Machine Learning Homework*

Zifan Zheng(郑子帆)†
Beijing Institute of Technology
School of Computer Science and Technology

日期： April 30, 2023

Abstract

This paper presents the final project report for the "Introduction to Machine Learning" course at Beijing Institute of Technology in the spring semester of 2023. This project utilizes Python to implement two classifiers, namely Inspection and Decision Tree, and conducts experiments on four datasets: politician, education, small, and mushrooms. The main focus of this paper is to elaborate on the design, implementation, and experimental results and analysis of the two classifiers.[1].

1 Inspecting the Data

1.1 Overview of the Purpose

Develop a generalized classifier program, *inspection.py*, which employs the majority voting method for classification. Calculate the information entropy and error rate (i.e., the proportion of incorrect samples) for the dataset, and output the results.

When executing the program, provide the file path of the dataset to be analyzed and the output file path as command line arguments to the program. For example, `python3 inspection.py <input> <output>`.

*I Chose to write the report paper in English because the course slides as well as the assignment document are in english, so writting in english is better fit with the course than in Chinese. My English proficiency is limited, so please forgive any errors or omissions.

†Class No.07112002; Student ID 1120200822

[1]Given that the overall difficulty of this assignment is not too high, but considering its significant weight in the course assessment, this report is written in a detailed manner. It elaborates on some fundamental and straightforward issues, aiming to enable even beginners to easily understand the content.

1.2 Design Ideas

1.2.1 Data Input and Preprocessing

Different from my previous project, the input file for this project is in .tsv format, which is distinct from the more commonly used .csv format in that the former is tab-separated while the latter is comma-separated. As such, when using the *read_csv* function from the pandas package, it is necessary to include the parameter `delimiter='\t'`.

The implementation of this classifier does not require information on the specific data attributes in the dataset. It only needs to traverse each row of the DataFrame and extract the labels. In Python, we can use a **dictionary** to count the number of instances for each label in the dataset. For example, after processing the *small_train.tsv* file, the resulting dictionary would be {"democrat": 15, "republican": 13}.

1.2.2 Calculate Label Entropy

After obtaining the list and dictionary containing the statistics of labels, and given that the dataset contains only two label values, we can calculate the information entropy using Formula 1, where p_0 and p_1 denote the frequencies of the two label values.

$$\textbf{Label Entropy} = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \tag{1}$$

1.2.3 Calculate Error Rate

Calculating the error rate is even simpler. Given that the dataset contains two label values, denoted by n_0 and n_1, respectively, we can compute the error rate using Formula 2.

$$\textbf{Error Rate} = \frac{\min(n_0, n_1)}{n_0 + n_1} \tag{2}$$

1.3 Code Implementation

The code implementation of this part is relatively straightforward. The function *Data_Preprocessor* and *Calc_Label_Entropy* are shown as follows.

Function *Data_Preprocessor*:

```
'''
input: 文件路径
output: 数据的 Label 的 list, Label 的 dict
'''
def Data_Preprocessor(filepath):
    # 因为读入的是 tsv, 为制表符分隔
```

```
    InputData = pd.read_csv(filepath, delimiter='\t')
    Label = []
    LDict = {}  # 创建 dict 记录每个 label 出现的次数
    for _, row in InputData.iterrows():
        row_label = row[-1] # 最后一列为 label
        Label.append(row_label)
        LDict[row_label] = LDict.get(row_label, 0) + 1
    return Label, LDict
```

Function *Calc_Label_Entropy*:

```
'''
input: Label 的 dict, 数据样本数 (即 Label 数)
ouput: Label Entropy
'''
def Calc_Label_Entropy(LDict, SampleNum):
    LabelEnt = 0
    for _, value in LDict.items():
        frac = value / SampleNum
        LabelEnt -= frac * math.log(frac, 2)
    return LabelEnt
```

2 Decision Tree Based on Mutual Information(ID3)

2.1 Overview of the Purpose

Write the program*decisionTree.py* , implementing a Decision Tree Classification Model. The requirements are:

- The program should take command line arguments for the following inputs: the training dataset file (train input), the testing dataset file (test input), the maximum depth of the decision tree (max depth), the predicted output file for the training dataset (train out), the predicted output file for the testing dataset (test out), and the output file for error rates (metrics out). Specifically, the command format should be `python3 decisionTree.py <train input> ... <metrics out>`.

- Split the Node's dataset using **mutual information**[2]. The calculation formula is shown in Formula 3.
- Split a feature only when its information gain is greater than 0, otherwise stop splitting.
- After splitting on a feature, it should be kept in the child datasets for further splitting.
- The depth of the decision tree should not exceed the maximum depth specified in the command line arguments.
- Use majority voting to classify the dataset represented by each leaf node. If multiple labels have the same number of data instances, choose the label with the last lexicographical label.
- When choosing the feature to split on, if there are multiple attributes have the same mutual information which is the highest one, choose the first attribute.

