**2012003385 Jin Hyun Kim**

**<Programming Assignment #2>**

**1. Algorithm Summary & Envrionment**

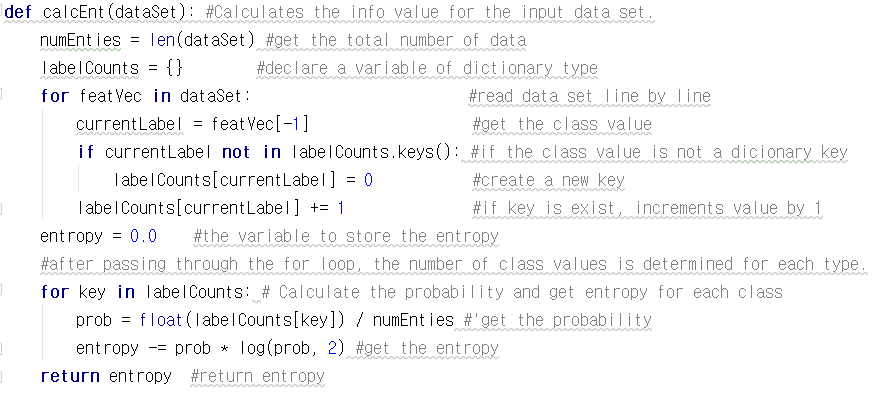
I use python 2.7 and I worked on JetBrains PyCharm Community Edition 2017.1.1 with Windows 10 64bit. I used one python file named ‘dt.py’ consists of a main function and the other functions and It was based on the gain ration algorithm.

makeDecisionTree : It takes a train.txt to create a decision tree and calls the createTree function.

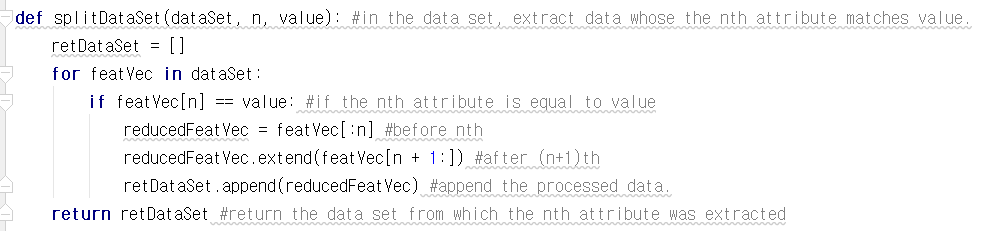
createTree : A function that actually creates a decision tree. DataSet and dataLabel are input, and a decision tree is created with help from three functions. In ‘chooseBestFeatureToSplit’, calculate the info value for each attribute using the entropy calculation function ‘calcEnt’ and the ‘splitDataSet’ which divides the dataSet. When the best feature is determined, the tree is completed using recursion with the remaining data except the best feature.

makeOutputFromTest, decideClassValue : Generate result text file through decision tree and test text file. Recursively search the decision tree defined by dictionary type and set the class value for each data and write to ouput file.

