# 构建决策树，maxDepth=0时使用
def buildDecisionTree(self, xTrain, yTrain, maxDepth):
    model = tree.DecisionTreeClassifier(
        criterion="entropy",
        max_depth=maxDepth,
        min_impurity_decrease=1e-9
    )
    model = model.fit(xTrain, yTrain)
    return model

# 处理预测值，相同概率标签选字典序大的
def predict(self, decisionTree: tree.DecisionTreeClassifier, data):
    probas = decisionTree.predict_proba(data)
    predict = []
    for proba in probas:
        candidate = []
        for i in range(len(proba)):
            if math.isclose(proba[i], max(proba)):
                candidate.append(self.numToLabel[i])
        predict.append(max(candidate))
    return predict
```

Majority Vote 分类器的构建与预测如下：

```python
# Majority Vote Classifier，max_depth=0时使用
def majorVotePredict(self, yTrain, yTest):
    label = yTrain.keys()[-1]
    label = pd.value_counts(yTrain[label]).sort_index(ascending=False).sort_values(ascending=False).idxmax()
    return [self.numToLabel[label] for i in range(len(yTrain))], [self.numToLabel[label] for i in range(len(yTest))]
```

最后按照实验要求撰写程序输出，包括输出.labels 文件、计算并输出 metrics.txt 文件、打印决策树。

输出.labels 文件。直接将预测结果写入文件：

```python
def writePredictLabel(self, predict, outLabelPath):
    f = open(outLabelPath, "w")
    for label in predict:
        f.writelines(str(label) + "\n")
    f.close()
```

计算并输出 metrics.txt 文件。计算时使用了 sklearn.metrics.classification_report()函数，该函数输入预测值和真实值，输出 precision、recall、f1_score、support 四个评价指标。本实验要求输出的错误率为指标中的召回率。

```
def judgeMetrics(self, predict, label):
    label = label.values.reshape(-1)
    label = [self.numToLabel[label[i]] for i in range(len(label))]
    measure = classification_report(y_true=label, y_pred=predict, output_dict=True)
    measure = pd.DataFrame(measure).transpose()
    return 1 - measure["recall"]["accuracy"]

def writeMetrics(self, trainPredict, testPredict, yTrain, yTest, outMetricsPath):
    trainError = self.judgeMetrics(trainPredict, yTrain)
    testError = self.judgeMetrics(testPredict, yTest)
    f = open(outMetricsPath, "w")
    f.write("error(train): " + str(trainError) + "\nerror(test): " + str(testError))
    f.close()
```

打印决策树结构，分为决策树深度>0 时决策树信息，和决策树深度等于 0 时多数投票分配器信息。

对 sklearn.tree.DecisionTreeClassifier 自动生成的决策树，决策树全部细节信息存储在 sklearn.tree.DecisionTreeClassifier.tree_中，根据本实验需求，取出决策树中 node_count、children_left、children_right、feature、value，这些参数均为列表形式，分别表示决策树结点数量、当前结点左子树、当前结点右子树、当前结点切分特征的编号、当前结点中各 label 的数量。根据示例，按照深度优先顺序从左到右打印决策树。

```
# 打印决策树
def printDecisionTree(self, decisionTree: tree.DecisionTreeClassifier, pre, now, depth):
    nodeCnt = decisionTree.tree_.node_count    # 结点数量
    childrenLeft = decisionTree.tree_.children_left    # 左子树
    childrenRight = decisionTree.tree_.children_right    # 右子树
    feature = decisionTree.tree_.feature    # 切分结点的属性
    value = decisionTree.tree_.value    # 各结点中各label的个数

    line = ""
    if pre != -1:
        line += "| " * depth + \
                self.features[feature[pre]] + \
                " = " + \
                (str(self.numToFeature[0]) if childrenLeft[pre] == now else str(self.numToFeature[1])) + \
                ": "
    line += "["
    for i in range(len(self.numToLabel)):
        num = int(value[now][0][i])
        line += str(num) + " " + str(self.numToLabel[i]) + ("/" if i != len(self.numToLabel) - 1 else "]")
    print(line)
    if now < nodeCnt and childrenLeft[now] != -1 and childrenRight[now] != -1:
        self.printDecisionTree(decisionTree, now, childrenRight[now], depth + 1)
        self.printDecisionTree(decisionTree, now, childrenLeft[now], depth + 1)
```

对 Majority Vote 分类器，只需输出原始训练样本中标签数量的统计即可。