2.2 Design Ideas

Based on the requirements and needs listed in Section 2.1 and the hints provided in the assignment document, the implementation of the program should be divided into the following steps:

1. Read data from .tsv files, process the data to obtain attribute names, attribute values, and label values.
2. Generate the decision tree using the training dataset.
3. Predict the classes of the samples in the training dataset and the testing dataset using the generated decision tree, and output them to the corresponding files. Calculate the error rates and output them to a file.
4. Output the specific information of the decision tree, the print format is shown in the assignment document.

Below, we will provide a brief outline of the design ideas for each of these four parts.

2.2.1 Read the Data

Write the function *Read_Data*, read the file by using function *read_csv* in pandas package.

[2]According to the lecture slides and "Machine Learning" by Zhou et al.[4], the commonly used term for decision tree construction is "Information Gain". However, the assignment document[1] uses the term "Mutual Information". After consulting more literature, I found that there are some differences in definition and meaning between the two terms, but they have the same calculation formula and results. Therefore I used the same function to calculate mutual information and information gain.

The training and testing datasets, attribute set, label set, etc. are obtained by using relevant functions provided by the pandas package. They are then converted to lists to facilitate subsequent calculations.

For this assignment, I did **NOT** consider using numpy, mainly because I wanted to minimize the number of imported packages in the code to avoid potential errors when running on different computers. In addition, after some simple testing, I found that using lists is slightly faster than numpy for processing the given datasets in a similar code framework (of course, the running time is still within 1 second).

2.2.2 Generate the Decision Tree

Write the function *Gen_Tree*. Because the decision tree is a tree structure, we design a TreeNode class with a member "child". We can connect the child nodes and their father through attribute values.

Generating a decision tree is essentially a **recursive algorithm**, so we also need to design a function *split* to select the feature to be split, and to perform dataset splitting, etc. The pseudo-code of this function is shown in Algorithm 1.

2.2.3 Predictions and Outputs

After generating the decision tree, we obtain a tree that can be used to classify data. The prediction function *Prediction* can classify each data point by traversing the tree one by one based on its attributes. Specifically, we start from the root node and select the edge corresponding to the attribute value of the data at each node to reach the child node, until we reach a leaf node to complete the classification.

Because we need to calculate the error rate, we need to count the number of samples for each predicted label during the classification process, and finally calculate the error rate and return it to the main function.

In addition to outputting the above content, we also need to print the "shape" of the tree in the terminal. This process is actually a recursive process similar to generating the tree. The output content is actually a **preorder traversal** of the decision tree. Compared to generating the decision tree, the additional content required is to first count the number of samples for each of the two labels every time the recursive function *PrintTree* is called, used for outputting.

Algorithm 1: Function split

Input: X: the dataset to be split; A: attribute set; $nowNode$: the node represents the current dataset.

Function `Split(`X, A, $nowNode$`)`**:**
- **if** *the depth of the current node==max depth* **then**
  - **return**;
- **if** *the label values in* X *are all the same* **then**
  - Mark the current node as a leaf node and classify it as the label value that appears most frequently in the dataset;
  - **return**;
- **for** *the attribute in* A *that has not been used for splitting* **do**
  - Calculate the mutual information;
  - Update the selected attribute to be the one with the highest mutual information;
- **if** *no attribute that satisfies its mutual information* > 0 **then**
  - **return**;
- Split X into two child datasets $V0$ and $V1$ based on the selected attribute's values;
- Generate two child nodes $newNode0$ and $newNode1$ based on the child datasets $V0$ and $V1$, respectively;
- Connect the child nodes $newNode0$ and $newNode1$ to the current node $nowNode$ based on their corresponding attribute values;
- call `Split(`$V0$, A, $newNode0$`)`;
- call `Split(`$V1$, A, $newNode1$`)`;
- **return**;

2.3 Code Implementation

In this section, we showed some code implementations of functions mentioned above.

