**Data Science (ITE4005)**

**Programming Assignment #2**

**Decision Tree**



컴퓨터공학부 컴퓨터전공

2009004065 유건열

목차

[1. Environment 2](#_Toc480412438)

[2. Summary of algorithm 2](#_Toc480412439)

[(1) Attribute Selection Measure 3](#_Toc480412440)

[A. Information Gain 3](#_Toc480412441)

[B. Gain Ratio 4](#_Toc480412442)

[C. Gini Index 5](#_Toc480412443)

[(2) Expanding the tree 5](#_Toc480412444)

[(3) Cutting the tree 5](#_Toc480412445)

[(4) Classifying the test data 6](#_Toc480412446)

[3. Detailed description of codes 6](#_Toc480412447)

[(1) Main flow 6](#_Toc480412448)

[(2) Detailed description 8](#_Toc480412449)

[ Imported Libraries 8](#_Toc480412450)

[ Class for Decision Tree Node 8](#_Toc480412451)

[ Handling user’s argument 9](#_Toc480412452)

[ Opening files 10](#_Toc480412453)

[ Getting attributes 10](#_Toc480412454)

[ Making the list of tuples from the training file 10](#_Toc480412455)

[ Getting class label from the attributes 11](#_Toc480412456)

[ Initialization of root node 11](#_Toc480412457)

[ Expanding decision tree 12](#_Toc480412458)

[ Classifying test data 22](#_Toc480412459)

[ Closing files 23](#_Toc480412460)

[4. Instructions for executing the source code 23](#_Toc480412461)

[5. Result of test 24](#_Toc480412462)

# 1. Environment

- OS : Windows 10

- Language : Python 2.7.12

# 2. Summary of algorithm



The objective of this assignment is to make decision tree. To make decision tree, the important thing is to decide which attribute is good for classify. There are three method for making a decision of attribute in this assignment. : Information gain, Gain ratio, Gini index.

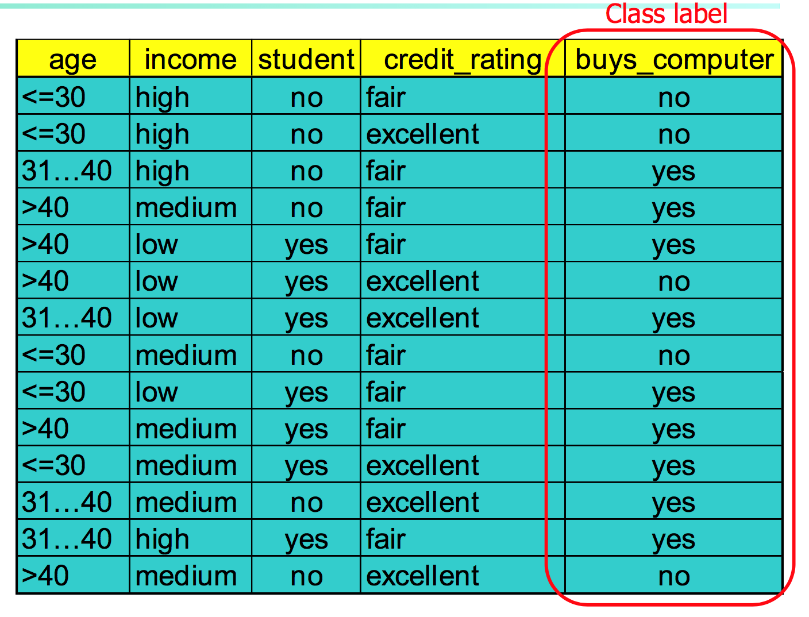
In this assignment, we have a training data set like Figure 1. Every tuple has some attributes. (age, income, student, credit\_rating, buys\_computer) And, they have each class label (buys\_computer).

Figure 1

We’ll make the decision tree with the training data set. Decision tree is for classifying the new data set called test data set. This process is giving labels to the test data set without class label.

I’ll make the root node of the tree first. It has every tuple of the training set. So the tree will be constructed in a top-down recursive divide-and-conquer manner. After deciding the attribute of the next depth, I’ll make the nodes of values in attributes. (ex. There are three values in income attribute. : high, medium, low) This partitioning process will be taken recursively. So we have to make conditions for stopping partitioning.

1. All samples for a given node belong to the same class. If every tuple in the node has same label, partitioning has to be stopped. That label will be labeled to the node.

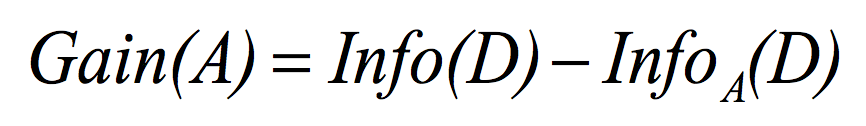
2. There are no remaining attributes for further partitioning. In this case, majority voting is employed for classifying the leaf. If a label has the biggest number of tuples, that label will be labeled to the node.

3. There are no samples left. In this case, that node will be removed in the decision tree.

## (1) Attribute Selection Measure

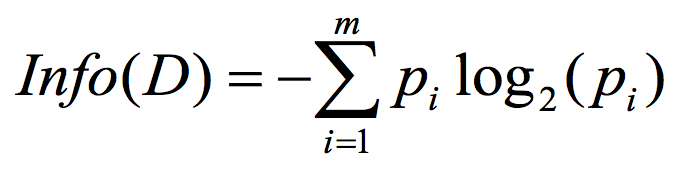
### A. Information Gain

This method is to select the attribute with the highest information gain. I’ll show you how to calculate the information gain of attributes.

****

Figure

Figure 2 is the formula to calculate the information gain of attribute. Info(D) is the expected information of class label. And Info A (D) is the expected information of target attribute.



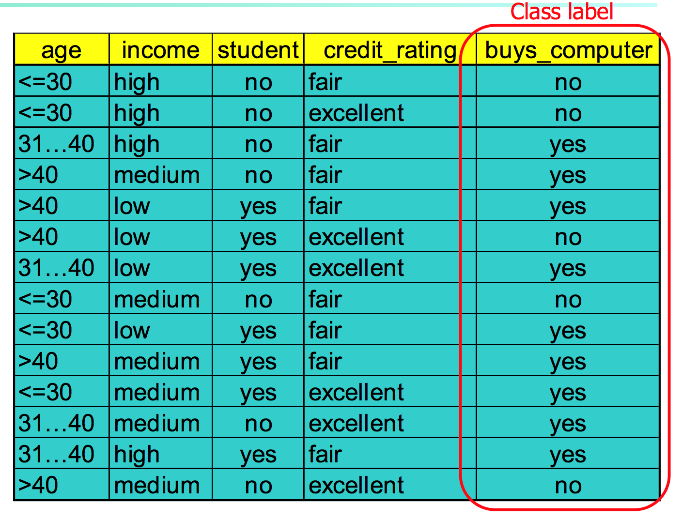
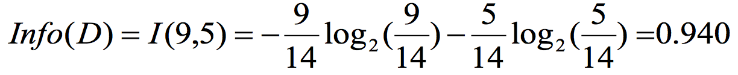
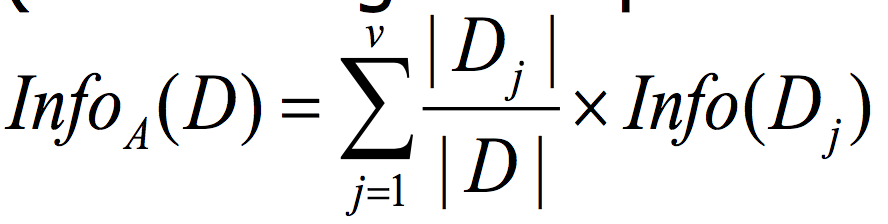
So, let’s calculate the expected information of class label. Pi is the probability that an arbitrary tuple in D belongs to class Ci, estimated by |Ci,D|/|D|

Figure 3

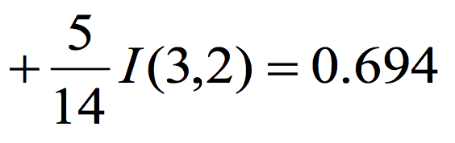
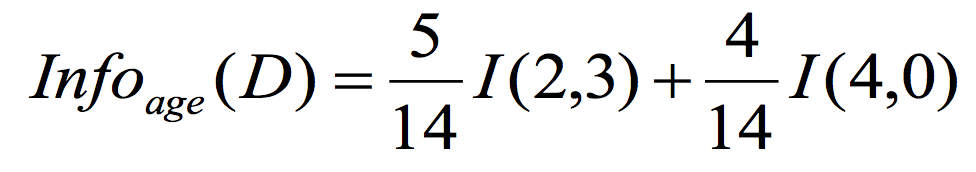
For example, buys\_computer has two values, ‘yes’, ‘no’. There are 14 tuples in the training set. 9 tuples are labeled ‘yes’, and 5 tuples are labeled ‘no’. So, the expected information of class label is



Next, we’ll calculate the expected information of the target attribute.



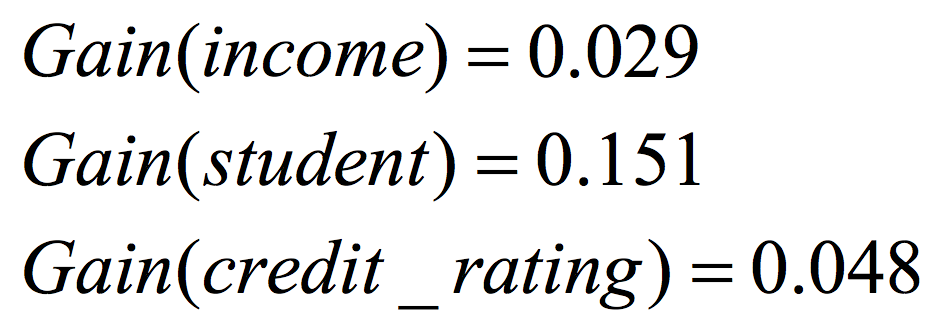
At this time, let’s calculate the expected information of ‘age’ as an example.



First item is for ‘age : <=30’. There are 5 tuples that have age <= 30. And 2 tuples are labeled ‘yes’. 3 tuples are labeled ‘no’. Other items can be calculated as same method. Finally, let’s get the information gain of age attribute.



