CMSC 5724 Project

Decision Tree

by

XU Shenghao 1155148147

CUI Ruize 1155154957

HUANG Shukang 1155145663

Chen Zheng 1155152566

DING Xinyun 1155154895

Contribution Declaration:

In this project, each team member's contribution is **20 percent** .

Nov.2020

Contents

Chapter 1

Introduction

1.1 Hunt's Algorithm and Hunt's Algorithm(modified)

The **Classification problem** is a universal problem and it is also the basis of other more complex decision-making problems. The essence of the classification problem is that when a data set is given, we are required to train a model that can predict an object with a set of new feature factors that should belong to which category.

There are a lot of algorithms we can use to solve this problem in machine learning and data mining and we have learned many algorithms in the class such as **SVM**,**K-means**, **Naive Bayes**, etc. In this project, we use Decision Tree Classifier to predict the income of the adult in America. Decision Tree Classifier is a simple and widely used classification technique. It applies a very useful way to make a decision. When we have built a decision tree model, we can test the objects from the root node to the leaf node and get the final classification result.

In general, because of the exponential size of the search space, it's computationally infeasible to find the optimal tree. Therefore, some algorithms use local optimal strategies to build a decision tree such as $CART$, $ID3$, $C4.5$. In this project, we choose **Hunt's algorithm** , which is discussed in the class.

In Hunt's algorithm, a decision tree is built recursively by dividing the training set into purer subsets. There are two recursive procedures in Hunt's algorithm, the first one is if all the objects in the training set have the same label, we can get a leaf node with the value of this label. the other one is if all the objects in the training set have the value of the same attribute, it's impossible to divide these objects, so we can get a leaf node whose label is the majority one in the training set. However, if the training set contains objects that belong to more than one label, we need to split the data into smaller subsets. There are countless choices to split the data, we need to get a better purity of the node so that we can have a better split way to get class distribution. The measurements of node purity are **Gini index**,**Entropy**, **Information gate**. we choose to use the **Gini index** in our project because this measurement is taught in the class and all of us are familiar with *Gini*. The smaller *Gini* is, the better the split quality. In this way, we can finally build a decision tree.

However, because of the **Generalization Theorem**, if we go deeper and make the subset too small, the *subtree* will become unreliable and overfitting will occur. To avoid this situation, we set a parameter called **'small_factor'** which can show how small the subset is, and by comparing the accuracy of the model, we can split the data-set more appropriately. This is the modified version of **Hunt's algorithm** .

Chapter 2

Build of Decision Tree

Our project is divided into two main parts:

1. Data Prepossessing

2. Model training and validation

2.1 Data Prepossessing

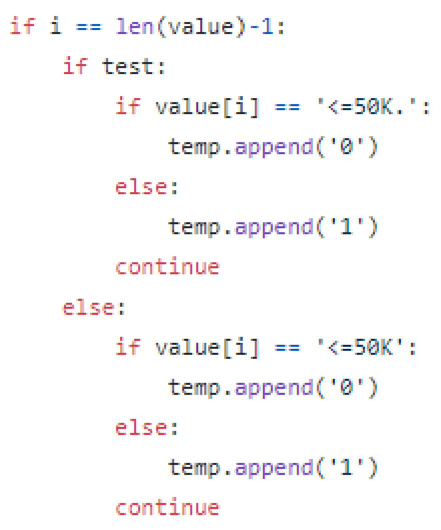
The data set has a total of 13 attributes and includes a label (14th-index).

First of all, we create a function called **clean_data**().**f.readlines()** is used to read the data in the data set line by line.Also, **data**=[ ] and **temp**[ ] are created for storing value in each lines.

We enter the training set and the test set into **clean_data**(). Then we use **line.strip()** to eliminate spaces at the beginning and end. After that,slicing strings by specifying delimiters(', ').

In our project,**for-loop** is used to iterate through each split. Due to we need remove the attribute "native-country",and "native-country" is the second last index of each line of value.When iterating to **len(value)-2** ("native-country"), we skip this attribute and do not record it in the **temp**[ ].The above method successfully removes the attribute "native-country".

As shown in Figure 1.1 belowWe change the labels $<= 50k$ and $> 50k$ to 0 and 1 respectively.Since the data format of the training set and the test set are different, we traverse both sides to filter the data.



```
if i == len(value)-1:
    if test:
        if value[i] == '<=50K.':
            temp.append('0')
        else:
            temp.append('1')
        continue
    else:
        if value[i] == '<=50K':
            temp.append('0')
        else:
            temp.append('1')
        continue
```

Figure 2.1: label

In the dataset, there are the following kinds of ordinal attributes: **age**,**fnlwgt**,**education_-num**,**capital_gain**,**capital_loss**,**hours_per_week**. Total 6 continuous attributes. So, we convert those attributes into **int** format and append in to **temp**[ ].

In the last,we remove all the records containing **'?'** (i.e., missing values).We use the **if conditional** to traverse the data in **temp**[ ], if there exist **'?'** in the data which store in the **temp**[ ], we will skip this line of data and append rest data into **data=**[ ].

In the end, the **data=**[ ] contains all valid data after cleaning. Then, the function **clean_data**() will return the valid data. To sum up, we have completed the data cleaning work and obtained the required valid data.

There are **32,562** lines of data in the test set and **16,283** lines of data in the training set (including empty lines and comment lines), and after passing our cleaning process, the training set contains **30,718** lines of data and the test set contains **15,315** lines of data.