Function *Read_Data*:

```
# 读入并处理数据
def Read_Data(trainfile, testfile):
```

```
    InputData_train = pd.read_csv(trainfile, delimiter='\t')
    InputData_test = pd.read_csv(testfile, delimiter='\t')
    # 将训练、测试数据集转成字符串的 list
    train_input = InputData_train.values.tolist()
    train_input = [[str(el) for el in data] for data in train_input]
    test_input = InputData_test.values.tolist()
    test_input = [[str(el) for el in data] for data in test_input]
    num_rows, num_cols = InputData_train.shape
    attr_name = []  # 每个属性的两个取值
    label_name = [] # label 的名字
    attr_title = [] # 属性的名称

    for _, col in InputData_train.iloc[:, :-1].items():
        col_unique = col.unique().tolist()
        # 此属性只有一个取值, 为方便后续计算, 添加一个空取值
        if len(col_unique) < 2:
            col_unique.append("null")
        col_unique = [str(elem) for elem in col_unique] # 转为字符串
        col_unique.sort()
        attr_name.append(col_unique)

    label_name = InputData_train.iloc[:, -1].unique().tolist()
    label_name = [str(elem) for elem in label_name]
    label_name.sort()

    attr_title = InputData_train.columns.to_list()[:-1]
    attr_title = [str(elem) for elem in attr_title]

    return train_input, test_input, attr_name, label_name, attr_title
```

Function *Gen_Tree*:

```
# 生成决策树的总函数
def Gen_Tree(X, max_depth, attr_name, label_name, attr_title):
    global NodeCnt
    NodeCnt += 1
```

```
    root = TreeNode(NodeCnt)
    isSplited = [0 for i in range(len(attr_name))]
    split(X, root, 0, max_depth, isSplited, attr_name, label_name)
    return root
```

Function *split*:

```
# 划分当前数据集的递归函数
def split(X, nowNode, now_depth, max_depth,
          isSplited, attr_name, label_name):
    if now_depth == max_depth:    # 到达最大深度
        nowNode.SetLeaf(X, label_name)  # 标记叶节点并分类
        return

    # 如果当前待分数据集的 label 都一样则标为叶节点
    if all(sample[-1] ==  X[0][-1] for sample in X):
        nowNode.SetLeaf(X, label_name)
        return
    # 选择最合适的 attribute
    max_ent_ind, max_MI = -1, 0
    for i, tag in enumerate(isSplited):
        if tag == 1:     # 该属性已被用于过划分, 跳过
            continue
        # 获取某一特定属性值的子数据集
        V0 = [sample for sample in X if sample[i] == attr_name[i][0]]
        V1 = [sample for sample in X if sample[i] == attr_name[i][1]]
        if len(V0) == 0 or len(V1) == 0:
            continue
        MI = Calc_Mutual_Information(V0, V1, label_name) # 互信息
        # MI = Calc_Gain_Rate(V0, V1, label_name)    # 增益率
        # MI = Calc_Gini(V0, V1, label_name)         # Gini 系数
        # print(attr_title[i], MI)
        if MI > max_MI: # 更新当前最大 MI 的属性
            max_MI = MI
            max_ent_ind = i
```

```
    # 没有可选的 attribute
    if max_ent_ind == -1:
        nowNode.SetLeaf(X, label_name)   # 标记叶节点并分类
        return

    # 划分并递归
    global NodeCnt
    nowNode.crit = max_ent_ind
    isSplited[max_ent_ind] = 1
    now_depth += 1
    NodeCnt += 1
    newNode = TreeNode(NodeCnt)
    nowNode.AddNode(attr_name[max_ent_ind][0], newNode)
    V0 = [sample for sample in X if
          sample[max_ent_ind] == attr_name[max_ent_ind][0]]
    split(V0, newNode, now_depth, max_depth,
          isSplited, attr_name, label_name)

    NodeCnt += 1
    newNode = TreeNode(NodeCnt)
    nowNode.AddNode(attr_name[max_ent_ind][1], newNode)
    V1 = [sample for sample in X if
          sample[max_ent_ind] == attr_name[max_ent_ind][1]]
    split(V1, newNode, now_depth, max_depth,
          isSplited, attr_name, label_name)
    now_depth -= 1   # 回溯
    isSplited[max_ent_ind] = 0   # 回溯
    return
```

3 Different Desicion Tree Algorithms

In Section 2, we implemented a decision tree based on the mutual information method to select the best attribute. However, according to what we have learned in class and related books, there are many other decision tree generation methods for discrete attributes. For example, we can solve the problem of "overfitting" by pruning to enhance the generalization ability of the model. In addition, we can use other methods

to select the "best attribute" for each split. In this section, we will explore the latter in more depth.