Similarly, the information gain of other attributes are :

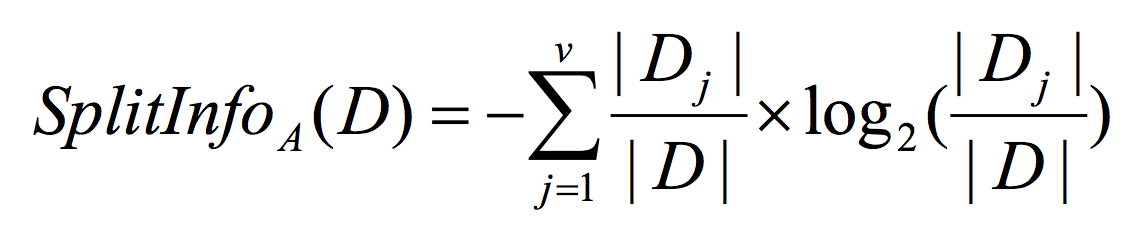


We can find the information gain of age is the biggest among the all information gains. So, this step, we can choose the age attribute! Because we choose the age attribute, we cannot choose this attribute in next step. I’ll save this information about the attribute remained in the nodes.

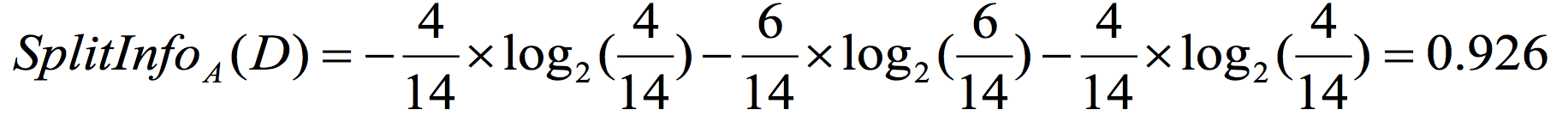
### B. Gain Ratio

Information gain has a problem. It is biased towards attributes with a large number of values. The large number of values makes the expected information smaller. And then, the information gain will be larger.

Gain Ratio is used to overcome the problem by normalization to information gain with split information.



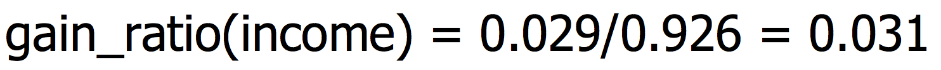
For example, let’s calculate the split information of income.



Next, we can calculate the gain ratio with this split information.



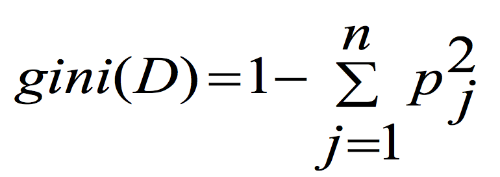
Gain Ratio is gain divided by split information.

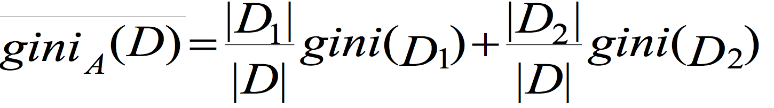


As same with the information gain, we have to choose the attribute that has the largest gain ratio.

### C. Gini Index

When I finished the process with Gain Ratio, I got 90% for accuracy. I wanted to increase the accuracy. So, I decided to attempt the Gini Index.





Gini index is calculating the impurity of data sets. It counts How many other labels are contained in data set. So, it needs to calculate by one attribute value. For example, when we calculate the Gini index of attribute ‘income’, we have to calculate the Gini index of each attribute value. (high, medium, low) And then, I used the lowest Gini Index for the attribute. (If the Gini index of the ‘high’ is the lowest, that is the representative of the ‘income’ Gini Index.) I’ll show you how to calculate Gini index exactly in detailed description section.

## (2) Expanding the tree

After selecting the attribute, the program will expand the tree. It will make children node that have the selected attribute and each key. If ‘income’ is selected, the children node will be three, ‘high’, ‘medium’, ‘low’. Of course, any child node has no tuple that has node’s attribute value, that child node will be deleted. Next, the tuples will be distributed to the correct child node. In my program, tree nodes have ‘attributeRemained’ list. It saves the attribute candidates to decide. If a node’s attribute is ‘income’. Its children node’s ‘attributeRemained’ are not contained ‘income’. After generating the children node, the selection of attribute will be processed recursively.

## (3) Cutting the tree

Building tree is recursive job. So, we need the terminate condition. There are some cases that should be stopped to expand the tree in this program.

1) The node has no attribute to decide.

In this case, the node should be labeled by voting. By counting the biggest number of label, the label of node will be decided.

2) The tuples in node have the same label.

If every tuple of a node has ‘yes’ for class label : buy computer, we don’t need to split the tuples anymore. We can just label the node ‘yes’, and cut the branch there.

3) There is no tuple for the node.

Let’s suppose we decided ‘income’ for the next attribute. But what if we don’t have any tuple for ‘high’? In this case, we don’t need to make child node of ‘income : high’.

4) The number of tuples in a node is less than five.

This is the special terminate condition of my program. When I did this things with normal Information gain, Gain ratio. Gini index, the highest number of correct tuples was 314. I wanted to increase the accuracy. And I found some nodes have very few number of tuples. (ex. 1, 2, …) So, I thought if I make the threshold of the minimum number of tuples, the accuracy would increase. I tested several thresholds, and I recovered 5 is the best threshold for these training sets and test sets. The number of correct tuples was 321. I’m not confidence that 5 is the best threshold in other cases. But I think it is worth a try.

## (4) Classifying the test data

Classifying the test data is very easy thing. Just put the tuple into the root the node one by one. At each node, the tuple searches the child node that has the same attribute and value with the attribute and value that the tuple has. If the tuple find the correct child node, the tuple will move to that child node. And above process will be taken recursively. In some case, the tuple cannot find the correct child node. For example, if there are no tuple that has ‘income : high’, of course, there are no tree node that has ‘income : high’. But what if there are some tuples that has ‘income : high’ in the test set? In this case, the tuple will go to the child node that has the biggest number of tuples within. (ex. If ‘income : low’ child node has 38 tuples, and ‘income : medium’ child node has 144 tuples, the test tuple will go ‘income : medium’ child node. Even though, the node has ‘income : low’ attribute value.)

# 3. Detailed description of codes

## (1) Main flow

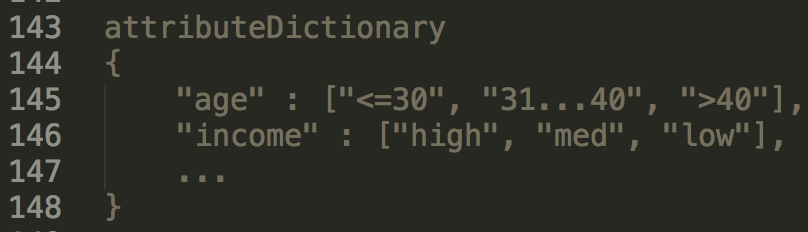
First, the program opens the training file, and test file. Next, it reads the tuples from the training file.

It starts making decision tree with root node. Root node has every tuple in training file, and has every attribute for decision.

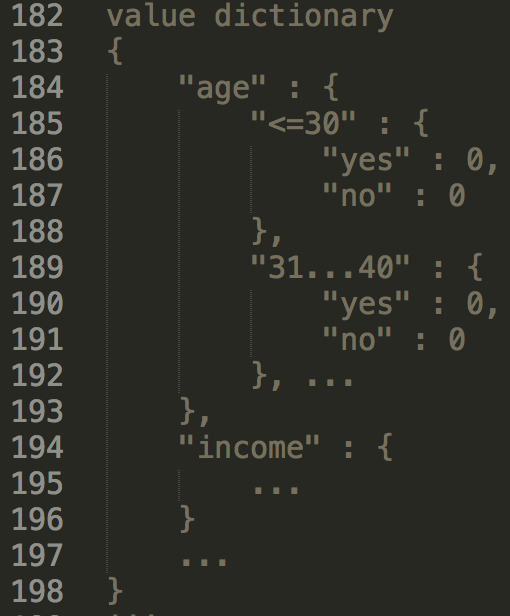
From this step until complete to build decision tree, these processes will be taken recursively. I will call this 'Expanding tree process'.

First, we need to know which attributes and values exist in this node. It scans the whole tuples and gets an 'attribute dictionary.'

* Attribute dictionary example



Using this 'attribute dictionary' as a frame, it makes 'value dictionary' for count the tuples that has certain attribute values.

* Value dictionary example

Using this value dictionary, we can calculate score for each attribute selection measure. (information gain, gain ratio, gini index) I will explain the detail of the process that calculates each score.

After calculating the score, we can choose the best attribute for next depth.

With this attribute, the program will make children node. And tuples in node will be distributed to children node. This 'Expanding tree process' will be taken recursively.

* Stop condition

There are some stop conditions for expanding the tree in this program as I mentioned in earlier section.

1) The node has no attribute to decide.

2) The tuples in node have the same label.

3) There is no tuple for the node.

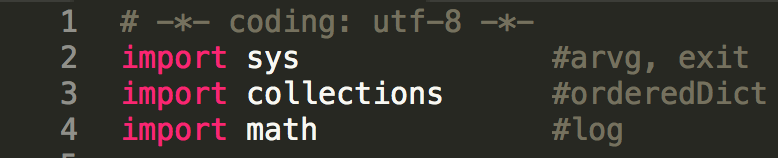
4) The number of tuples in a node is less than five.

I’ll explain about how to be implemented this stop condition in the next section.

After building the decision tree, the program will read the test file and classify each data tuple one by one. Finally, write the consequence of the classification into the output file.