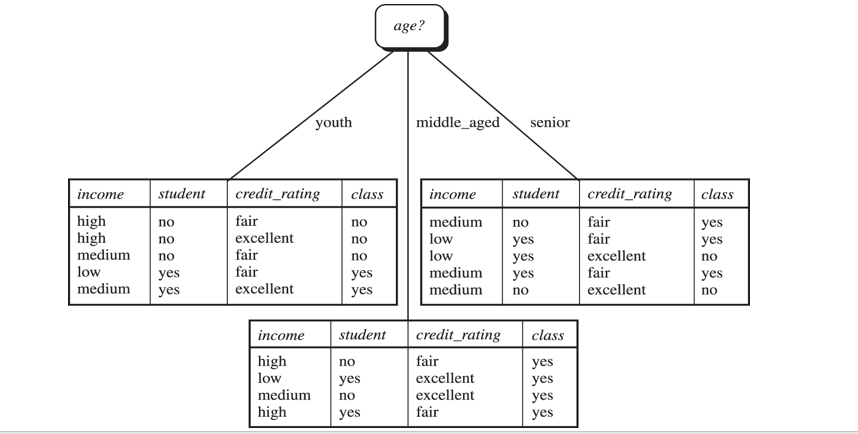
**2. Detailed description of codes**

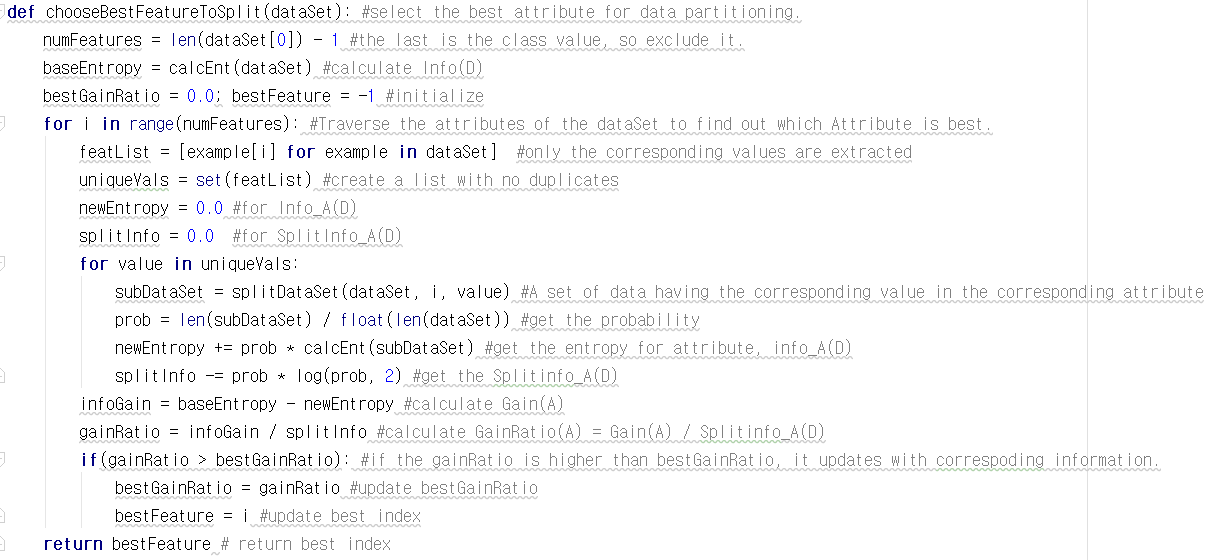


‘caclEnt’ function calculates the entropy value for a data set. The number of data sets is determined through ‘len’ function. Through the first for loop, determine the count of each class values. Then, in the second for statement, calculate the probability for each class and update the entropy. Returns the computed entropy.



If the ‘n’th attribute of data is equal to ‘value’, insert into retDataSet. The ‘n’th attribute is excluded when you insert it. It excutes all data using for loop. It is used to calculate Info\_A (D) or to pass the decision tree to the next parameter when it completes recursively. In the example below, age is determined and the table is split for age.