```
# 打印Majority Vote Classifier分类器
def printMajorVoteClassifier(self, yTrain):
    label = yTrain.keys()[-1]
    label = pd.value_counts(yTrain[label])
    line = "["
    for i in range(len(self.numToLabel)):
        line += str(label[i]) + " " + str(self.numToLabel[i]) + ("/" if i != len(self.numToLabel) - 1 else "]")
    print(line)
```

主控程序如下：

```
def run(self):
    xTrain, yTrain = self.loadData(self.inTrainPath)
    xTest, yTest = self.loadData(self.inTestPath)
    if self.maxDepth > 0:
        decisionTree = self.buildDecisionTree(xTrain, yTrain, self.maxDepth)
        self.printDecisionTree(decisionTree, -1, 0, 0)
        trainPredict = self.predict(decisionTree, xTrain)
        testPredict = self.predict(decisionTree, xTest)
    else:
        trainPredict, testPredict = self.majorVotePredict(yTrain, yTest)
        self.printMajorVoteClassifier(yTrain)
    self.writePredictLabel(trainPredict, self.outTrainLabelPath)
    self.writePredictLabel(testPredict, self.outTestLabelPath)
    self.writeMetrics(trainPredict, testPredict, yTrain, yTest, self.outMetricsPath)
```

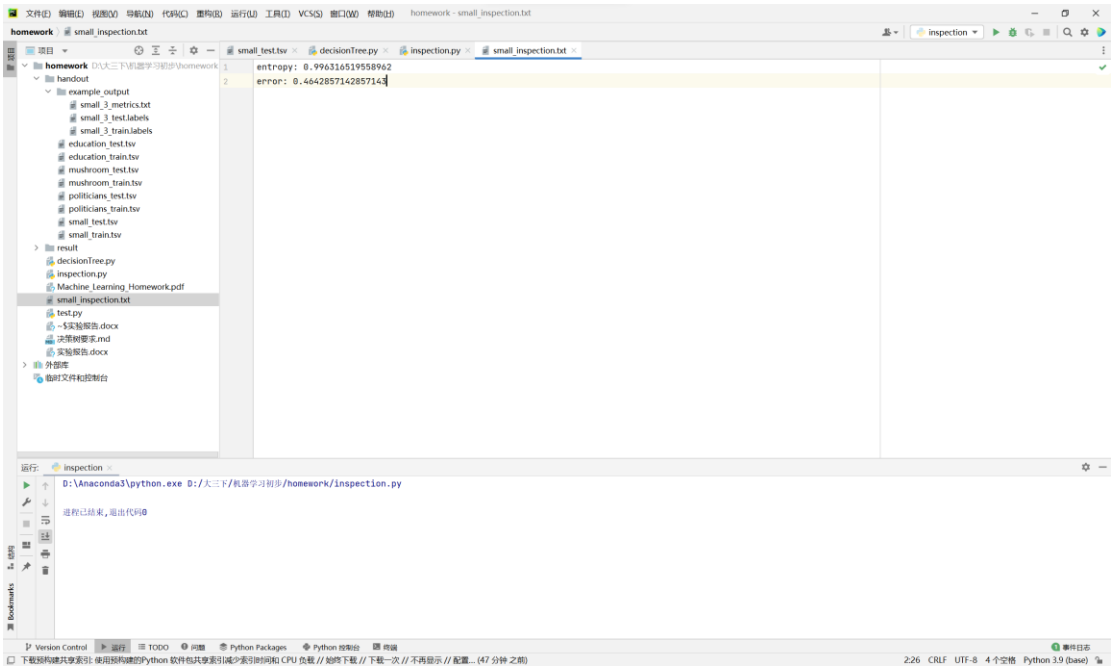
5.3 实验测试

在 small、politicians、education、mushroom 四个数据集上分别测试 inspection.py 和 decisionTree.py 程序。对 inspection.py，需检查在四个数据集上输出的交叉熵与误差是否正确。对 decisionTree.py，需检查在四个数据集上输出的预测标签.labels 文件、训练集测试集误差 metrics.txt 文件、生成的决策树结构是否正确，并调节树的深度参数，观察在不同条件尤其是边界条件下的输出。

6. 实验结果与分析

6.1 实验结果呈现

（1）对 inspection.py 程序，按照 5.3 中的方法进行测试，以 small 数据集为例展示实验结果如下：



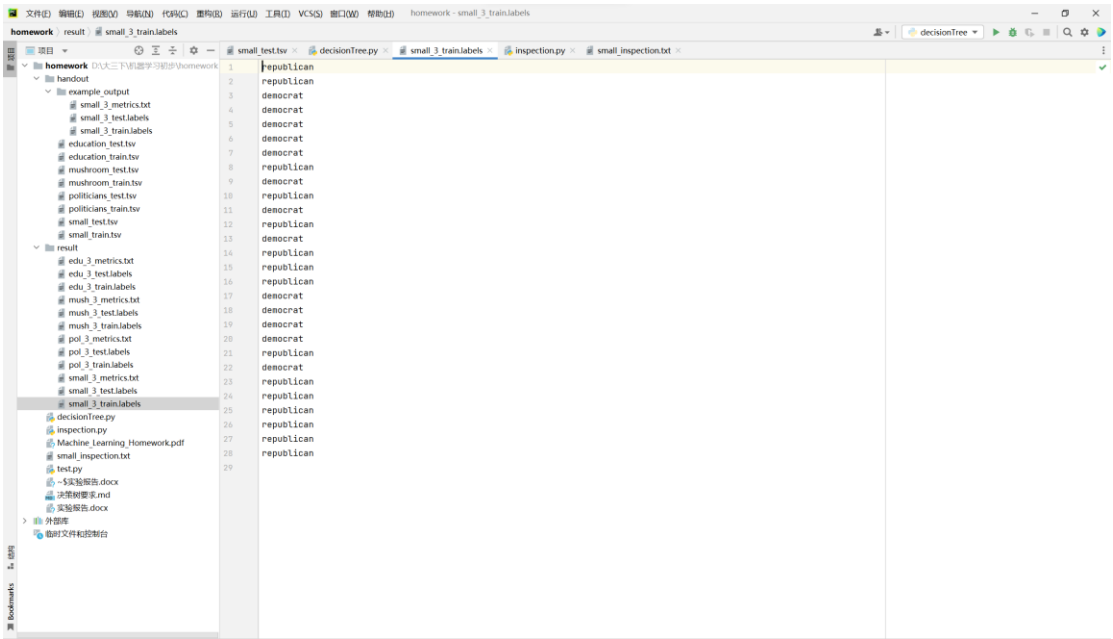
对比实验说明中测试用例结果，inspection.py 程序能够满足实验要求，正确输出所需文件。

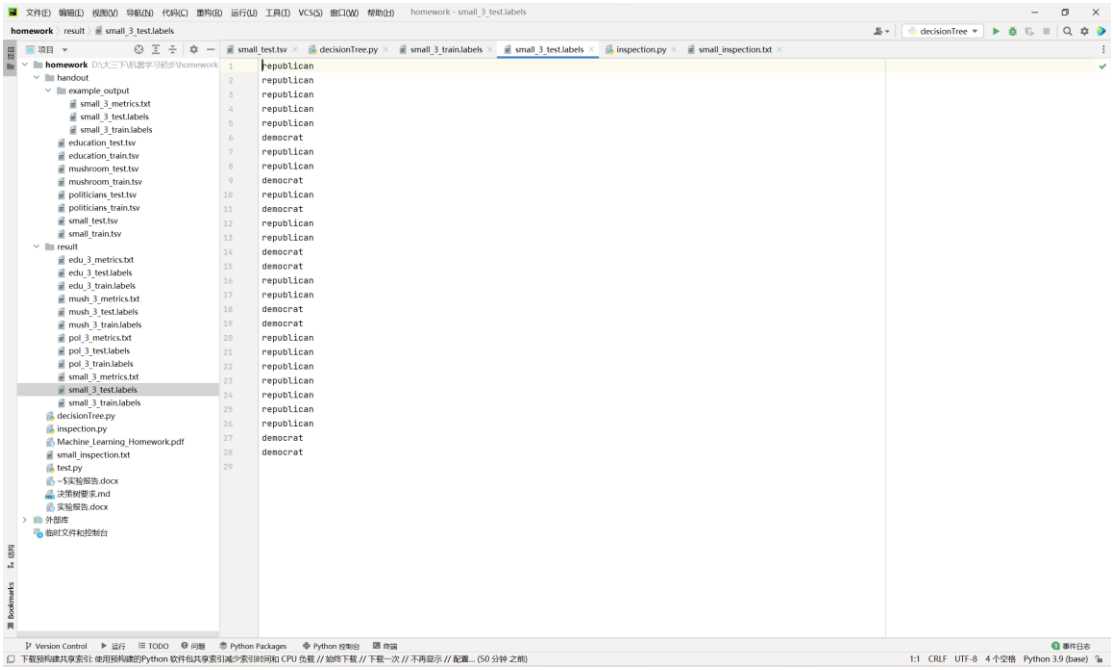
（2）对 decisionTree.py 程序，按照 5.3 中的方法进行测试，以 small 数据集在 max_depth=3 时为例展示实验结果如下。

生成的决策树结构:

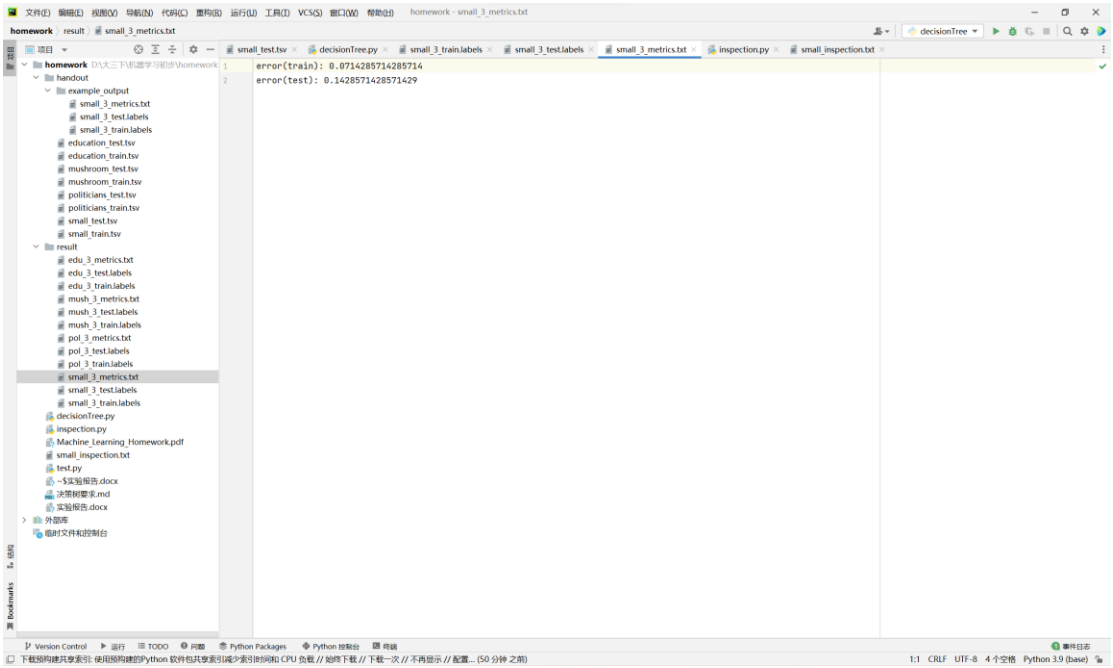
```
[15 democrat/13 republican]
| Anti_satellite_test_ban = y: [13 democrat/1 republican]