3.1 Select Features Based on Mutual Information(ID3)

In Section 2, the decision tree we implemented adopts the method of calculating mutual information to select the attribute to be split each time. The calculation formula is as follows:

$$\begin{aligned} I(Y;X) &= H(Y) - H(Y \mid X) \\ &= H(Y) - P(X=0)H(Y \mid X=0) - P(X=1)H(Y \mid X=1) \end{aligned} \tag{3}$$

In code implementation, we wrote a function *Calc_Mutual_Information*:

```
# 计算指定属性的互信息
def Calc_Mutual_Information(V0, V1, label_name):
    TotEnt, Ent0, Ent1 = 0, 0, 0
    # 分别统计两个子数据集的两个标签的样本数
    cnt00 = sum(1 for sample in V0 if sample[-1] == label_name[0])
    cnt01 = sum(1 for sample in V0 if sample[-1] == label_name[1])
    cnt10 = sum(1 for sample in V1 if sample[-1] == label_name[0])
    cnt11 = sum(1 for sample in V1 if sample[-1] == label_name[1])
    num0, num1 = len(V0), len(V1) # 第一、二个子数据集的样本数
    cnt0 = cnt00 + cnt10 # 总的为 label[0] 的样本数
    cnt1 = cnt01 + cnt11
    frac0 = cnt0 / (cnt0+cnt1)
    frac1 = cnt1 / (cnt0+cnt1)
    if cnt0 != 0 and cnt1 != 0:
        TotEnt = -(frac0)*math.log(frac0, 2) - frac1*math.log(frac1, 2)
    frac0 = cnt00 / num0
    frac1 = cnt01 / num0
    if cnt00 != 0 and cnt01 != 0:
        Ent0 = -(frac0)*math.log(frac0, 2) - frac1*math.log(frac1, 2)
    frac0 = cnt10 / num1
    frac1 = cnt11 / num1
    if cnt10 != 0 and cnt11 != 0:
        Ent1 = -(frac0)*math.log(frac0, 2) - frac1*math.log(frac1, 2)
    p0 = num0 / (num0+num1)
    p1 = num1 / (num0+num1)
```

```
    MI = TotEnt - p0*Ent0 - p1*Ent1
    return MI
```

We will now discuss some other methods for selecting the attribute to split on.

3.2 Select Features Based on gain ratio(C4.5)

In the development and improvement of decision tree modeling algorithms, it has been found that the method of selecting attributes based on mutual information(information gain) often tends to favor attributes with a larger number of possible values, leading to extreme instability of the model. To solve this problem, Quinlan et al.[3] invented an algorithm that selects attributes based on gain ratio.

Specifically, as shown in Equation 4, the gain ratio of an attribute is defined as the information gain of the attribute divided by its "intrinsic value". If an attribute can take more values, its "intrinsic value" will be larger, which weakens the influence of the number of possible values.

$$
\begin{aligned}
GR(Y;X) &= \frac{I(Y;X)}{SplitInfo(Y;X)} \\
SplitInfo(Y;X) &= -\sum_{i=0}^{1} P(X=i)\log_2 P(X=i) \\
&= -P(X=0)\log_2 P(X=0) - P(X=1)\log_2 P(X=1)
\end{aligned}
\tag{4}
$$

In code implementation, We can define a new function *Calc_Gain_Rate* to calculate the gain ratio of an attribute, which is the mutual information of the attribute divided by its **intrinsic value**. We can then use the result of this function along with the result of the *Calc_Mutual_Information* function to determine the attribute to split on.

Function *Calc_Gain_Rate*:

```
# 计算指定属性的增益率
def Calc_Gain_Rate(V0, V1, label_name):
    IG = Calc_Mutual_Information(V0, V1, label_name)
    num0, num1 = len(V0), len(V1)
    frac0 = num0 / (num0+num1)
    frac1 = num1 / (num0+num1)
    IV = - frac0*math.log(frac0, 2) - frac1*math.log(frac1, 2)
    Gain_Rate = IG / IV
    return Gain_Rate
```

3.3 Select Features Based on Gini Index(CART)

Different from the two previous methos, Lewis et al.[2] invented **CART** in 2000, using Gini Index to select features.

Specifically, the purity of dataset D is shown in Formula 5:

$$\text{Gini}(D) = \sum_{k=1}^{|y|} \sum_{k_k' k} p_k p_{k'} = 1 - \sum_{k=1}^{|y|} p_k^2 = 2p_0 \times (1 - p_0) \tag{5}$$

The Gini coefficient represents the probability of randomly selecting two samples with different labels in the dataset D. The smaller the value, the higher the purity of the dataset.

The definition of the Gini Index of feature a is shown in Formula 6.

$$\text{Gini}(D, a) = \sum_{v=1}^{V} \left| \frac{D^v}{D} \right| \text{Gini}\left(D^v\right) \tag{6}$$

In code implementation, we wrote function *Calc_Gini*, which calculate Gini Index following the principle of Formula 6.

```
# 计算指定属性的 Gini 系数
def Calc_Gini(V0, V1, label_name):
    num0, num1 = len(V0), len(V1)
    frac0 = num0 / (num0+num1)
    frac1 = num1 / (num0+num1)
    cnt00 = sum(1 for sample in V0 if sample[-1] == label_name[0])
    cnt01 = sum(1 for sample in V0 if sample[-1] == label_name[1])
    cnt10 = sum(1 for sample in V1 if sample[-1] == label_name[0])
    cnt11 = sum(1 for sample in V1 if sample[-1] == label_name[1])
    # 计算两个子数据集的 Gini 系数
    Gini0 = 1 - (cnt00/num0)**2 - (cnt01/num0)**2
    Gini1 = 1 - (cnt10/num1)**2 - (cnt11/num1)**2
    Gini = frac0*Gini0 + frac1*Gini1
    return Gini
```

4 Experimental Results and Analysis

In the experiment part, since there is only one parameter, max_depth, so we only analyzed its effect on the classification results in the parameter analysis. Additionally, we also compared the impact of different attribute selection methods on the results.

