MACHINE LEARNING HOMEWORK

黄昱欣
1120200807
School of Computer Science and Technology
Beijing Institute of Technology

Contents

1 Report Structure **2**

2 The Datasets and Tasks **2**

2.1 Datasets 2

2.2 Tasks 2

3 Implementation **3**

3.1 Environment 3

3.2 inspection.py 4

3.3 decisionTree.py(ID3) 6

3.4 decisionTree_C4.5.py(C4.5) 9

3.5 decisionTree_CART.py(CART) 11

3.6 decisionTree_bagging.py(Bagging) 13

4 Experiment **15**

4.1 Parameter experiment on ID3 15

4.2 Performance comparison experiment 17

4.3 Discussion 18

5 Summary **19**

A Structure of the submission folder **21**

B Parameter experiment on other models **22**

1 Report Structure

In this section, I will introduce the structure of this report. In Section 2, I will explain the content of this assignment and the datasets I used, as well as the main function of each code. In Section 3, I will explain these codes explicitly. In Section 4, I will run some end-to-end experiments on three tasks (predicting the party of a politician/ predicting final grade for high school students/ predicting whether a mushroom is poisonous) and report my results. In Section 5, I will summarize the whole homework. In appendix A, you can see the structure of the submission folder, which contains 16 output files and 6 python programs. In appendix B, you can see the result of parameter experiment on models, which use different methods to choose feature to be spilt on.

2 The Datasets and Tasks

2.1 Datasets

There is four datasets. Each one contains attributes and labels and is already split into training and testing data. The first line of each .tsv file contains the name of each attribute, and the class is always the last column. All attributes are discretized into just two categories.

2.2 Tasks

In this Assignment, my goal is to implement a binary classifier, entirely from scratch.

Although written in the Machine_Learning_Homework.pdf, I only have to write two program, I have extended the task to five classifiers based on this two. Overall, I write six programs in this task:

inspection.py uses the majority vote strategy(picking the label with the most examples). It reads data from input file, calculates the label entropy at the root (i.e. the entropy of the labels before any splits) and the error rate (the percent of incorrectly classified instances) of classifying, and outputs them into the output file. Besides, we should use the following command to run the program.

```
python3 inspection.py <input> <output>
```

decisionTree.py is a Decision Tree learner which uses mutual information to determine which attribute to split on. If attributes have equal values for mutual information,

it will split on the first attribute. It learns a decision tree with a specified maximum depth and can print the decision tree in a specified format. At each leaf, it uses a majority vote to make classification decisions. If the vote is tied, it will choose the label that comes last in the lexicographical order. Also, it outputs the prediction of the training and testing sets in <train out> and <test out>, training and testing errors in <metrics out>. We should use the following command to run the program.

```
python3 decisionTree.py <train input> <test input> <
max depth> <train out> <test out> <metrics out>
```

decisionTree_C4.5.py is a Decision Tree learner which uses gain ratio to determine which attribute to split on. The strategy of choosing new feature is the only difference between decisionTree_C4.5.py and decisionTree.py.

decisionTree_CART.py is a Decision Tree learner which uses Gini index to determine which attribute to split on.

decisionTree_bagging.py is a Decision Tree learner which uses bootstrap aggregating methods(bagging). Bagging is an ensemble method that involves training multiple models on different subsamples of the training datasets, which is randomly selected with replacement, and combining their predictions. In this program, it aggregates the three individual tree in programs above, and make the final prediction by majority voting.

t.py is used to analyze the performance in each decision tree. Firstly, we use the maximum depth as a parameter, so we can analyze different error rate on different maximum depth. Secondly, we have four different classifiers, we can compare the error rate between them.

3 Implementation

3.1 Environment

Configuration Item	Specifications
Processor	AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
Memory	16GB RAM
Operating System	Windows 10
Programming Language	Python 3.9
IDE	PyCharm Community Edition
Packages	numpy 1.23.3, pandas 1.5.0, matplotlib 3.6.0, matplotx 0.3.10

3.2 inspection.py

In this subsection, I will introduce the way I implement this program and the results. Compared to other program, this classifier is really simple.

3.2.1 Code Implementation

The first part is read filename from command line, I choose to read the dataset into a DataFrame, so that I can use its method to calculate the number of samples, labels and so on.

```
dataset = pd.read_csv(str(sys.argv[1]), sep='\t', header=0)
output = str(sys.argv[2])
```

Then, I compute the number of samples and the frequency of each label.

```
max_num = dataset.shape[0]
last_column = dataset.columns[-1]
num = dataset[last_column].value_counts().max()
p1 = num / max_num
p2 = 1 - p1
```

Afterwards, I compute entropy and error rate based on this formula:

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

where C_k represents the number of samples belonging to the k-th class.

```
entropy = - p1 * log(p1, 2) - p2 * log(p2, 2)
error = p2
```

In the last part, I output the entropy and error rate to the output file.

```
fp = open(output, "w")
print("entropy:", entropy, file=fp)
print("error:", error, file=fp)
fp.close()
```

3.2.2 Results

In PyCharm, we can use this method to modify command line:

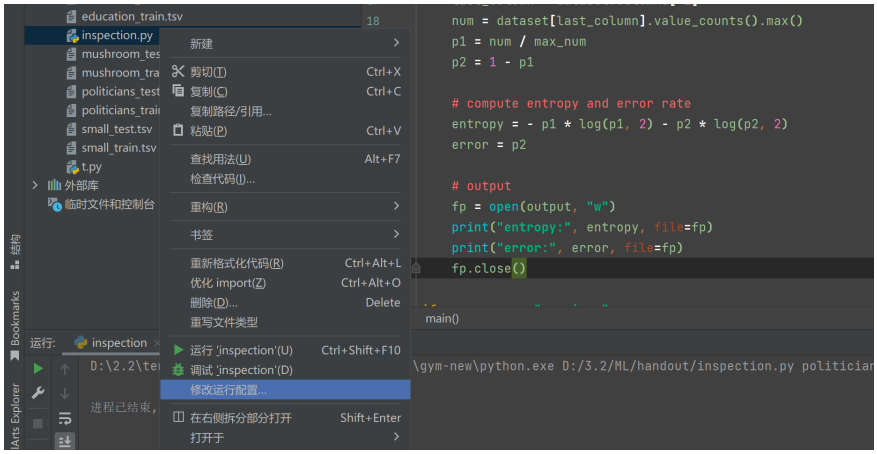

Figure 1: step 1

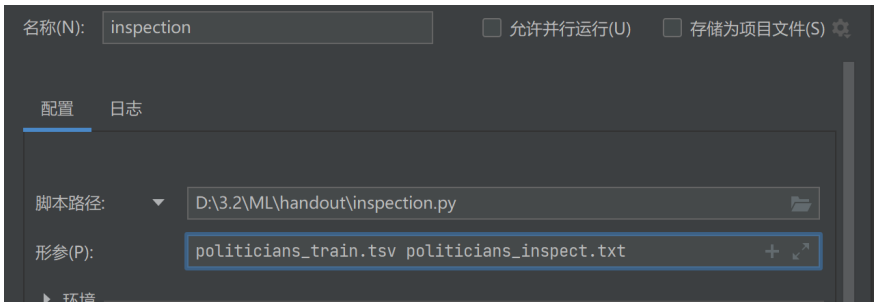

Figure 2: step 2

The output files of four datasets are as follows:

education_inspect.txt:

```
entropy: 0.909736122531
error: 0.325000000000
```

mushroom_inspect.txt:

```
entropy: 0.962614705998
error: 0.386666666667
```

politicians_inspect.txt

```
entropy: 0.990589428654
error: 0.442953020134
```

small_inspect.txt

```
entropy: 0.996316519559
error: 0.464285714286
```

3.3 decisionTree.py(ID3)

In this subsection, I will introduce how I implement this program and the results.

3.3.1 Code Implementation

After thoughtful consideration, I complete this program gradually.

Firstly, after reading the input, we should use the data to train a decision tree. Therefore, we need a data structure to store a decision tree. So, I create a class Node to store decision tree. Only the leaf node has label. If this node isn't a leaf node, then its label will be None, but we can reach its son node by feature and value.

```
class Node:
def __init__(self, feature=None, branch1=None, value1=None, branch2=None,
            value2=None, label=None):
    self.feature = feature
    self.value1 = value1
    self.value2 = value2
    self.branch1 = branch1
    self.branch2 = branch2
    self.label = label
```

Then, I implement the calculation of mutual information(Gain(D,A)).

$$
\begin{aligned}
H(D \mid A) &= \sum_{i=1}^{n} \frac{|D_i|}{|D|} H\left(D_i\right) \\
&= -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \left(\sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|} \right)
\end{aligned}
$$

$$
\operatorname{Gain}(D, A) = H(D) - H(D \mid A)
$$

```
def choose_feature(self, X, y):
# H(y)
y_entropy = self.calc_entropy(y)
mutual_information = []
# calculate H(y) - H(y|feature)
for feature in X.T:
    mutual_information.append(y_entropy - self.calc_conditional_entropy(feature,
            y))
# choose argmax
if all(i <= 0 for i in mutual_information):
    return -1
return np.argmax(mutual_information)
```

After this, I think of the function to train the decision tree. The pseudo code is as follows.

Algorithm 1: Building a decision tree node

```
Input  : current depth, dataset
Output: tree node
if over the depth then
    using majority vote;
    if no ties then
        return a leaf node
    else
        choose the label last in the lexicographical order;
        return a leaf node
if all sample belongs to one kind then
    return a leaf node
choose feature to spilt;
if no feature can be spilt then
    using majority vote;
    if no ties then
        return a leaf node
    else
        choose the label last in the lexicographical order;
        return a leaf node
else
    spilt data into two datasets;
    print( |* (cur_depth + 1));
    print(feature = value1: );
    branch1 = build(cur_depth + 1, dataset1);
    print( |* (cur_depth + 1));
    print(feature = value2: );
    branch2 = build(cur_depth + 1, dataset2);
    return node(branch1, value1, branch2, value2);
```

Above is the rough structure of the code. For the detailed code, please see the attachment.

3.3.2 Results

```
[ 135 A / 65 notA ]
| F = A : [ 119 A / 23 notA ]
| | M4 = A : [ 56 A / 2 notA ]
| | | P1 = A : [ 41 A / 0 notA ]
| | | P1 = notA : [ 15 A / 2 notA ]
| | M4 = notA : [ 63 A / 21 notA ]
| | | M2 = A : [ 37 A / 3 notA ]
| | | M2 = notA : [ 26 A / 18 notA ]
| F = notA : [ 16 A / 42 notA ]
| | M2 = A : [ 13 A / 15 notA ]
| | | M4 = A : [ 6 A / 1 notA ]
| | | M4 = notA : [ 7 A / 14 notA ]
| | M2 = notA : [ 3 A / 27 notA ]
| | | M4 = A : [ 3 A / 5 notA ]
| | | M4 = notA : [ 0 A / 22 notA ]
```

Figure 3: tree structure in education(depth=3)

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

Figure 4: tree structure in small(depth=2)

The output files are as follows(labels file can be checked in attachment):

edu_3_metrics.txt:

```
error(train):  0.17
error(test):  0.205
```

small_2_metrics.txt

```
error(train):  0.07142857142857142
error(test):  0.14285714285714285
```