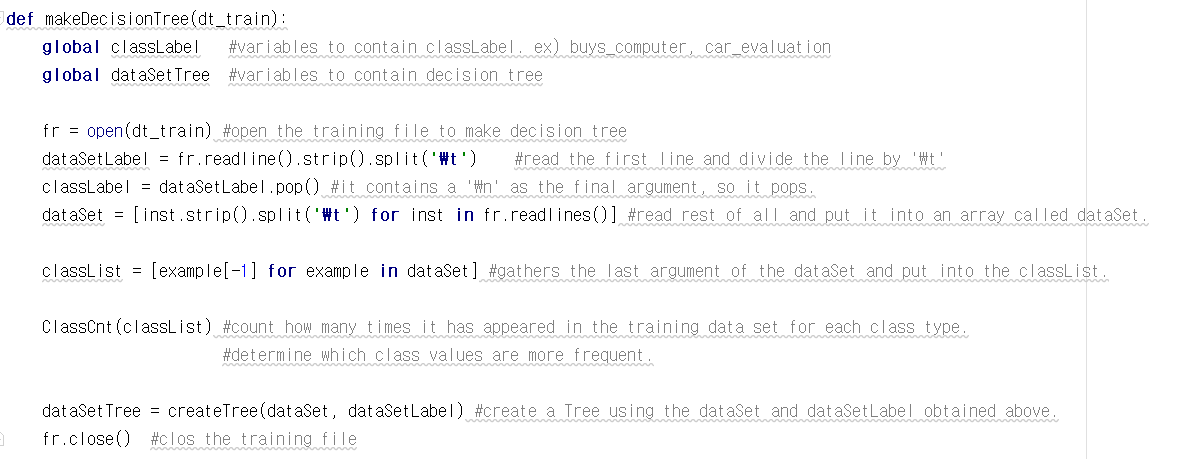




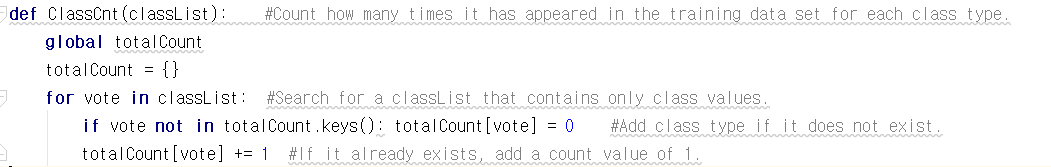
I use the two functions described above. First, calculate Info (D) by doing calcEnt on dataSet. Then we use the splitDataSet to calculate the entropy value and gain ratio for each attribute and return the index of the attribute with the highest gain ratio.



First, get the best feature to split using ‘chooseBestFeatureToSplit’. Then update the value in the decision tree called myTree and create an array called uniqVals with values corresponding to bestFeat. Use the for loop and ‘splitDataSet’ functions to complete the sub-tree recursively.

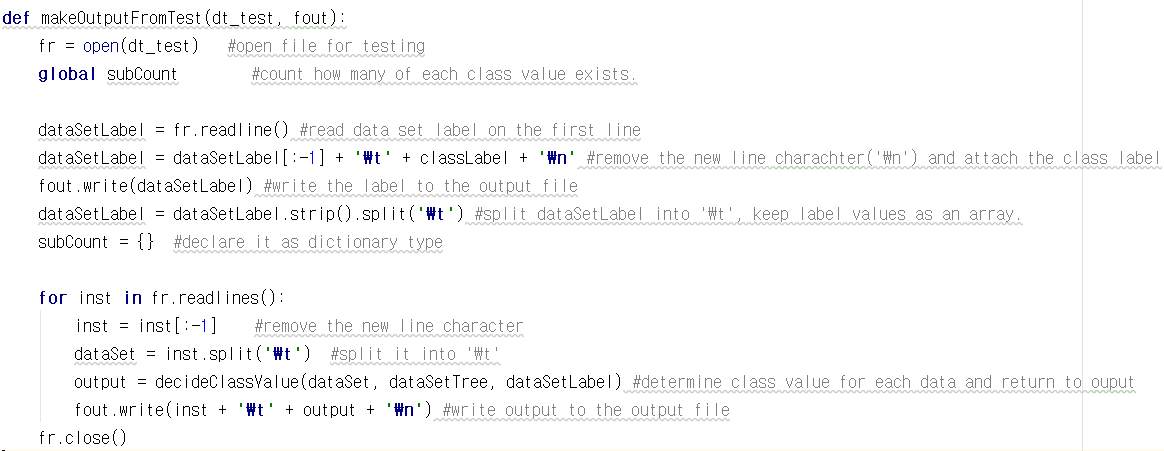


It takes a train text file as an input value and generates a dataset. Declare datSetTree as a global variable so that I can access it anywhere. The first line is read as a dataSetLabel and then the dataSet is received. Use ‘split’ to receive data like a two-dimensional array. And use the ‘classCnt’ function to store the frequency of each class value for later. Then we make decision trees using the functions mentioned above.