4.1 Experimental Environments

- OS: Windows 10
- Language: Python3.9
- Development environment: Visual Studio Code
- Python's packages: sys, pandas, math

4.2 The Impact of different max_depth on Error Rate

We implemented a program *ParaTest.py*(detailed in Appendix A) for parameter testing in this section. For max_depth, we tested its value from 0 to 10, and plotted a line chart for each dataset's training and testing set separately.

4.2.1 Decision Tree Implemented with Mutual Information

Running the program *ParaTest.py* on the four datasets, we can obtain the curves of Error Rate for different values of max_depth, as shown below.

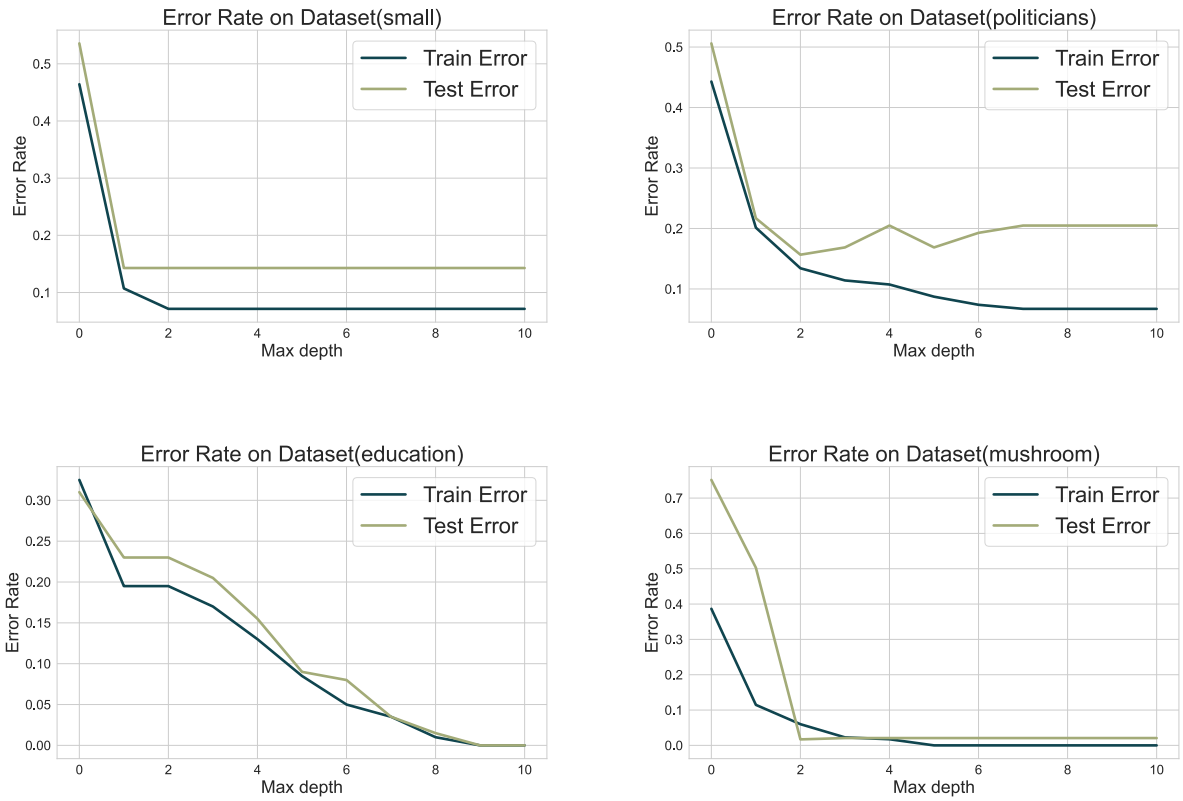

Figure 1: Mutual Information

Observing the graphs, we can find that for the small dataset, the lowest error rate was achieved at max_depth=2. Similar results were obtained for the politicians and

mushroom datasets, where a low error rate was reached relatively quickly. Overall, the train error decreased as the max_depth increased for all datasets.