## (2) Detailed description

### Imported Libraries

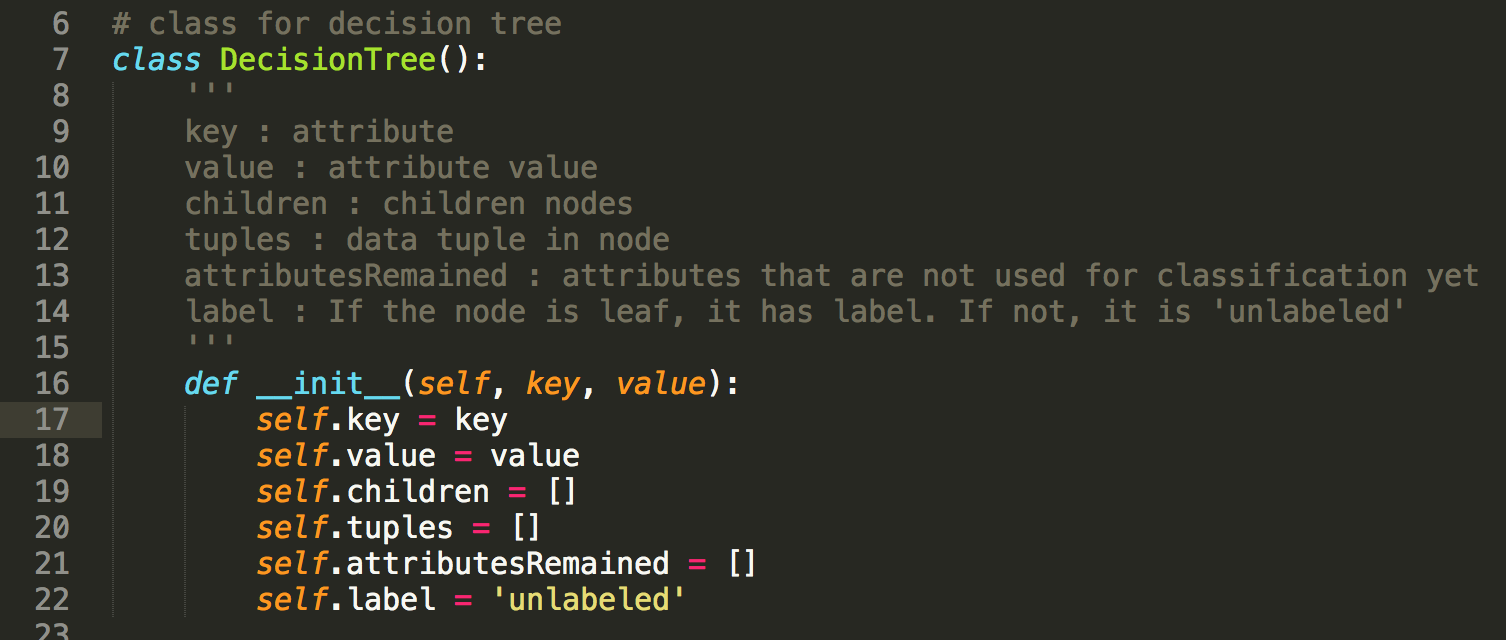


I imported sys for ‘argv’ that is for user’s argument, and for exit.

Collections is for ‘orderedDict’. The original dictionary of python doesn’t have order. So I thought it is not proper for this assignment. That’s why I used ‘orderedDict’.

In every formula to calculate the measure for the selection of attribute, there are logs within. Therefore, I imported math for log.

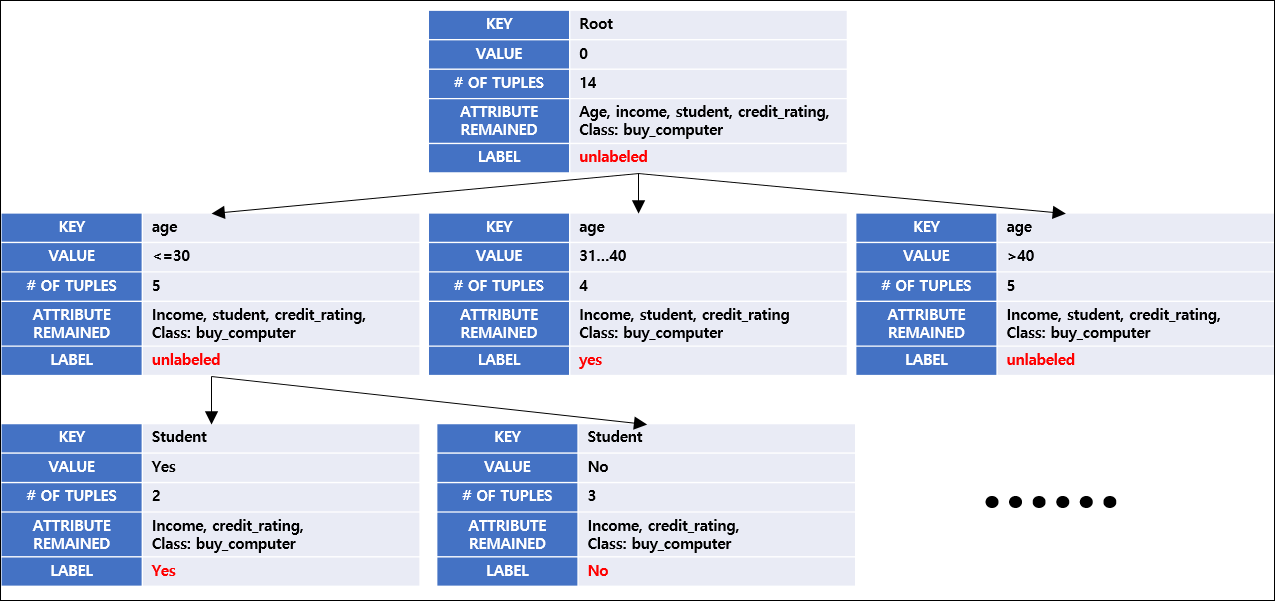
### Class for Decision Tree Node



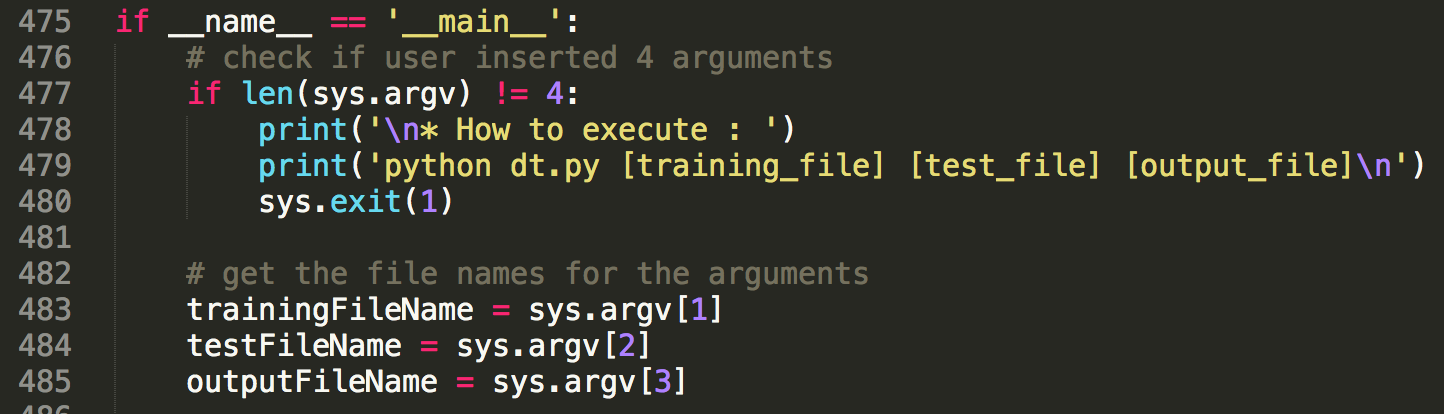
This is ‘DecisionTree’ class in my program. The attributes are for each node of decision tree.

* Key : the attribute of node. If the node is root node, the key is ‘root’.
* Value : the attribute value of node. If the node is root node, the value is ‘value’.
* Children : the list of the children node
* Tuples : the list of the tuples in the node
* AttributesRemained : the list of the attributes remained in the node. Root node has every attribute in the training set.
* Label : the label of the node. If the node is not leaf node, label value is ‘unlabeled’ as default.

For example, the output of decision tree in this program will looks like this.



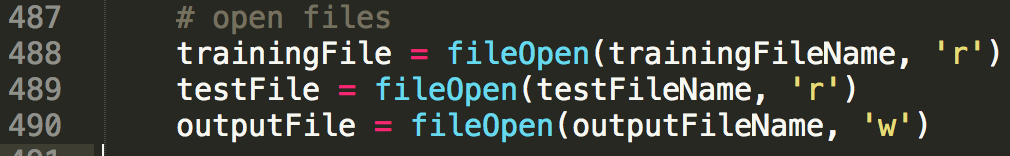
### Handling user’s argument

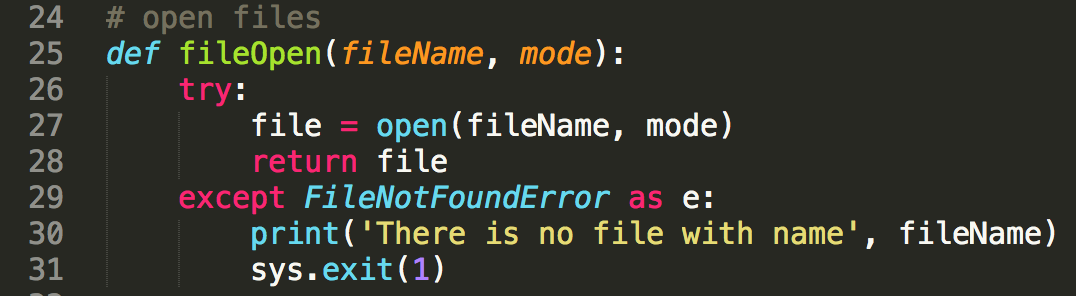


In main function, it checks the argument’s number first. We need four argument including file name. (training file name, test file name, output file name). If the argument is not 4, it shows ‘how to execute’ and the program will be exit.

If user insert 4 arguments exactly. We can get training file name, test file name, and output file name.

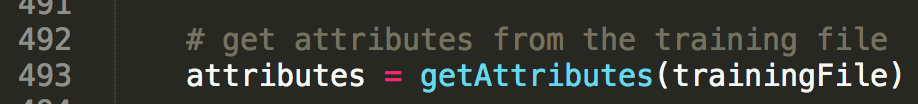
### Opening files

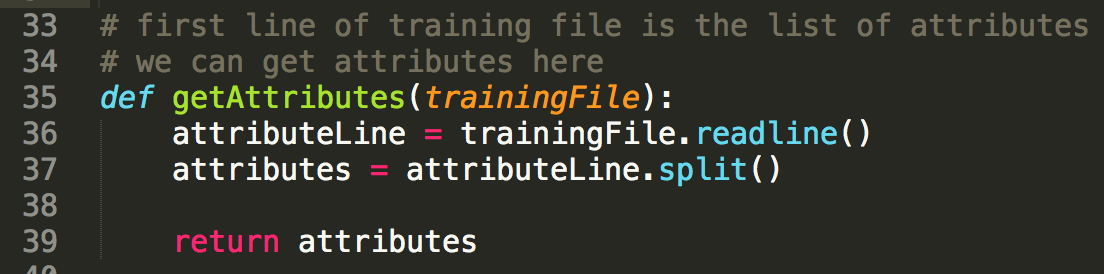




And, it opens files for training data, test data, and output. ‘fileOpen’ function will do that. If ‘FileNotFoundError’ occurs, program will be exit. (If user insert file name is not in directory.)