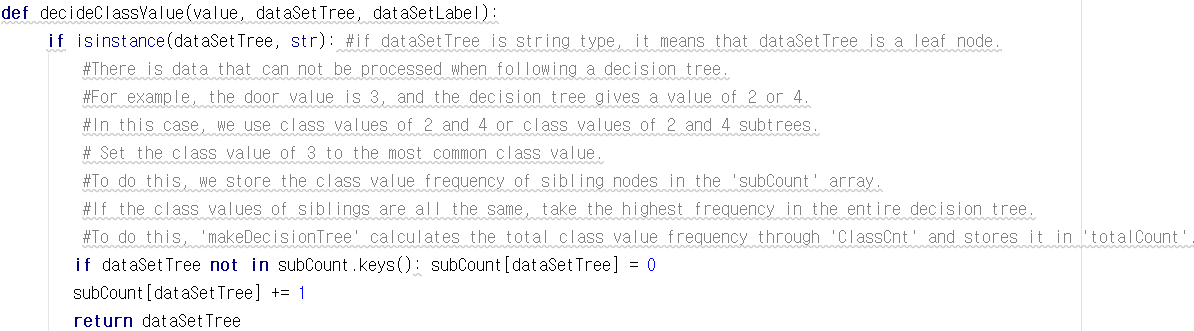


In ‘classCnt’, calculate and store the frequency of class values for the entire train data set.

This completes the decision tree. We can now estimate the class value with the test data set.

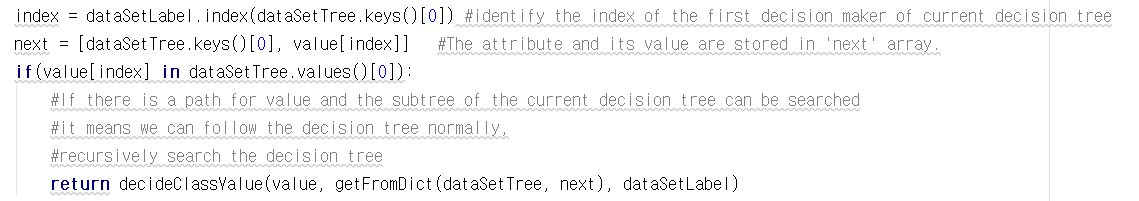


‘MakeOutputFileFromTest’ opens the test text file to get data set and creates the output file. First read the label value from the text file and write it to the ouput file according to the form. Then, read the data line by line in the test file, put it into ‘decideClassValue’ function, get classValue and write it to outputfile.



It is the first part of the function ‘decideClassValue’ which sets classValue. Since dataSetTree will search recursively, the tree is gradually reduced to its subtrees, which will eventually become leaf nodes. Because the tree is a dictionary type and the leaf node is a string type, you can check whether the leaf node has been reached by checking type of dataSetTree.

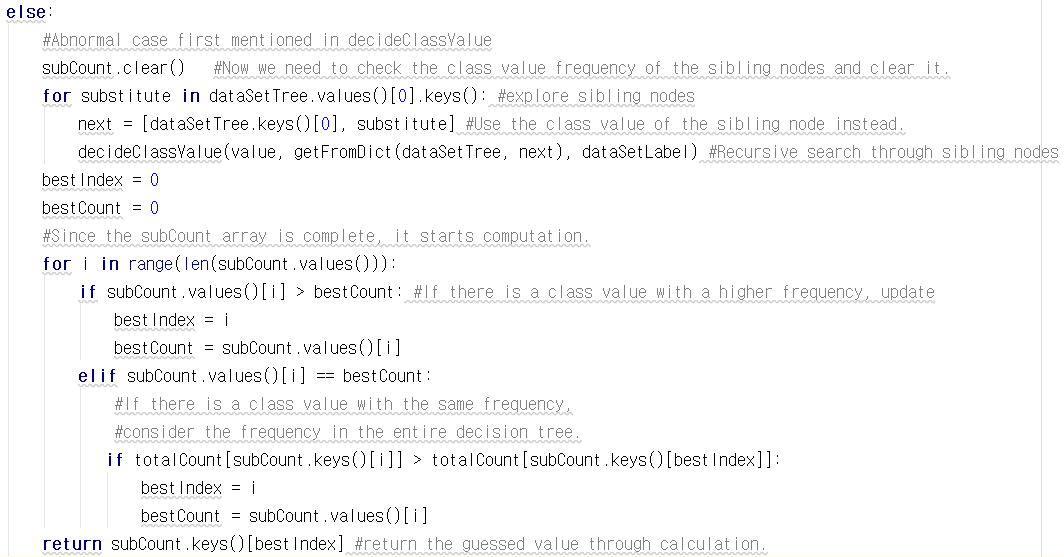
The outer if statement is related to the above. But the if statement inside is the content of counting the number of class values. This is necessary for exception handling. Sometimes there is a problem that the data goes down the decision tree and the path disappears. For example, suppose the decision tree is {door: {2: 'acc', 4: 'unacc'}}. In this situation, if the door value is 3, it can not be processed. Therefore, we decided to use the class value frequency of sibling nodes. The array containing the frequency is a subCount array and a global variable. I just used the more frequent class value, which comes from the end of this function.