However, it should be noted that for the politicians dataset, the test error was lowest at max_depth=2, and with the increase of max_depth, although the train error decreased, the test error increased, indicating **overfitting** of the model.

The best max_depths of the four datasets are 2, 3, 9, 5, respectively.

4.2.2 Decision Tree Implemented with Gain Ratio

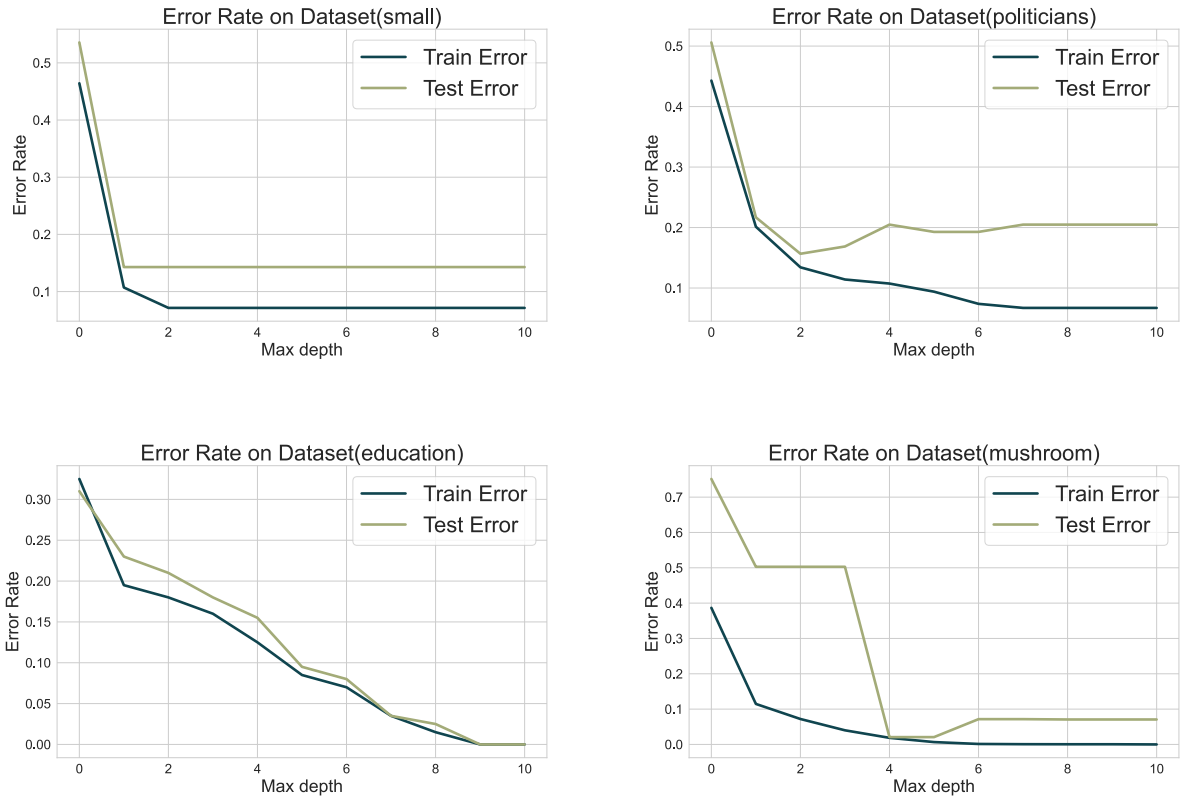

Figure 2: gain ratio

The results obtained after changing mutual information to gain ratio are shown in the above figure. The final results are very similar to Figure 1, with the train error decreasing with max_depth increasing, while overfitting occurs for larger max_depth in the politicians dataset.

The best max_depths of the four datasets are 2, 3, 9, 5, respectively.

4.2.3 Decision Tree Implemented with Gini Index

The line chart obtained from the Gini Index method is shown in Figure 3, which is similar to Figure 1 and reflects the same patterns as described before. This aspect will

not be discussed further in this section.

The best max_depths of the four datasets are 2, 2, 9, 2, respectively.