| | Export_south_africa = y: [13 democrat/0 republican]
| | Export_south_africa = n: [0 democrat/1 republican]
| Anti_satellite_test_ban = n: [2 democrat/12 republican]
| | Export_south_africa = y: [2 democrat/7 republican]
| | Export_south_africa = n: [0 democrat/5 republican]
```

生成的.labels 文件:





生成的 metrics.txt 文件：



small 数据集上训练集错误率 0.0714285714285714，测试集错误率 0.1428571428571429。对比实验说明中测试用例结果，decisionTree.py 程序能够满足实验要求，正确输出所需文件。

对 politicians、education、mushroom 数据集在 max_depth=3 时的错误率统计如下：

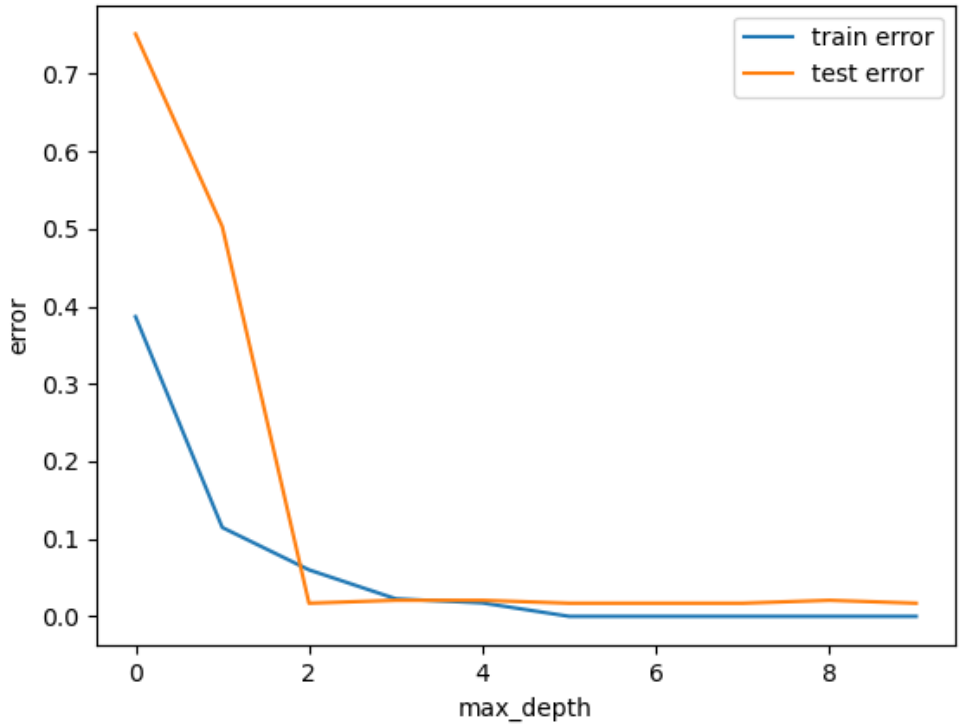
max_depth=3	train	test
politicians	0.11409396	0.16867470
education	0.17000000	0.20500000
mushroom	0.02266667	0.02071563

6.2 实验结果分析

（1）从数据集大小的角度进行分析。当训练集过小时（如 small 数据集），建立的模型精度过低，表现为对新数据预测的准确率与训练集准确率差距较大，因此模型不具有参考价值，随着数据集的增大，建立的模型分类精度也会提高，表现为对新数据预测的准确率与训练集准确率差距更接近。

（2）从决策树剪枝的角度对实验结果进行分析。由于 mushroom 数据集较大，随机误差较小，以下分析均在 mushroom 数据集上进行。

设置不同 max_depth，对比决策树在训练集和验证集上准确率变化：



max_depth=9 时生成的决策树:

```
[3680 0/2320 1]
| odor_foul = 1: [0 0/1632 1]
| odor_foul = 0: [3680 0/688 1]
| | gill-size_broad = 1: [3392 0/72 1]
| | | spore-print-color_green = 1: [0 0/72 1]
| | | spore-print-color_green = 0: [3392 0/0 1]
| | gill-size_broad = 0: [288 0/616 1]
| | | odor_none = 1: [192 0/40 1]
| | | | stalk-surface-below-ring_scaly = 1: [0 0/32 1]
| | | | stalk-surface-below-ring_scaly = 0: [192 0/8 1]
| | | | | bruises?_bruises = 1: [0 0/8 1]
| | | | | bruises?_bruises = 0: [192 0/0 1]
| | | odor_none = 0: [96 0/576 1]
| | | | gill-spacing_crowded = 1: [96 0/96 1]
| | | | | odor_creosote = 1: [0 0/96 1]
| | | | | odor_creosote = 0: [96 0/0 1]
| | | | gill-spacing_crowded = 0: [0 0/480 1]
```

可以看到，在决策树深度较小时，增加层数可以使决策树准确率有显著提升。对含有 122 个特征的 mushroom 数据集，决策树深度超过 5 后训练集错误率已降至 0，测试集错误率在

小范围波动，并未出现过拟合现象，说明对 mushroom 数据集，实际与蘑菇毒性相关的属性并不多。同时，当 max_depth=9 时，生成的决策树深度仅为 5，说明采用“限制最大深度”和“互信息小于 0 时停止分裂”的剪枝是有效的，可以避免过拟合。

7. 个人收获

本次实验是我第一次编写决策树相关的代码，刚开始我对决策树的构建没有整体的认识，对 sklearn 库提供的决策树生成工具也不完全理解，因此花了很多时间阅读文档，理解 sklearn 库的决策树相关函数及其参数含义。在理解了相关参数后，我判断 sklearn.tree 库提供的自动工具可以满足要求，之后便是写代码与调试的过程。通过本次实验，我有如下收获：

（1）对决策树的原理有了更深刻的认识，包括决策树结点分裂条件、最佳分裂特征的选择、叶结点属性选取等等，同时也对决策树剪枝原理及方法有了更加深入的认识。

（2）对 sklearn.tree 库有了一定了解，知道如何使用该库实现基本的决策树算法和简单的剪枝，对该库的部分函数参数有了深入了解，能够通过调整参数生成不同的决策树模型。