### Getting attributes

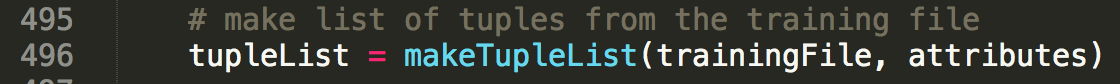


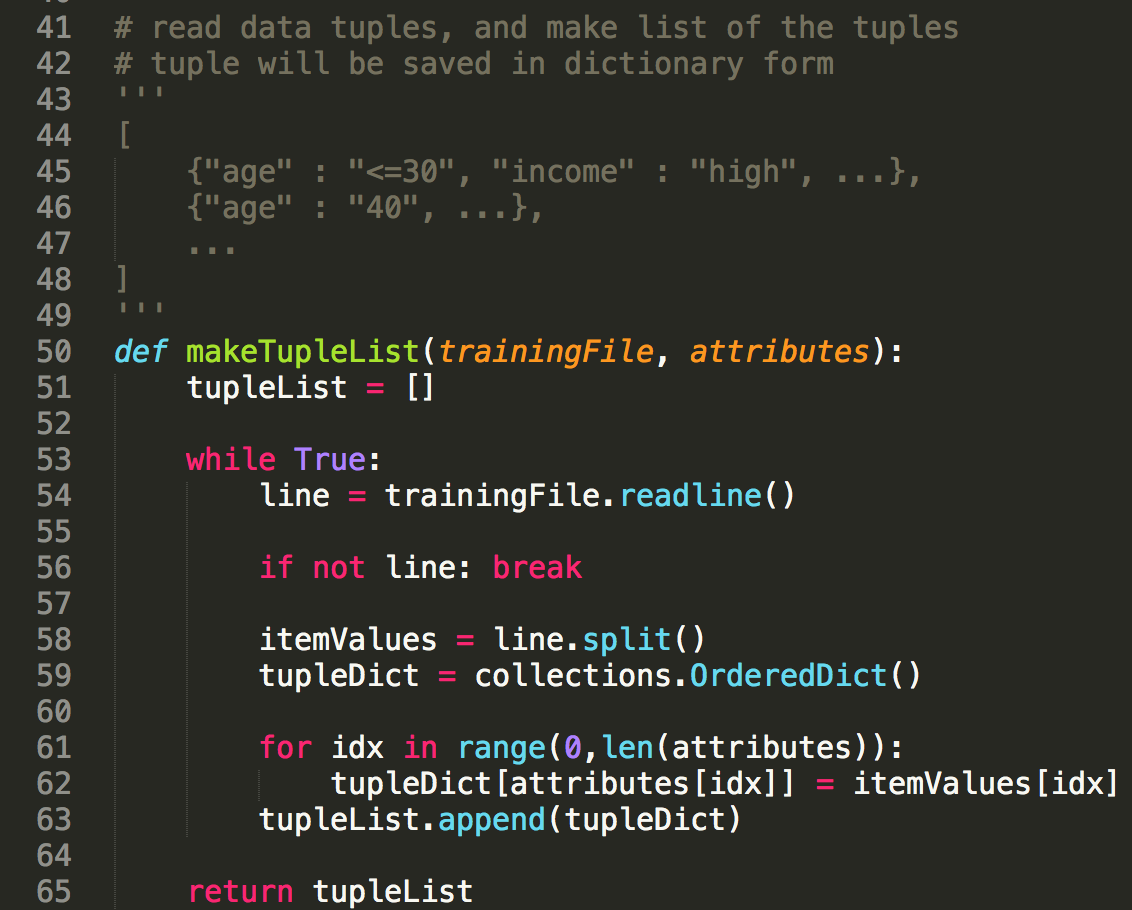


When building the decision tree, the most important thing is attribute. The first line of the training file is the list of the attributes. So, by reading it, we can get attributes.

Split function splits sentence into string word by blank.

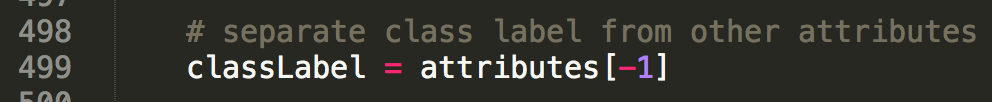
### Making the list of tuples from the training file





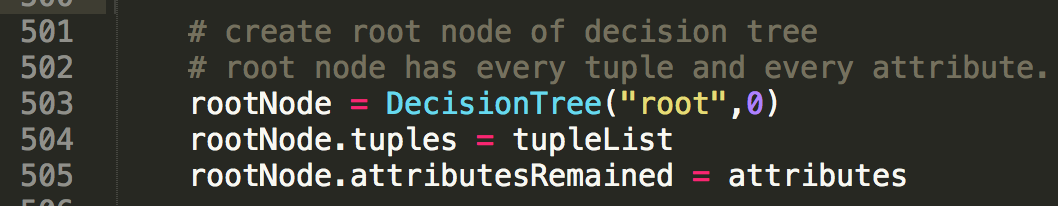
The program reads the rest of the lines of the training file. And every line is saved as a dictionary. The keys are attribute name, and values are attribute value. ‘tupleList’ list variable contains every dictionary of tuple in list.

### Getting class label from the attributes



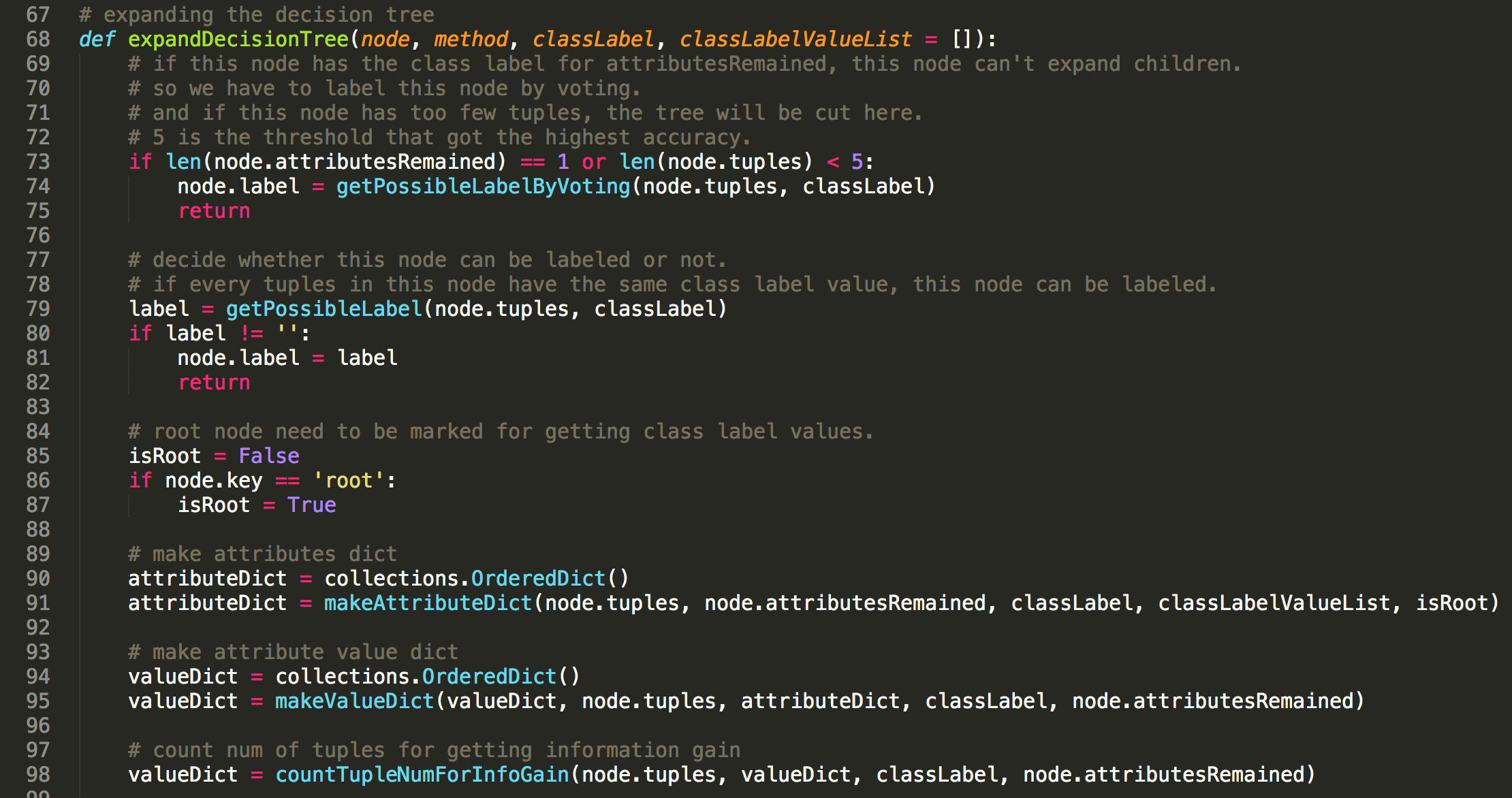
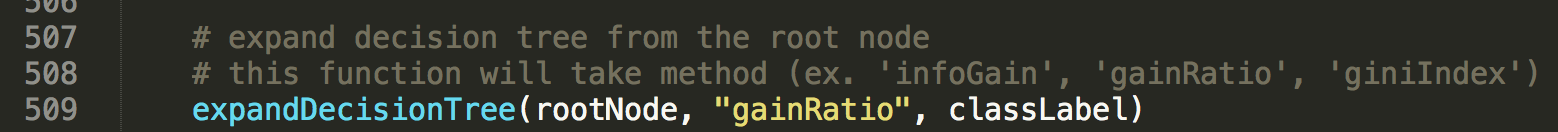
Then, we can get the class label from the list of attributes. In the training file, the last attribute is the class label. That’s why I get it from ‘attributes[-1]’.

### Initialization of root node

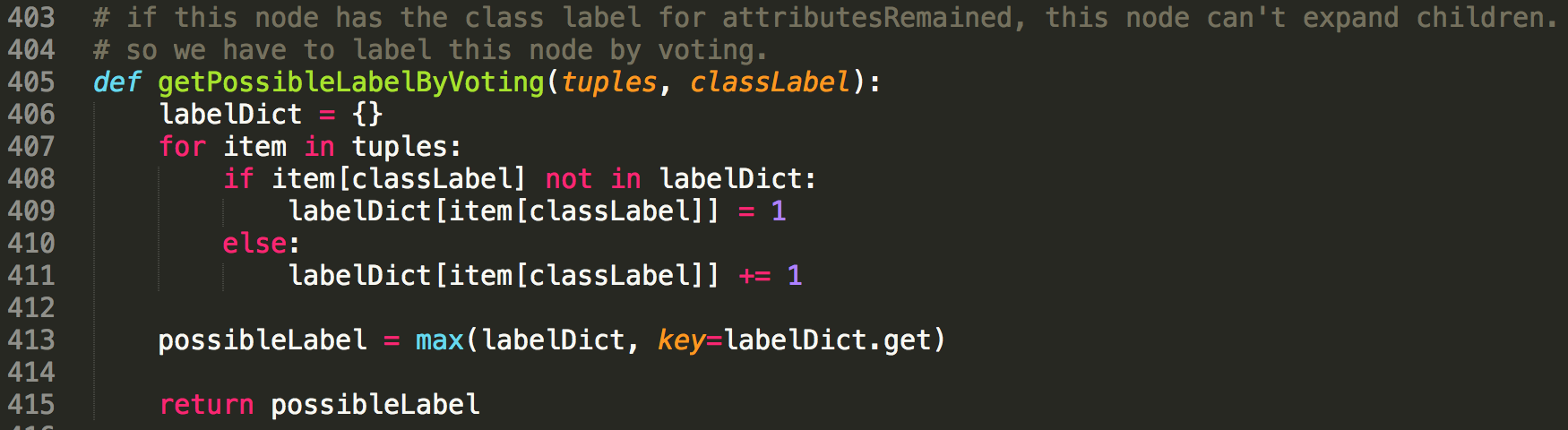


As I mentioned earlier, the key of root node is ‘root’, and value of it is 0. After generating root node, input every tuple in the training file into the node, and input every attribute into the node.