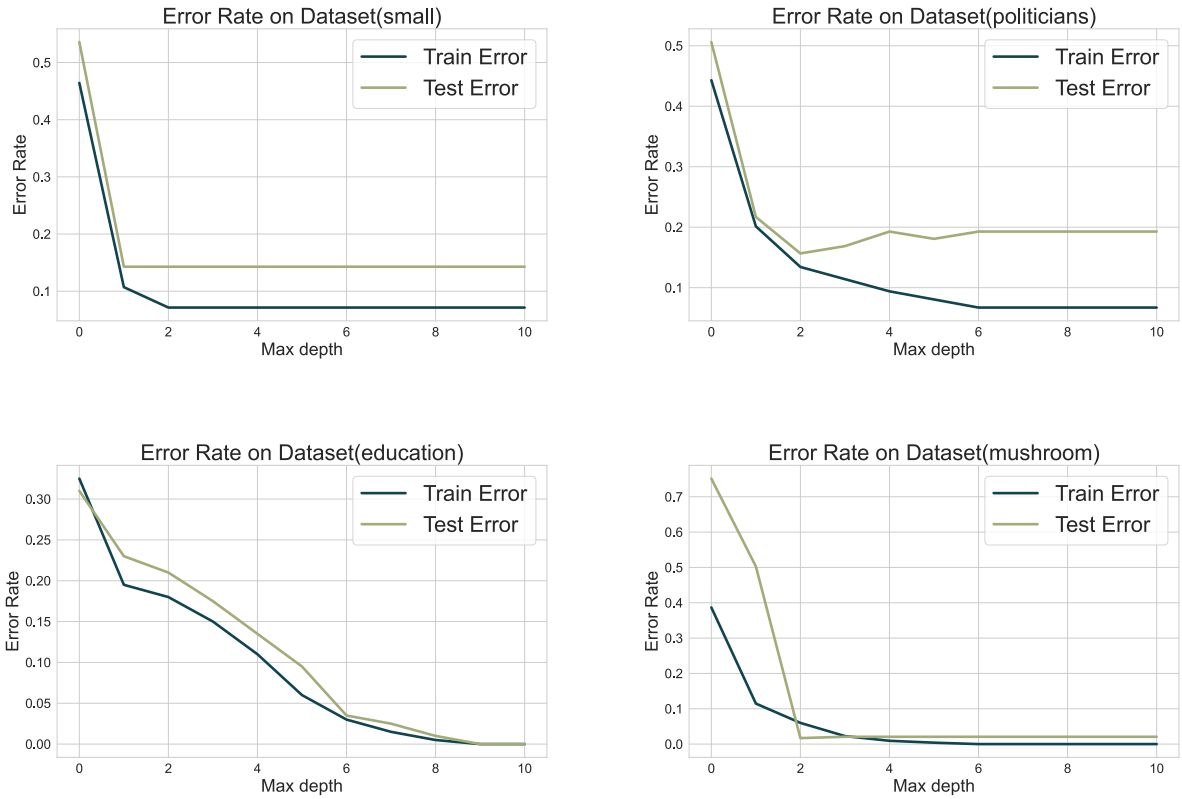

Figure 3: Gini Index

4.3 The Impact of Different Feature Selection Algorithms on Error Rate

We can have some observations by comparing the results obtained by the three methods on the same datasets. To facilitate comparison, I plotted the results of the three methods on the representative dataset education on the same graph, as shown in Figure 4. From the figure we can see that, the performance of the three algorithms are nearly the saame. However, Gini Index is a little bit better than Mutual Information and gain ratio, while it has the lowest test error at all max_depths. That means the decision tree with Gini Index method has a better generalization ability.

What more, given that all attributes are binary, so we can **NOT** compare the preference of mutual information and gain ratio for the number of possible values.

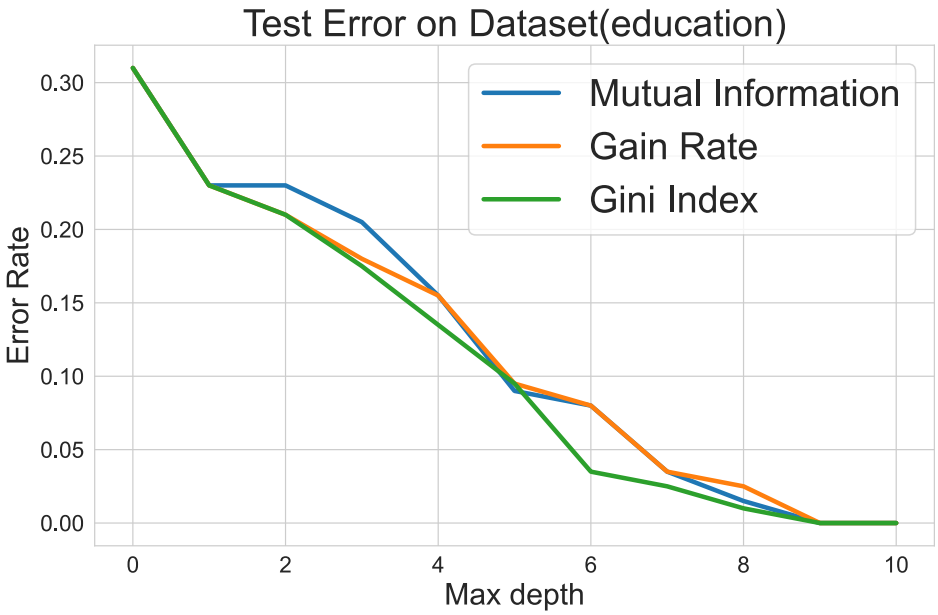

Figure 4: Error Rate of the three methods on the testing set

5 Something to say...

I have benefited a lot from completing this assignment. To be fair, the difficulty of this assignment is not very high, so it tests my abilities in detail handling, experimental design, and report writing.