3.4 decisionTree_C4.5.py(C4.5)

3.4.1 Code Implementation

The only difference between this program and last is the choose_feature function, because the gain ratio = mutual information/entropy of the feature:

$$\text{Gain}_{\text{ratio}}(D, A) = \frac{\text{Gain}(D, A)}{H_A(D)}$$

$$H_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}$$

```
def choose_feature(self, X, y):
# H(y)
y_entropy = self.calc_entropy(y)
gain_ratio = []

for feature in X.T:
    # (H(y) - H(y|feature)) / H(feature)
    conditional_entropy = self.calc_conditional_entropy(feature, y)
    if y_entropy - conditional_entropy == 0:
        gain_ratio.append(0)
    else:
        gain_ratio.append((y_entropy - conditional_entropy)/\
                self.calc_entropy(feature))

# choose argmax
if all(i <= 0 for i in gain_ratio):
    return -1
return np.argmax(gain_ratio)
```

3.4.2 Results

```
[ 135 A / 65 notA ]
| F = A : [ 119 A / 23 notA ]
| | M4 = A : [ 56 A / 2 notA ]
| | | P1 = A : [ 41 A / 0 notA ]
| | | P1 = notA : [ 15 A / 2 notA ]
| | M4 = notA : [ 63 A / 21 notA ]
| | | M2 = A : [ 37 A / 3 notA ]
| | | M2 = notA : [ 26 A / 18 notA ]
| F = notA : [ 16 A / 42 notA ]
| | M4 = A : [ 9 A / 6 notA ]
| | | M5 = A : [ 8 A / 1 notA ]
| | | M5 = notA : [ 1 A / 5 notA ]
| | M4 = notA : [ 7 A / 36 notA ]
| | | M2 = A : [ 7 A / 14 notA ]
| | | M2 = notA : [ 0 A / 22 notA ]
```

Figure 5: tree structure in education(depth=3)

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

Figure 6: tree structure in small(depth=2)

The output files of four datasets are as follows(labels file can be checked in attachment):

edu_3_metrics.txt:

```
error(train): 0.16
error(test): 0.18
```

small_2_metrics.txt

```
error(train): 0.07142857142857142
error(test): 0.14285714285714285
```

3.5 decisionTree_CART.py(CART)

3.5.1 Code Implementation

The only difference between this program and last is the choose_feature function, too. The formula of Gini index:

$$\begin{aligned}\operatorname{Gini}(D) &= \sum_{k=1}^{K} \frac{|C_k|}{|D|}\left(1-\frac{|C_k|}{|D|}\right) \\ &= 1-\sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2 \\ \operatorname{Gini}(D \mid A) &= \sum_{i=1}^{n} \frac{|D_i|}{|D|} \operatorname{Gini}\left(D_i\right)\end{aligned}$$

```
    def choose_feature(self, X, y):
        gini_index = []
        for feature in X.T:
            gini_index.append(self.calc_gini_index(feature, y))
        return np.argmin(gini_index)

    def calc_gini_index(self, feature, label):
        values = np.unique(feature)
        gini_index = 0.0
        for value in values:
            value_indices = np.where(feature == value)
            value_labels = label[value_indices]
            # gini(value) = 1 - (p)^2
            gini_for_value = 1.0
            unique_labels = np.unique(value_labels)
            for i in unique_labels:
                prob = np.count_nonzero(value_labels == i) / len(value_labels)
                gini_for_value -= prob ** 2
            # gini(y|feature) += p_value * gini(value)
            gini_index += len(value_labels) / len(feature) * gini_for_value
        return gini_index
```

3.5.2 Results

```
[ 135 A / 65 notA ]
| F = A : [ 119 A / 23 notA ]
| | M3 = A : [ 95 A / 10 notA ]
| | | M5 = A : [ 79 A / 2 notA ]
| | | M5 = notA : [ 16 A / 8 notA ]
| | M3 = notA : [ 24 A / 13 notA ]
| | | M1 = A : [ 15 A / 2 notA ]
| | | M1 = notA : [ 9 A / 11 notA ]
| F = notA : [ 16 A / 42 notA ]
| | M4 = A : [ 9 A / 6 notA ]
| | | M5 = A : [ 8 A / 1 notA ]
| | | M5 = notA : [ 1 A / 5 notA ]
| | M4 = notA : [ 7 A / 36 notA ]
| | | M2 = A : [ 7 A / 14 notA ]
| | | M2 = notA : [ 0 A / 22 notA ]
```

Figure 7: tree structure in education(depth=3)

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

Figure 8: tree structure in small(depth=2)

The output files of four datasets are as follows(labels file can be checked in attachment):

edu_3_metrics.txt:

```
error(train):  0.15
error(test):  0.175
```

small_2_metrics.txt

```
error(train):  0.07142857142857142
error(test):  0.14285714285714285
```

3.6 decisionTree_bagging.py(Bagging)

3.6.1 Code Implementation

Bagging, which stands for Bootstrap Aggregating, is a technique used to improve the performance of machine learning models. Bagging on decision trees is done by creating bootstrap samples from the training data set and then built trees on bootstrap samples and then aggregating the output from all the trees and using majority voting to predicting the output.