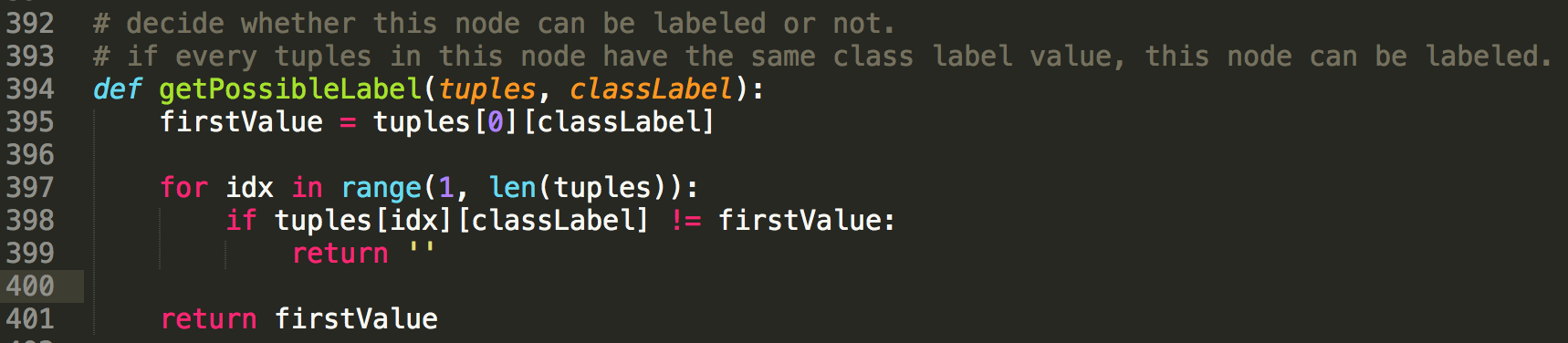
### Expanding decision tree



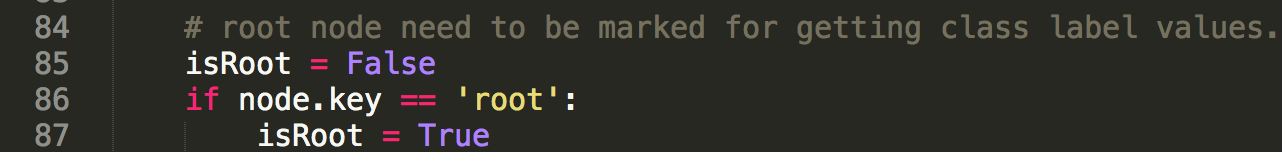
The process expanding decision tree will be taken recursively from the root node. I made all three methods to measure, Information gain, Gain ratio, and Gini index. And Gain ratio showed the best consequence. That’s why I chose it.



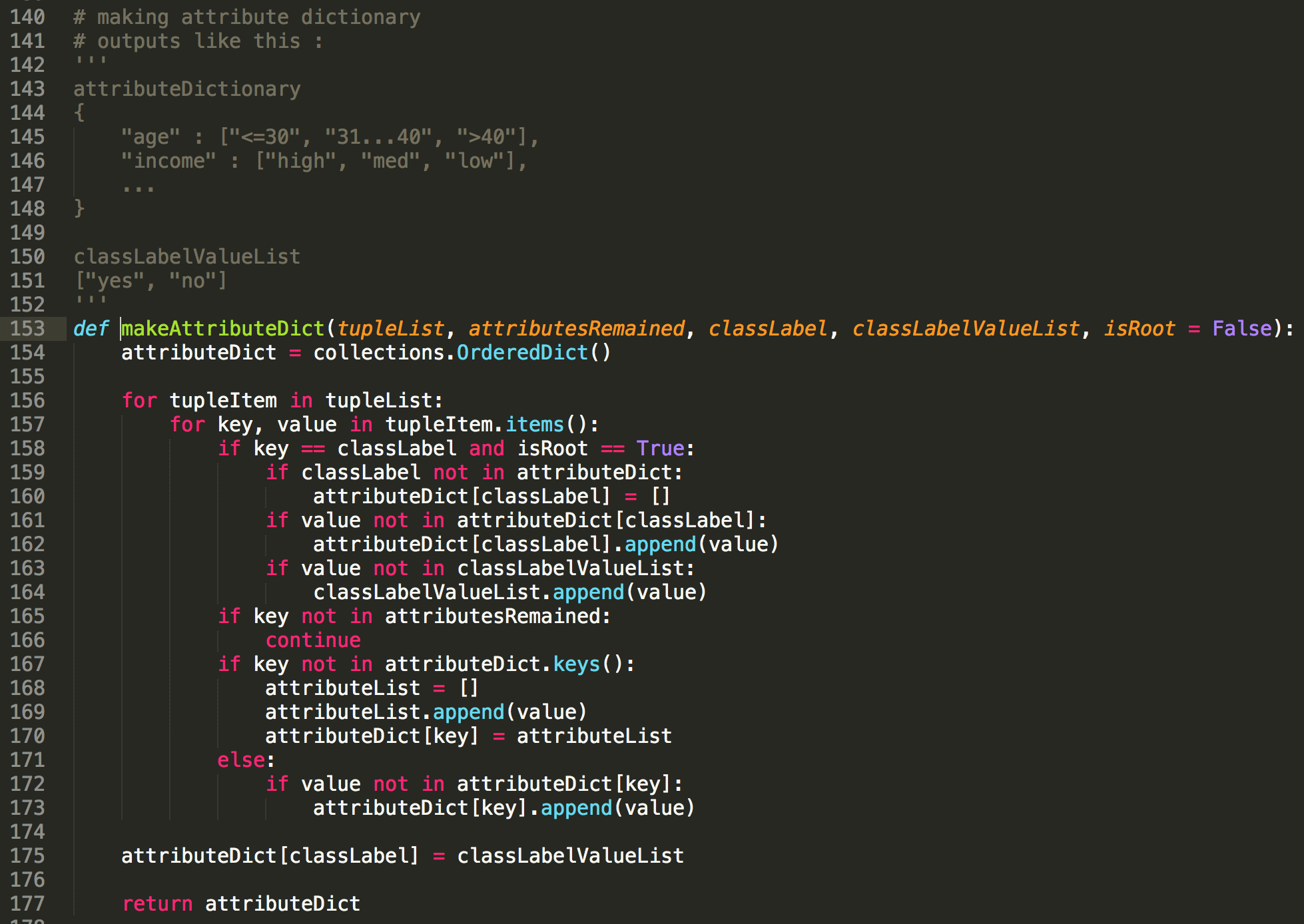
First of the function, it checks how many attributes are remained, and how many tuples are in node. If there are just one attribute, class label, we cannot expand the tree from that node anymore. So, we have to choose the attribute by voting in this case. I added another threshold here. If tuples in node are less than 5. It will do voting as well. After labeled, it stops to expand the tree there. Voting is just counting process for labels of every tuple. It returns the label that many tuples have.



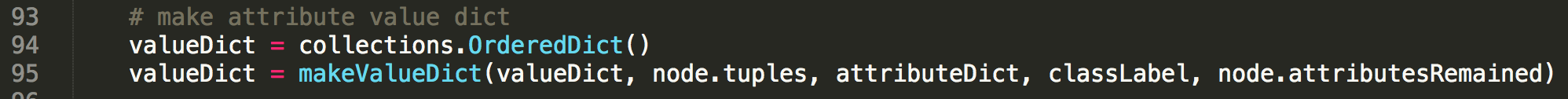
Otherwise, we should check the node is labelable or not. ‘getPossibleLabel’ function will do that. It is simple job. It just checks every tuple in node have the same label or not. Save the label of the first tuple in the beginning. And check the tuples’ label sequentially. If any tuple has the different label from that of the first tuple, the node can’t be labeled and the function return empty string. If ‘expandDecisionTree’ function got empty string, the process will go normally. Unless, the node will be labeled, and it’ll stop to expand the tree.