In this case, dataSetTree is a tree, not a leafnode. The key value to be passed next and the corresponding value are put in ‘next’ array. 'Next' contains the condition of the next subtree. For example, ['age', '<30']. If you put it as an argument to getFromDict function, it returns subtree according to the condition in 'next' in dataSetTree.

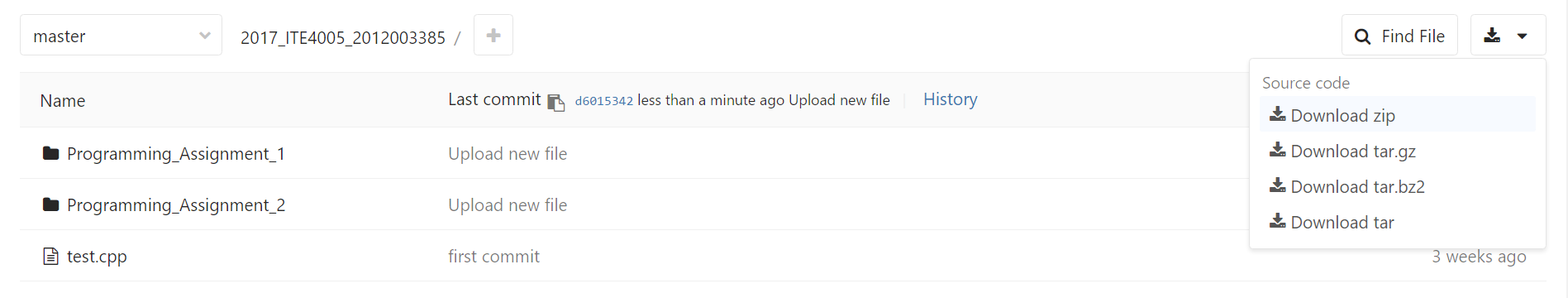


Returns the subtree according to the condition of mapList.



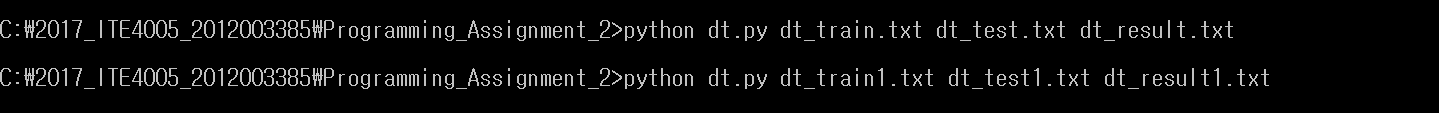
This is the case ‘can not find the path’ mentioned above. ‘subCount.clear’ should be used to count the number of class values of siblings since this problem occurred. Through the for loop, it recursively searches for sibling nodes and completes subCount. Then simply select the most frequent class value. If the siblings have the same class value frequency, they return the most frequent value according to the frequency in the whole decision tree.

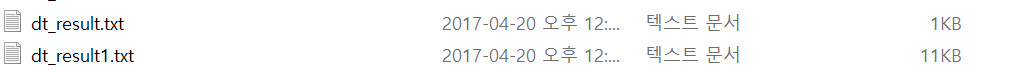
**3. Instruction for compiling codes**



First, download the zip file from gitlab and extract it.

Then open the command window and move to the location where dt.py exists and enter the following in the command window.





Then you can see that the result file is created.