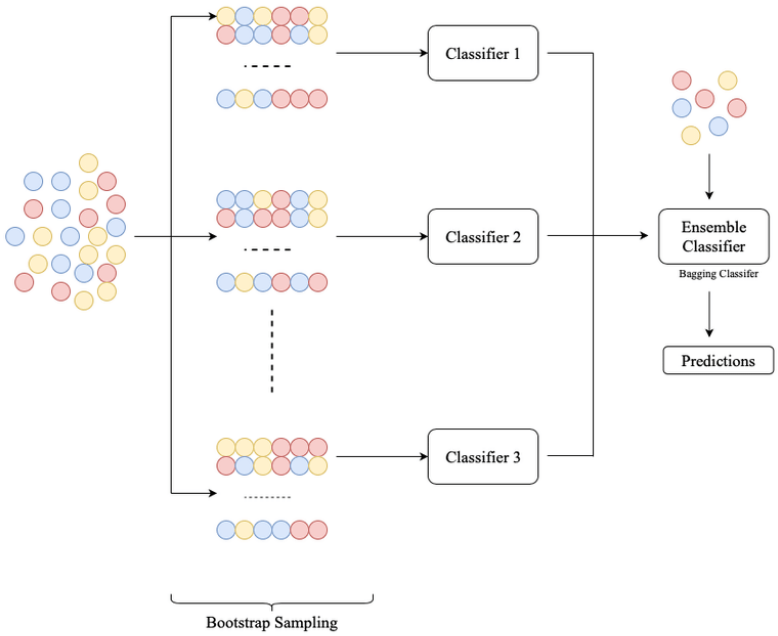

Figure 9: bagging

In my program, I combine the three classifiers above into a bagging classifier. It will separately train three decision tree, and in the test function uses majority vote to choose the label.

```
print('------------------tree_mutual--------------------')
tree_1 = decisionTree(train_data, max_depth, 0)
print('-------------------tree_c4.5---------------------')
tree_2 = decisionTree(train_data, max_depth, 1)
print('-------------------tree_cart---------------------')
tree_3 = decisionTree(train_data, max_depth, 2)
error_train = test(tree_1, tree_2, tree_3, train_data)
error_test = test(tree_1, tree_2, tree_3, test_data)
```

3.6.2 Results

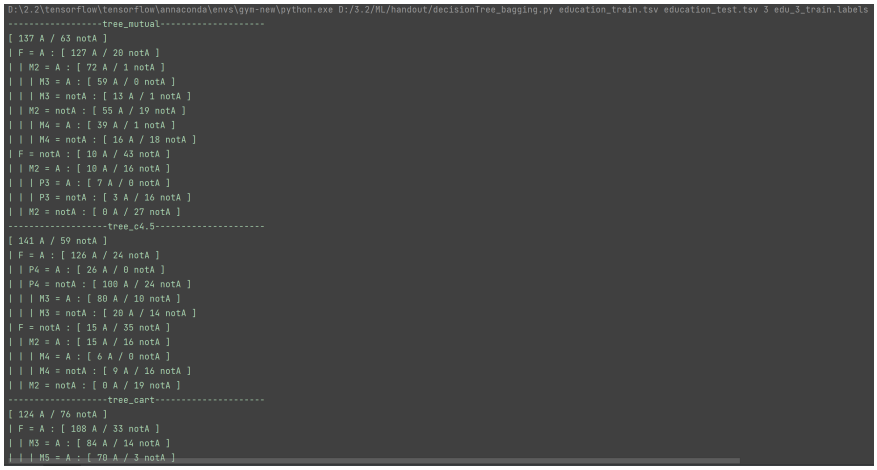

Figure 10: tree structure in education(depth=3)

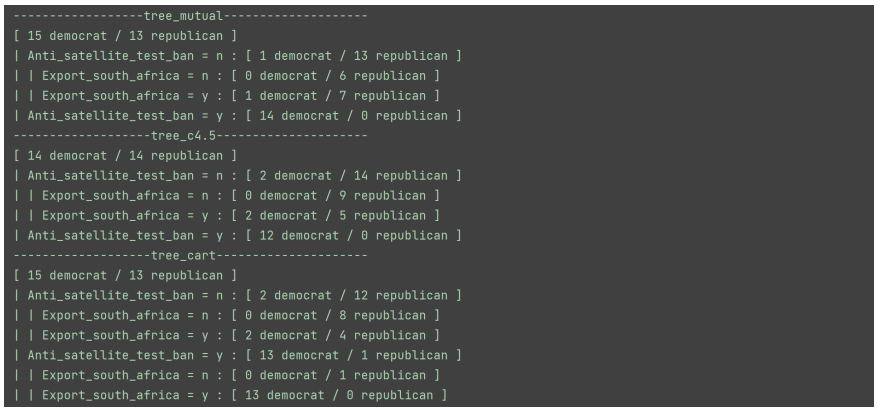

Figure 11: tree structure in small(depth=2)

The output files of four datasets are as follows(labels file can be checked in attachment):

edu_3_metrics.txt:

```
error(train):  0.15
error(test):  0.175
```

small_2_metrics.txt

```
error(train):  0.10714285714285714
error(test):  0.14285714285714285
```

4 Experiment

In this section, I will conduct parameter experiments for each dataset and compare the performance of different methods.

4.1 Parameter experiment on ID3

For each dataset, I will choose the max depth that minimizes the test error rate and provide the output files corresponding to these max depths in the submitted folder. If the test error rate is equal, then I will choose the lower max depth.

In the **politicians** dataset, when max depth equals 2, the error rate reaches its minimum.