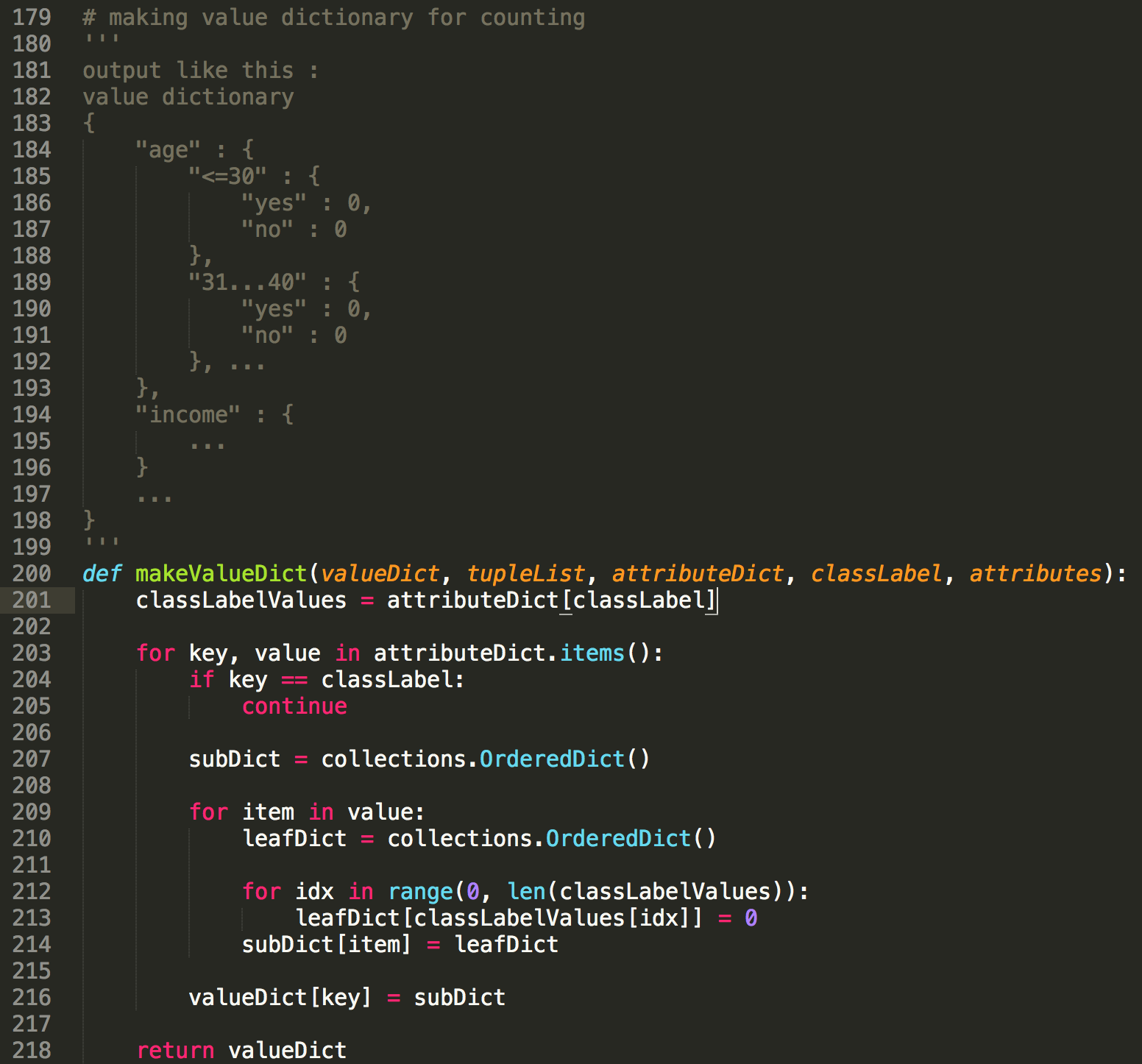


It checks if the node is root node or not. This boolean value will be used to make attribute dictionary later.

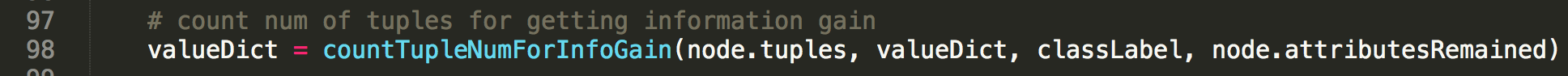


Next, the program will make attribute dictionary. The example of attribute dictionary is written in comments of the source code. This attribute dictionary will be used as a frame to make value dictionary. In addition, if the node is root node, it will make the list of class label values. This is important thing to label the tree nodes.

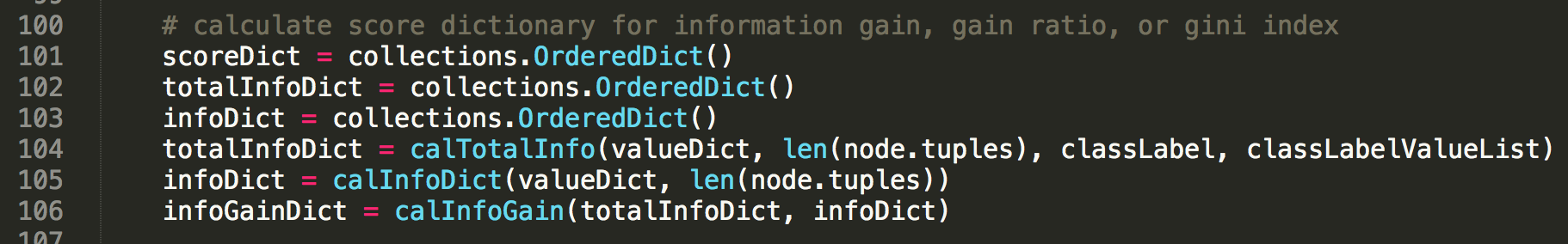




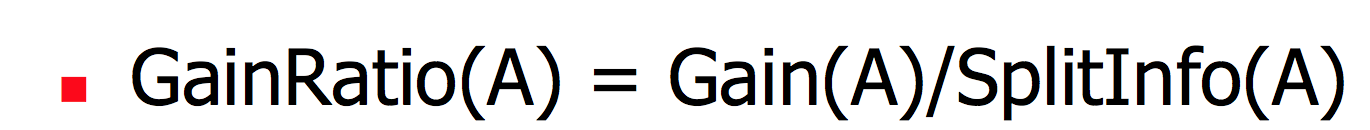
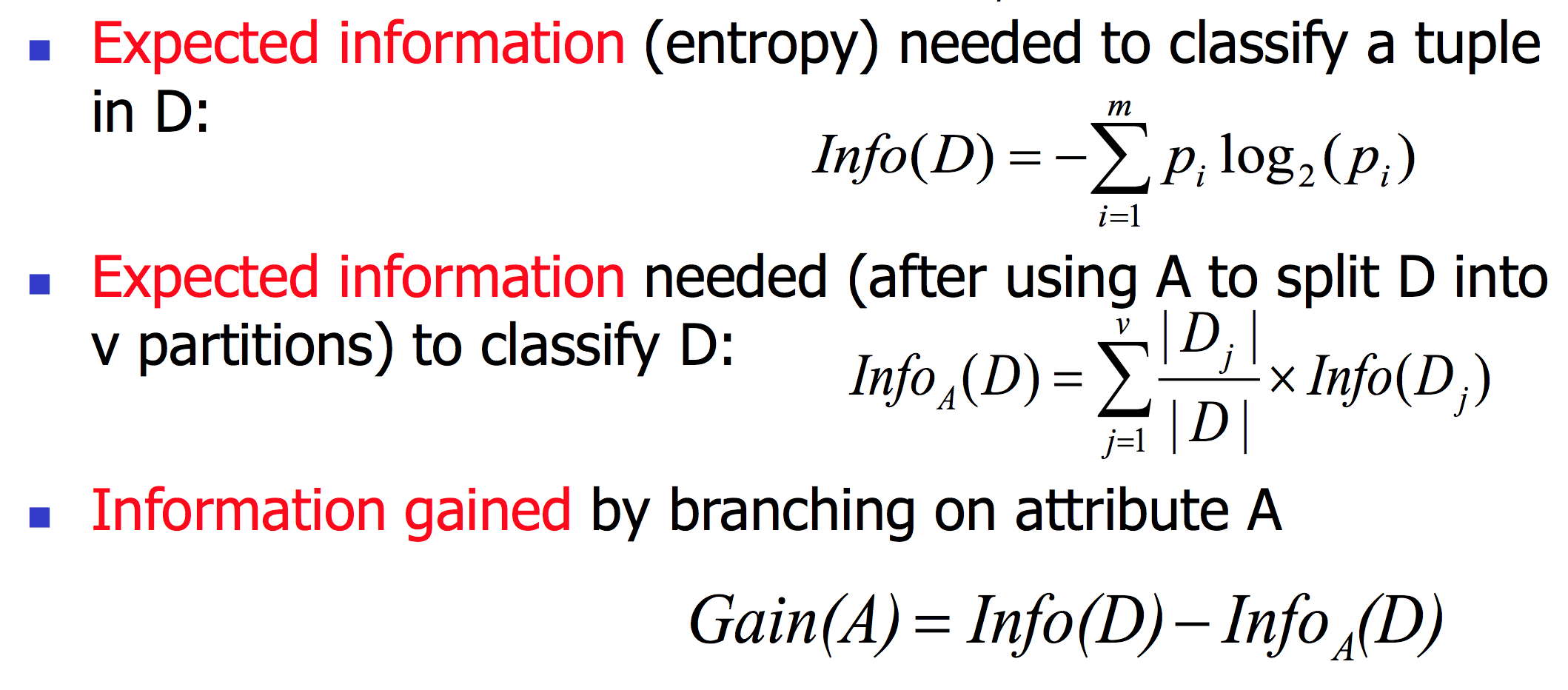
The program will make value dictionary by using the attribute dictionary. This value dictionary will save every number of each labeled tuple that has every attribute value. This function just makes the bone of data structure, and initialize every value to 0.



‘countTupleNumForInfoGain’ function just counts the number of tuples for getting the attribute measures. (Information gain, Gain ratio, Gini index)

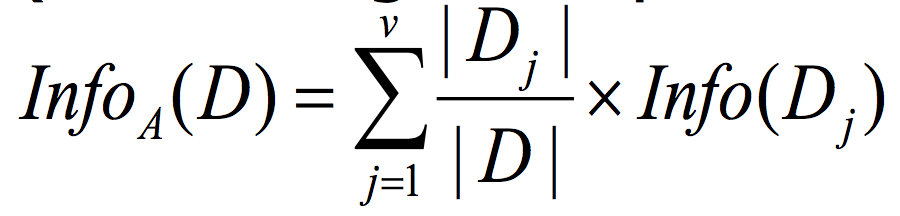
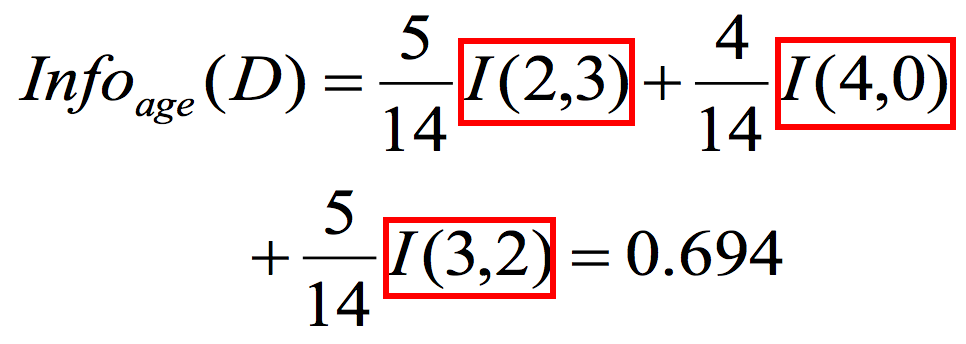


Finally, it’s time to calculate the attribute selection measures.