In the code writing part, I spent a total of 2 days to complete all the code writing. What I am very satisfied with is that in this code writing, I did **NOT** refer to any materials (such as CSDN, Chat-GPT, Newbing, etc.), and I wrote all the code 100% according to my own ideas. However, the downside is that I may have written some functions more complicated and produced redundant code. Next, I will further modify my code to make it look more "Pythonic".

In the experimental design part, there are not many experiments that can be done in this assignment because the only parameter is max_depth. So, in order to make more comparisons and discussions, I did additional research on the selecting features part, that is, adding gain ratio and Gini Index methods. This made the experiment a bit richer. Of course, we also learned some "pruning" methods in class to deal with overfitting problems. I will also explore this aspect in the future.

In the report writing part, in order to fit better with the assignment document, I chose to write in English. In this report, I used **Google Translate** to look up some unfamiliar words and **Grammarly** to check the grammar of some paragraphs.

In the process of completing this assignment, I also gained a deeper understanding of decision trees. The combination of theoretical learning and practice has allowed me

to go further on the path of machine learning. I would also like to thank my classmates **Yuanfang Zhang**(张远方) and **Yuxin Huang**(黄昱欣) for discussing some aspects of this assignment with me, such as the specific submission form, report writing format, and experimental design methods. These positive discussions have helped me to complete this assignment better.

Regarding this semester's course, I am very grateful to Yuhang Guo and Kan Li teachers for their generous teaching. Teacher Guo's meticulous preparation and explanation have made me more interested in machine learning and confident in learning this subject well. **I wish both professors all the best in their future teaching and research careers!**

References

[1] 100074302 INTRODUCTION TO MACHINE LEARNING. *HOMEWORK-DECISION TREES*. 2023.

[2] Roger J Lewis. "An introduction to classification and regression tree (CART) analysis". In: *Annual meeting of the society for academic emergency medicine in San Francisco, California*. Vol. 14. Citeseer. 2000.

[3] J Ross Quinlan. "Program for machine learning". In: *C4. 5* (1993).

[4] 周志华. "《机器学习》". In: 中国民商 03.No.21 (2016), pp. 93–93.

A ParaTest.py

```
import pandas as pd
import decisionTree as dt
import math
import sys
import matplotlib.pyplot as plt
import seaborn as sns

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 ParaTest.py <dataset>")
        sys.exit(0)

    dataset = sys.argv[1]
    train_input_file = dataset + "_train.tsv"
    test_input_file = dataset + "_test.tsv"
    train_out = "train.labels"
    test_out = "test.labels"
    metrics_out = "test_metrics.txt"

    list_depth = [i for i in range(11)]
    list_train_err, list_test_err = [], []

    for max_depth in range(11):
        # 从输入文件得到训练集、测试集的 list, 每个属性的两个值
        train_input, test_input, attr_name, label_name, attr_title =\
            dt.Read_Data(train_input_file, test_input_file)
        # 生成决策树
        Tree = dt.Gen_Tree(train_input, max_depth,
                           attr_name, label_name, attr_title)
        # 根据生成的决策树预测训练集并输出到文件
        error_train = dt.Prediction(Tree, train_input, train_out)
        # 根据生成的决策树预测测试集并输出到文件
        error_test = dt.Prediction(Tree, test_input, test_out)
        list_train_err.append(error_train)
```

```
    list_test_err.append(error_test)

# 利用 seaborn 创建图标
sns.set_style('whitegrid')
fig, ax = plt.subplots(figsize=(10, 6))

# 绘制折线图
sns.lineplot(x=list_depth, y=list_train_err, label='Train Error',
             color='#10454F', linewidth=2, ax=ax)
sns.lineplot(x=list_depth, y=list_test_err, label='Test Error',
             color='#A3AB78', linewidth=2, ax=ax)

# 修改坐标等
title = "Error Rate on Dataset("+dataset+")"
ax.set_title(title, fontsize=16)
ax.set_xlabel("Max depth", fontsize=14)
ax.set_ylabel("Error Rate", fontsize=14)
ax.tick_params(labelsize=12)
ax.legend(fontsize=12)

figname = dataset + "_Gini.png"
plt.savefig(figname, dpi=300)
```