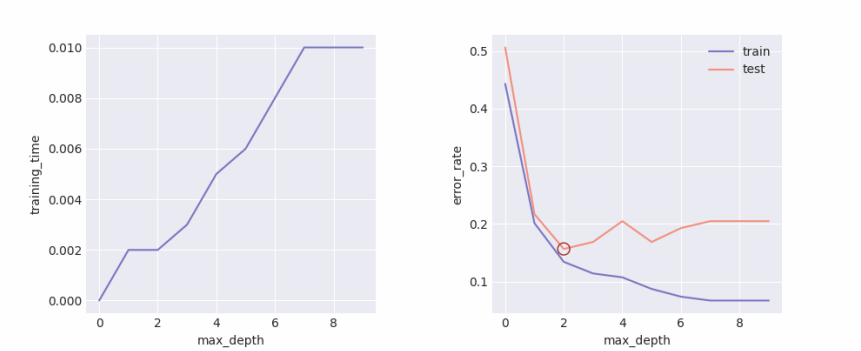

Figure 12: error rate in politicians dataset using ID3

```
politicians_2_metrics.txt
error(train):  0.1342281879194631
error(test):  0.1566265060240964
```

In the **education** dataset, when max depth equals 9, the error rate reaches its minimum.

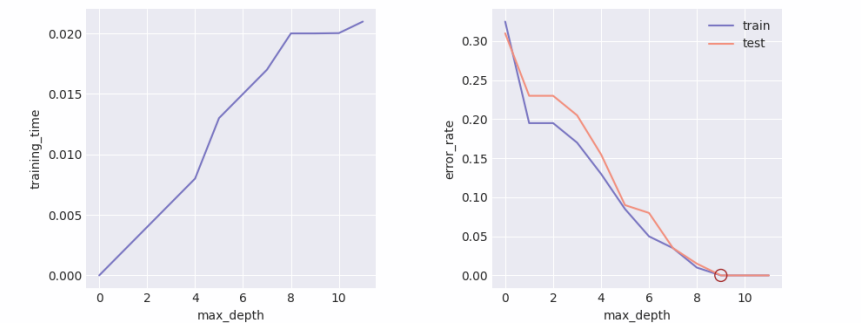

Figure 13: error rate in education dataset using ID3

```
education_9_metrics.txt
error(train):  0.0
error(test):  0.0
```

In the **mushroom** dataset, when max depth equals 9, the error rate reaches its minimum.

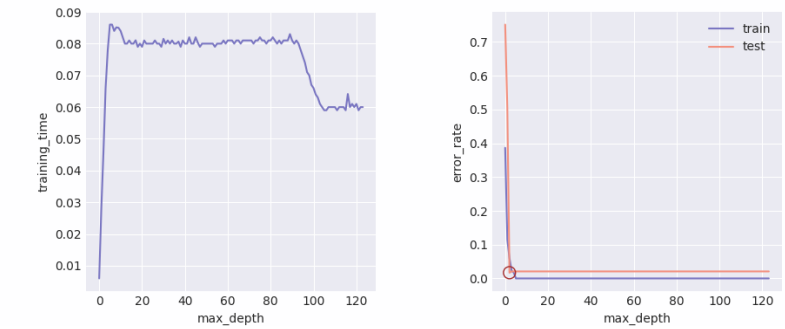

Figure 14: error rate in mushroom dataset using ID3

```
mushroom_2_metrics.txt
error(train):  0.06
error(test):  0.01694915254237288
```

In the **small** dataset, when max depth equals 1, the error rate reaches its minimum.

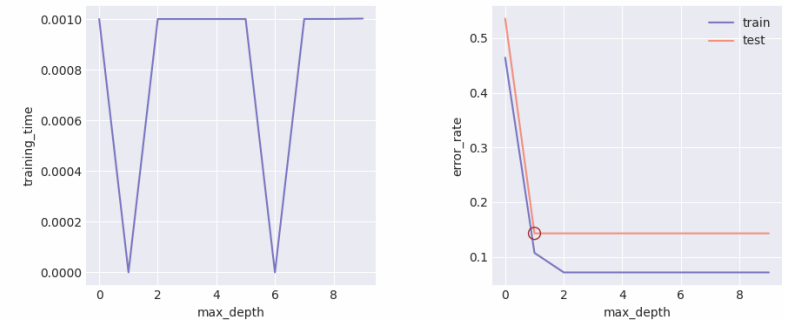

Figure 15: error rate in small dataset using ID3

```
small_1_metrics.txt
error(train):  0.10714285714285714
error(test):  0.14285714285714285
```

In terms of time, although I do make comparisons, in reality, due to the fact that the datasets are not large, the training time is within an acceptable range, so I will not make further time comparisons.

The parameter experiment results of other models are in the appendix B.

4.2 Performance comparison experiment

From the above comparison, the optimal max depth for different models are roughly the same. In this section, we will compare the performance of each model in detail.

I originally thought bagging would have the best performance, but in reality, on the first dataset **politicians**, it can be seen in figure 16 that due to the randomness of the training set selection and the fact that only three classifiers were integrated in this experiment, the performance is not as good as expected. Among the other three classifiers, it can be seen that CART performs the best.

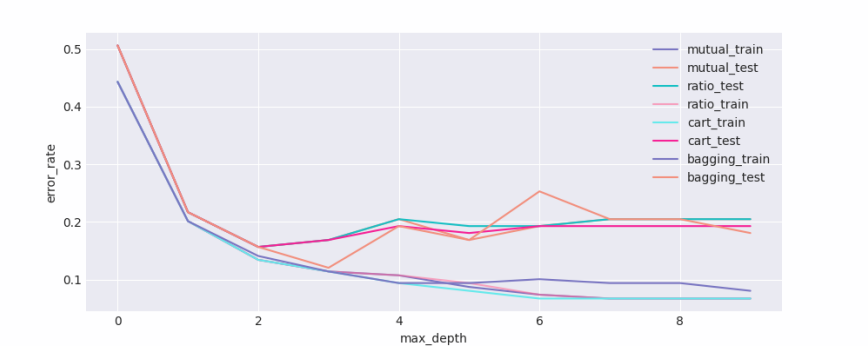

Figure 16: performance comparison in politicians dataset

In the **education** dataset, the training and testing rates achieve 0 when the max depth reaches 9. However, bagging still remains the worst-performing model among them, and its error rate does not even reach 0 when the max depth is 9.