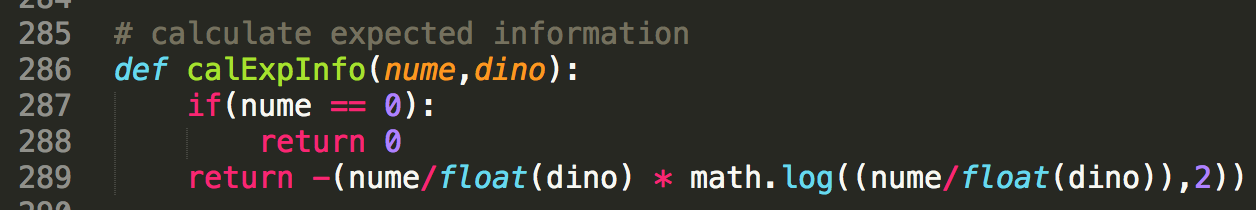


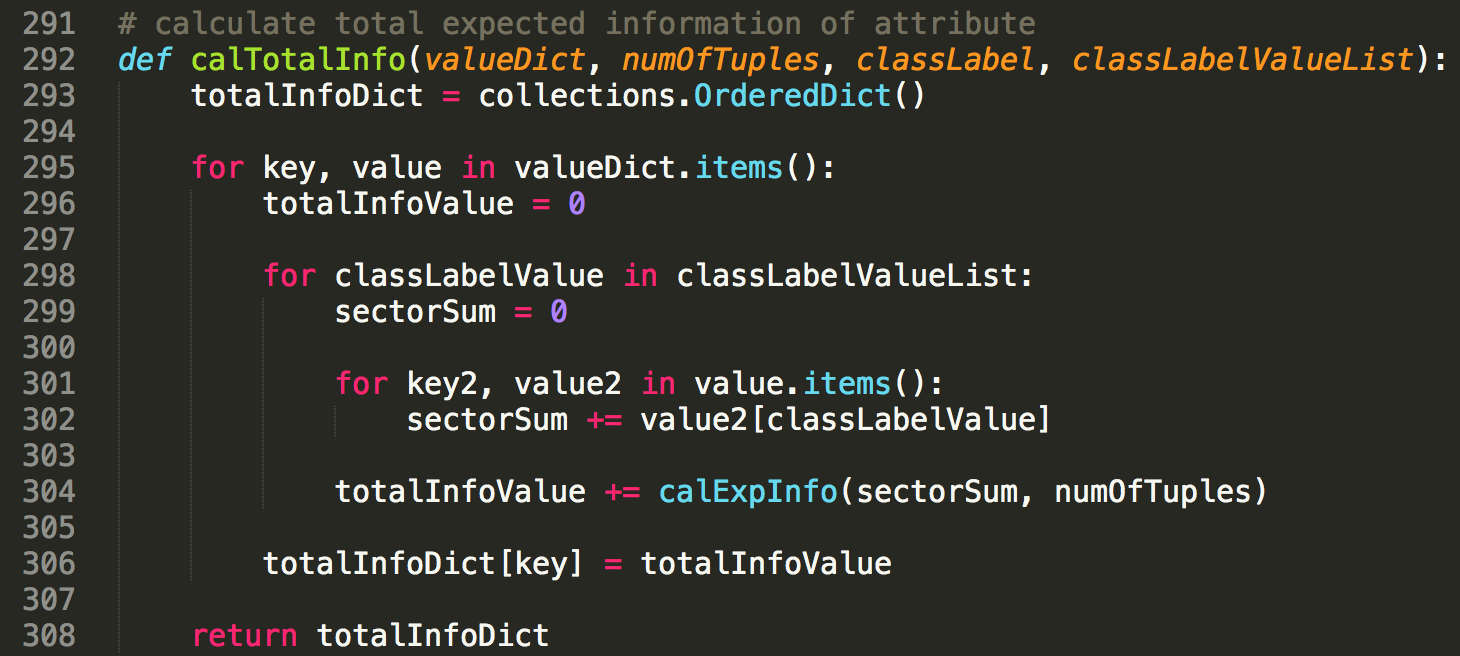
When we calculate Information gain, we need expected information of attribute. (not of only one value of attribute). Likewise, calculating Gain ratio also needs Information gain. So, we can say it needs expected information of attribute as well.

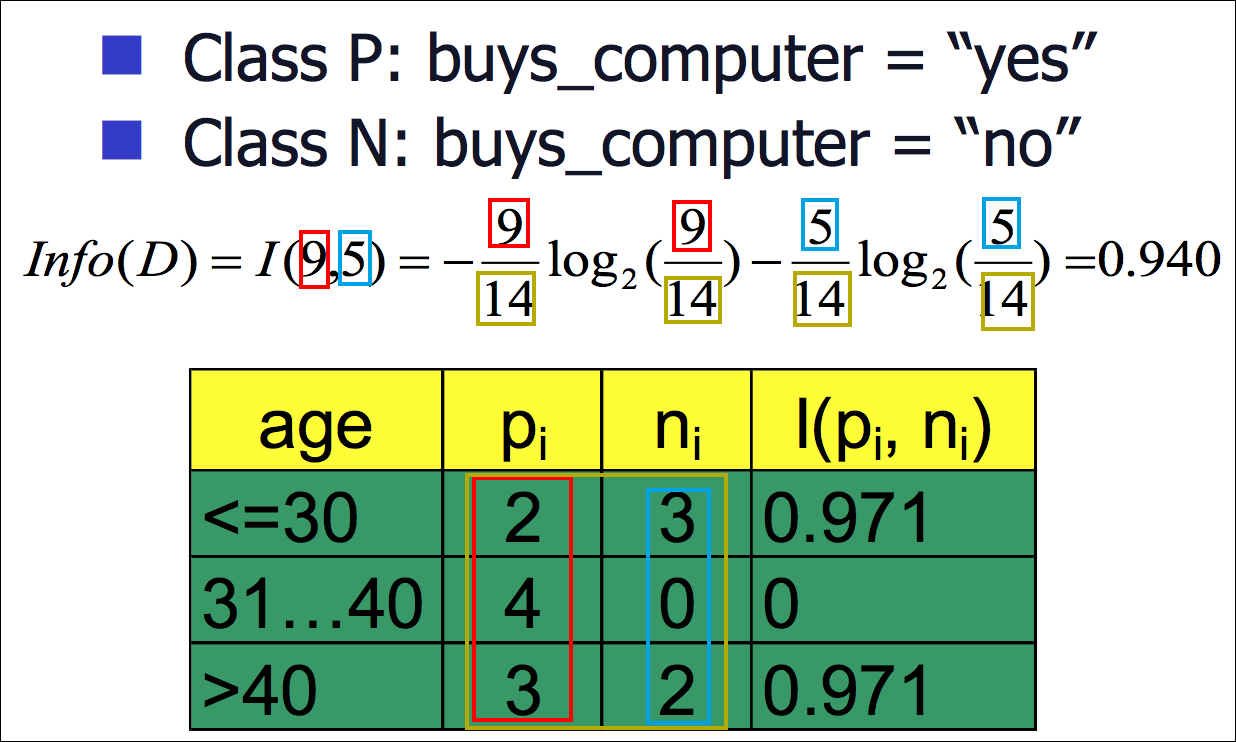
I called this expected information of attribute ‘total info’. ‘totalInfoDict’ will save these values. ‘infoDict’ will save the expected information of each attribute values. And then, Information gain can be got by substracting ‘infoDict’ value from ‘totalInfoDict’ value.



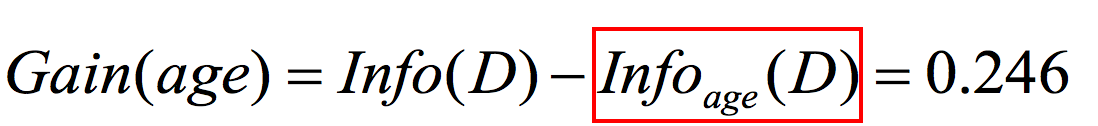
When we calculate the measure, we can see this n/m \* log (n/m) form in many times. So, I made it a function named ‘calExpInfo’.



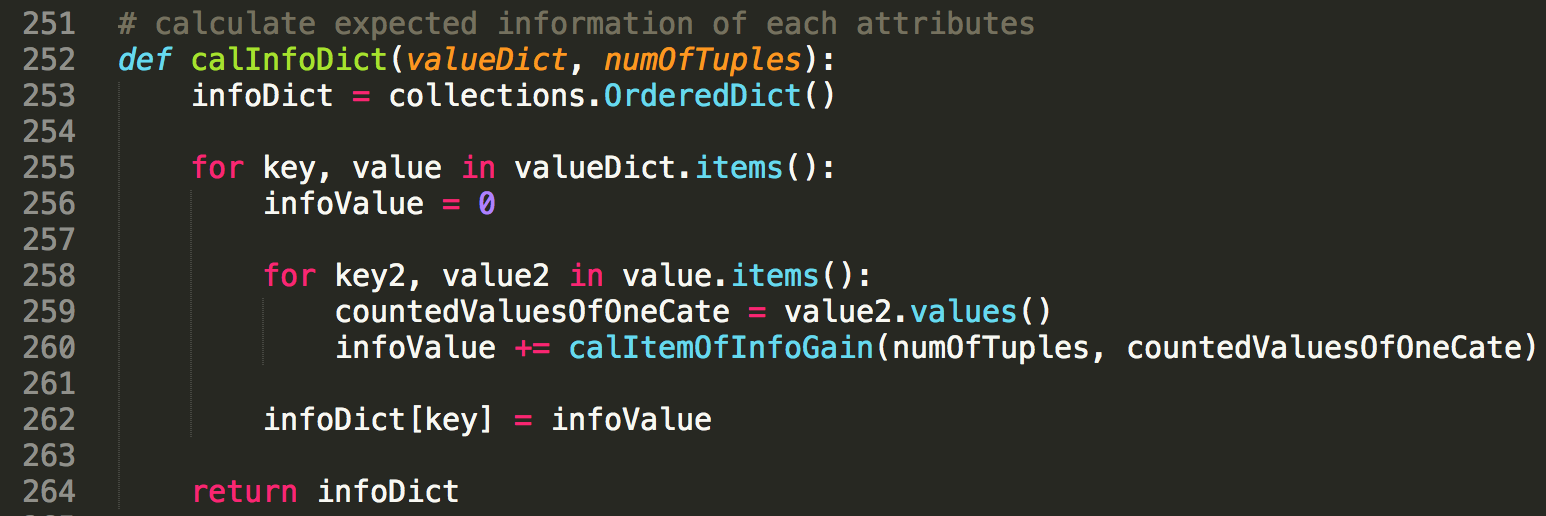


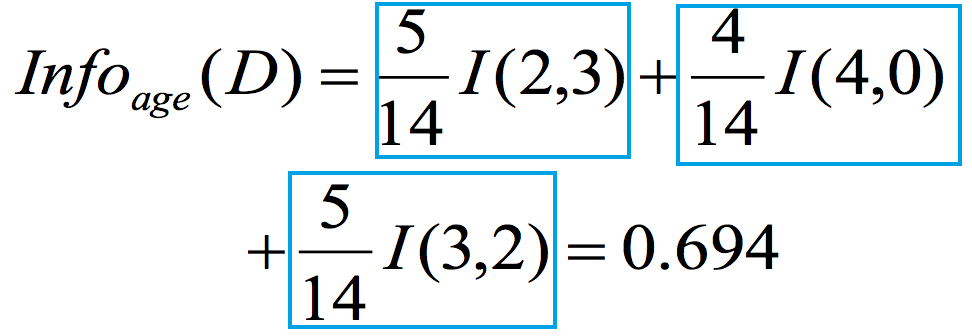


So, let’s calculate the total information first. Above picture shows how to calculate the total information. See the red, blue, and yellow squares. 9 can be got from the sum of the values in red square of the table. (2+4+3 = 9) Other colored squares are same. So, it shows if we have the table, we can calculate the total information easily. In this program, we have that table of every attribute. That is the value dictionary! So we don’t need to scan the DB again. We can get every value that we need from the value dictionary.