2.2 Model training and validation

2.2.1 Class Diagram

In this part, we will use the **class diagram** ,shown in Figure 2.2, to describe the classes contained in the system and the relationship between them.

Firstly, we instantiate the class, as we all know, a class represent a concept which encapsulates **state** (*attributes*) and **behavior**(*operations*), so you can see the attributes

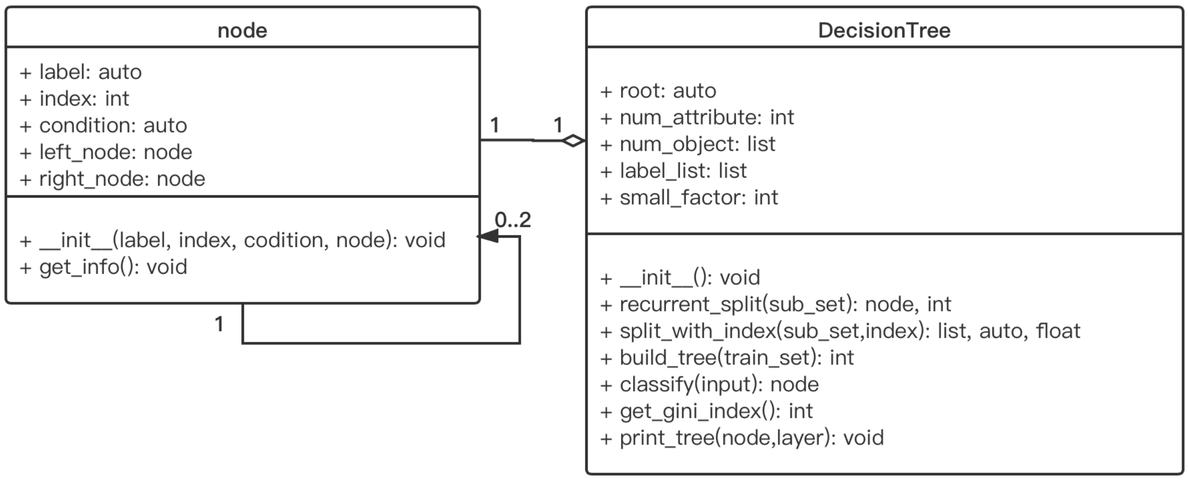


Figure 2.2: class diagram

of the node class are shown in the second partition and the operations appear in the third partition and operations map onto class methods in code.

In the node class, we package $left_node$ and $right_node$ which can help to initialize the decision tree. In the $DecisionTree$ class, we can use the $Gini$ index which we calculate by the subsets of data to train the tree and return with the error.

2.2.2 Methods for Expressing Test Conditions

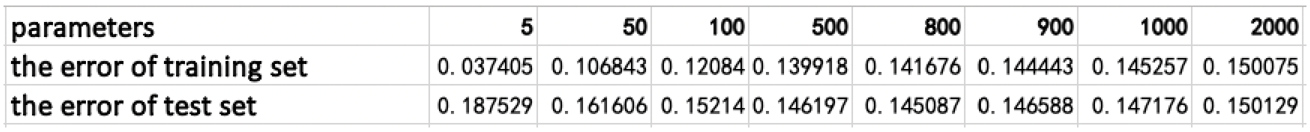
The methods we choose for expressing test conditions depend on attribute types and the number of ways to split.

In the original dataset, there are two different kinds of attribute types, such as *continuous* and *discrete*. And we use a $2 - way$ split to divide values into two subsets. Therefore, as for continuous attributes, we convert the string type to integer and consider all possible splits and find the best cut, for example, like ($A \geq V$ or $A < V$). As for discrete attributes, we divide all values into two-part(one meets the conditions while the other not), in this way, we can also find the **best split**.

2.2.3 Enhanced Performance

In order to get a better performance or increase the accuracy, we promote a function called **small_factor**

As know from the **Generalization Theorem**, $generalizationerror$ of classifier h is



parameters	5	50	100	500	800	900	1000	2000
the error of training set	0. 037405	0. 106843	0. 12084	0. 139918	0. 141676	0. 144443	0. 145257	0. 150075
the error of test set	0. 187529	0. 161606	0. 15214	0. 146197	0. 145087	0. 146588	0. 147176	0. 150129

Figure 2.3: error rate compare with the parameters

the sum of $empiricalerror$ of h and $\sqrt{\frac{ln(\frac{1}{\delta})+ln|H|}{2|S|}}$. we observe two summands, they are monotonically decreasing and monotonically increasing in H, respectively.However, these two summands do not contribute the same proportion to the $generalizationerror$. In another word, if I adjust the **small_factor**, it is possible that result grew in $\sqrt{\frac{ln(\frac{1}{\delta})+ln|H|}{2|S|}}$ compared to the previous adoption variables.But, it also result the decrease in $empiricalerror$,it may result in a situation where the reduction in error rate caused by changing **small_factor** is greater than the increase in error rate caused by changing **small_factor**.the sum of this two factors may further reduce the value of generalization error. So, this is kind of trad-off between $empiricalerror$ and $\sqrt{\frac{ln(\frac{1}{\delta})+ln|H|}{2|S|}}$. And, we want to find the balance between two factor.

In the class $DecisionTree$ node, we package an attribute called **small_factor** which shows the size of the subsets. As we know, if we split the training set into a very small subset, it will $overfit$, so we test the decision tree by adjusting this parameter. And we can see when we set the **small_factor** as 800, we can avoid the over-fitting.The experiment result is shown in Figure 2.3.