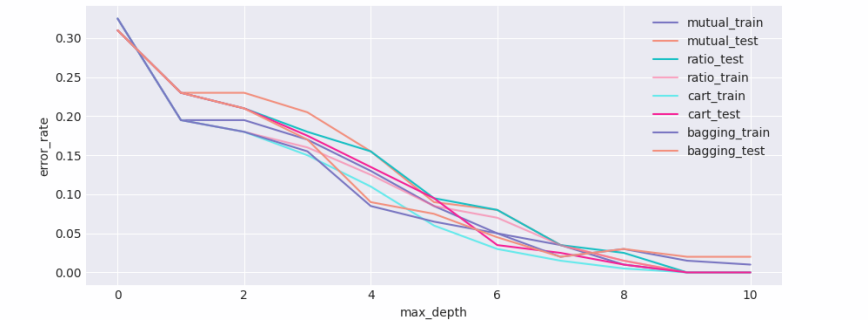

Figure 17: performance comparison in education dataset

In the **mushroom** dataset, since it is observed that the training results of all models do not change significantly after a max depth of 8, I only compare the parts where max depth varies from 0 to 8. It can be seen that all training error rates have

been reduced to 0. In terms of test error rate, bagging and were the lowest, while the three models show similar performance.

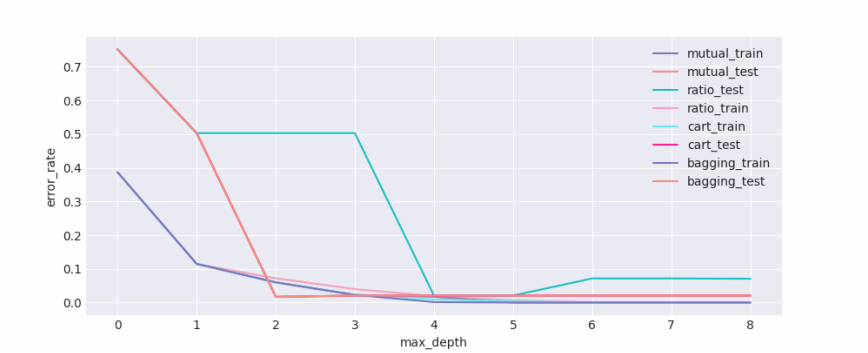

Figure 18: performance comparison in mushroom dataset

Due to the small size of the small dataset, the four models have almost identical performance.

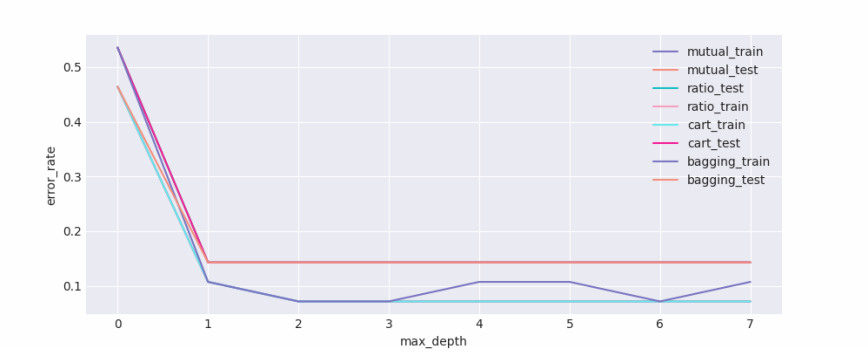

Figure 19: performance comparison in small dataset

4.3 Discussion

I think one of the reasons for the unclear difference in this experiment is that the dataset is too small, and the performance of each method is actually very close. Therefore, I cannot analyze time and accuracy in more detail. Another issue is that the performance of the bagging method is poor: the main reason is that only three methods were integrated in the bagging method. We can improve the accuracy of bagging by adding weights or integrating more methods.

5 Summary

Although the requirements for this homework were not many, after thinking, I decided to increase the workload on my own. After completion, I gained a lot.

Firstly, I had a deeper understanding of decision tree concepts, which is more than understanding the definitions. It helped me understand the training and testing process of decision trees better. Besides, conducting parameter and comparative experiments gave me a greater understanding of different decision tree methods.

Secondly, I developed my engineering skills. Although the amount of code was not very large, it still allowed me to improve my proficiency in using the Pandas and Numpy packages in Python. I can integrate different programs to launch experiment, and this experience was valuable for me.

Thirdly, I improved my ability to use English. Since I don't have many opportunities to write academic reports in English, using it in my report was a bit difficult at first. However, this experience helped me become more fluent in using academic vocabulary and sentence structures.

Regarding ChatGPT, I also sought its help during the report writing process. When I encountered errors or warnings that I didn't understand in Python or LaTeX, I would ask for its explanation. As a "private tutor," he was able to provide more accurate and precise solutions than if I searched for error messages online.

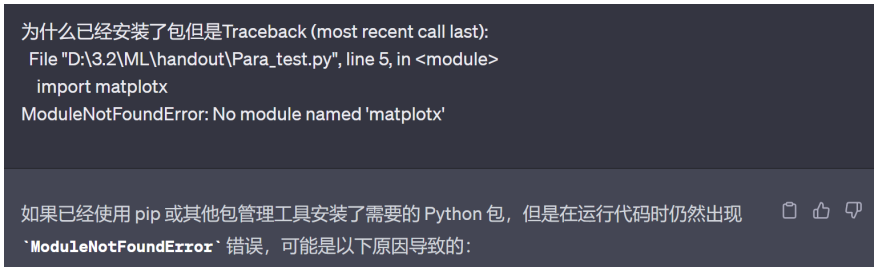

Figure 20: explain error message

Additionally, I asked him about some data structure questions in Python, such as member functions of a DataFrame object, so that I could write more concise code and focus on the core code instead of using many "for" and "while" loops to implement simple functions that could be solved by the built-in member functions.

Figure 21: questions about data structure

In summary, ChatGPT was like a small teacher for me. I didn't rely on his creativity, but rather his vast knowledge to increase my efficiency.

A Structure of the submission folder

```
/
├── 1120200807_ 黄昱欣 _report.pdf
├── education_9_metrics.txt
├── education_9_test.labels
├── education_9_train.labels
├── education_inspect.txt
├── mushroom_2_metrics.txt
├── mushroom_2_test.labels
├── mushroom_2_train.labels
├── mushroom_inspect.txt
├── politicians_2_metrics.txt
├── politicians_2_test.labels
├── politicians_2_train.labels
├── politicians_inspect.txt
├── small_1_metrics.txt
├── small_1_test.labels
├── small_1_train.labels
├── small_1_inspect.txt
├── decisionTree.py
├── decisionTree_bagging.py
├── decisionTree_C4.5.py
├── decisionTree_CART.py
├── inspection.py
└── t.py
```

B Parameter experiment on other models

C4.5:

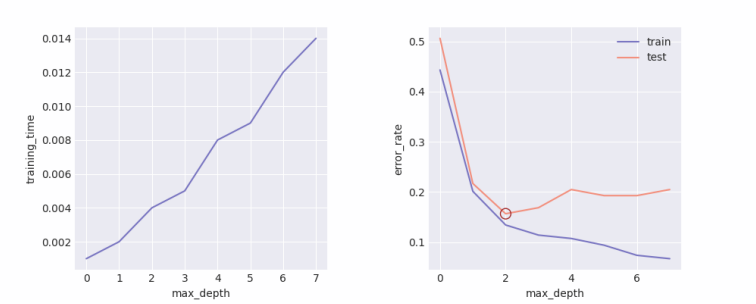

Figure 22: error rate in politicians dataset using C4.5

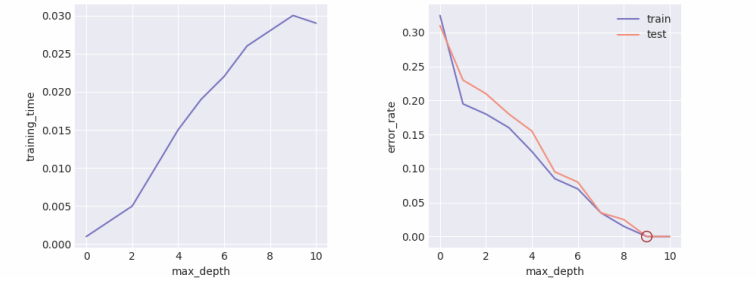

Figure 23: error rate in education dataset using C4.5

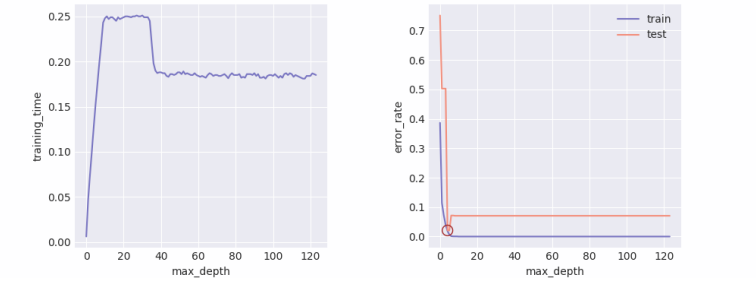

Figure 24: error rate in mushroom dataset using C4.5

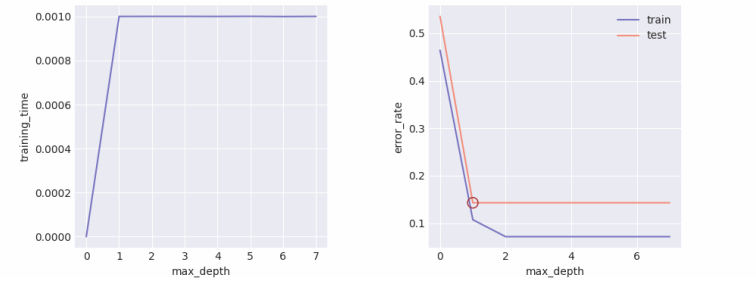

Figure 25: error rate in small dataset using C4.5

CART:

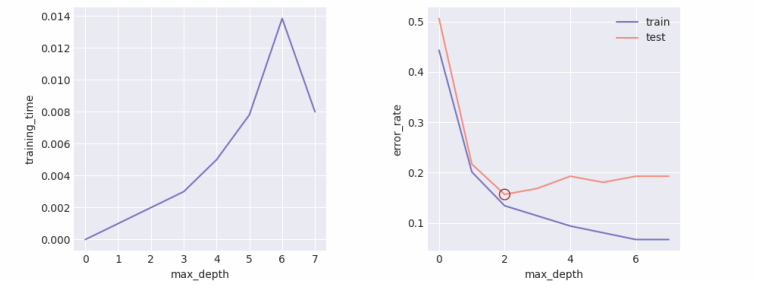

Figure 26: error rate in politicians dataset using CART

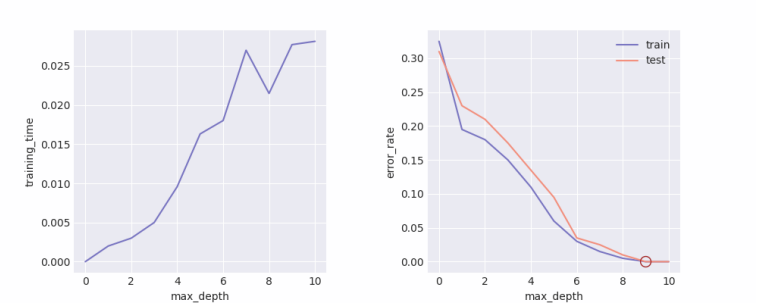

Figure 27: error rate in education dataset using CART

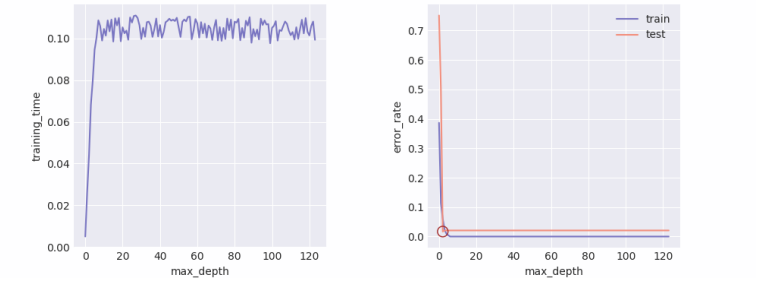

Figure 28: error rate in mushroom dataset using CART

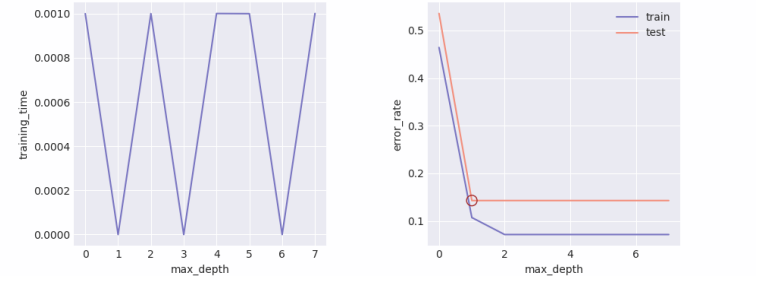

Figure 29: error rate in small dataset using CART

bagging:

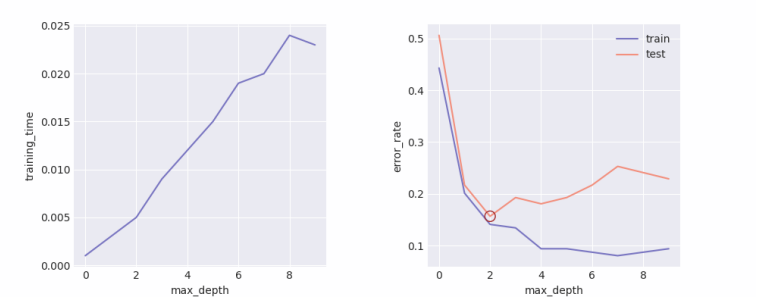

Figure 30: error rate in politicians dataset using bagging

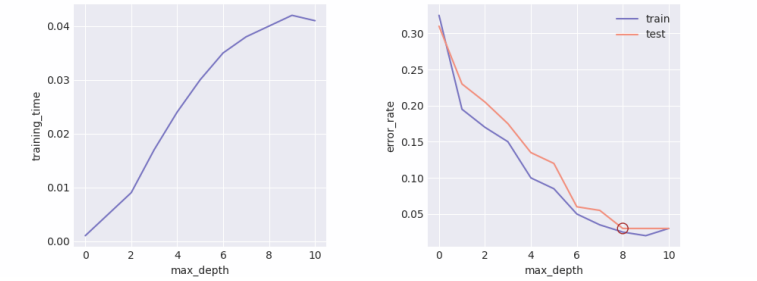

Figure 31: error rate in education dataset using bagging

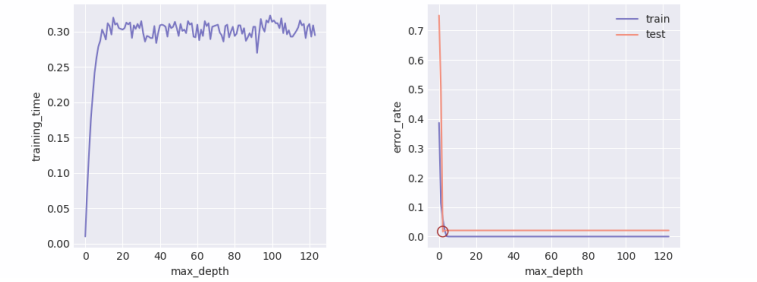

Figure 32: error rate in mushroom dataset using bagging

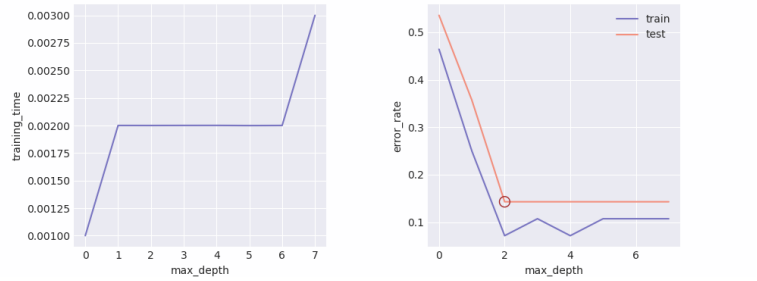

Figure 33: error rate in small dataset using bagging