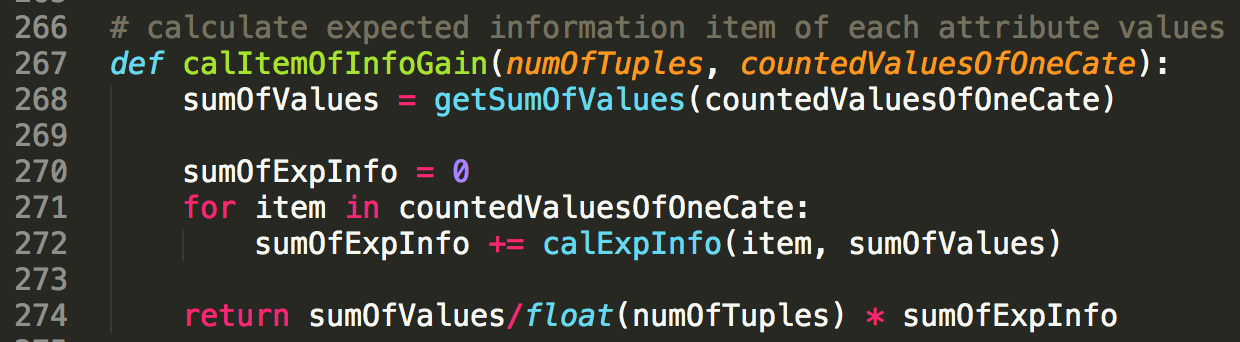


Then, we should count the expected information of each attribute that is the value in the red square.

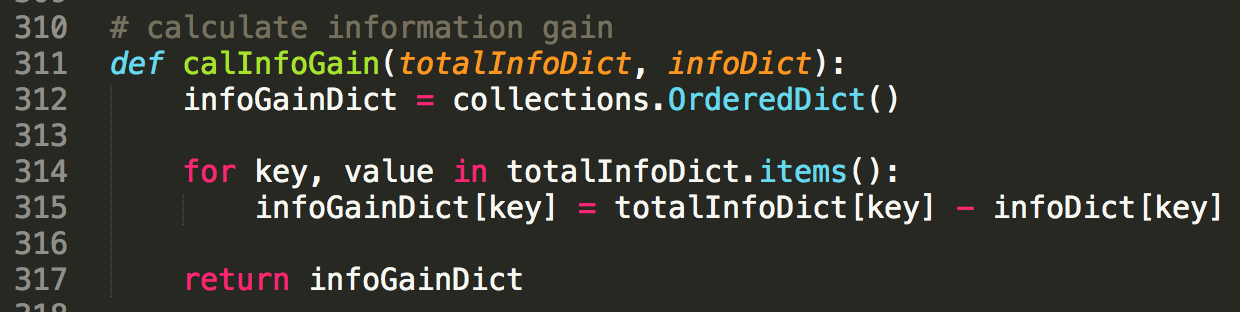




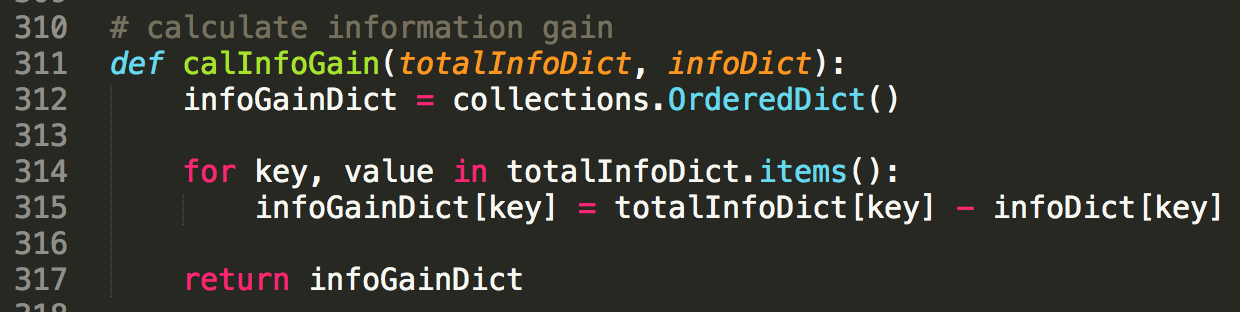
‘calInfoDict’ function will do that. When calculate this, we need to calculate the each item for every attribute value. That values are the items in the blue box.



That things will be done by ‘calItemOfInfoGain’ function. It uses ‘calExpInfo’ function as well.

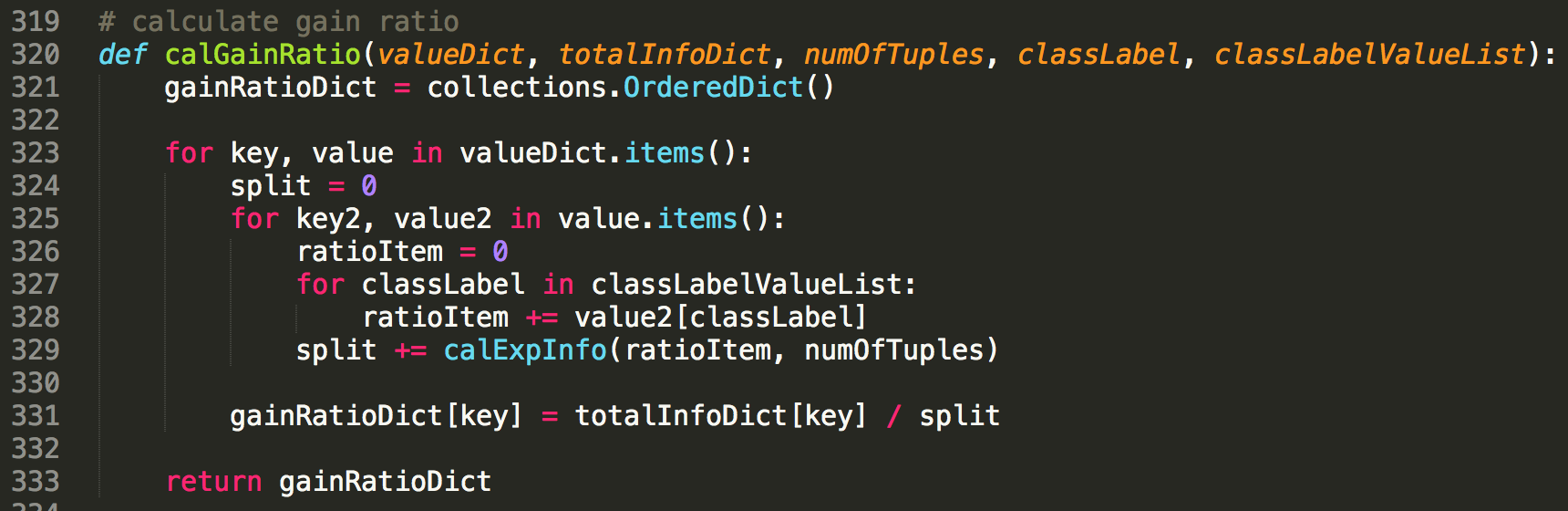


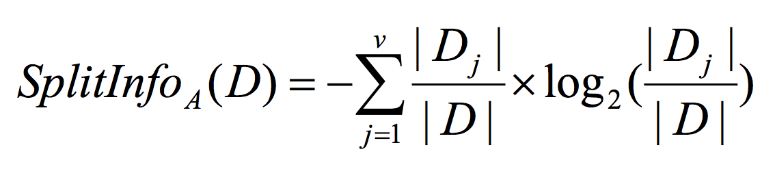
After these process, we can get the information gain easily. Because we already calculate every value for information gain. It is just subtracting ‘infoDict’ from ‘totalInfoDict’.





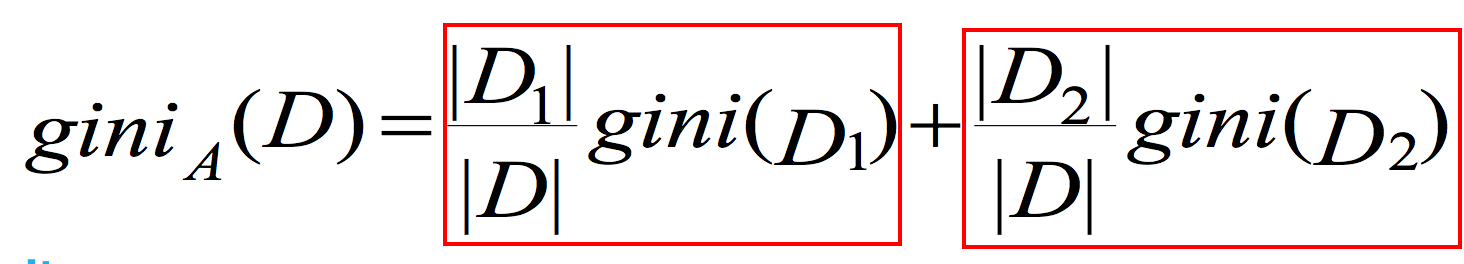
Next, I also used gain ratio. We can get the Gain ratio by the above formula. We already have Information Gain. So we just need is calculating split information of attribute.

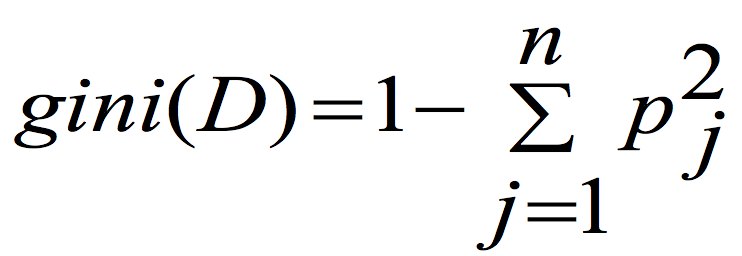


‘

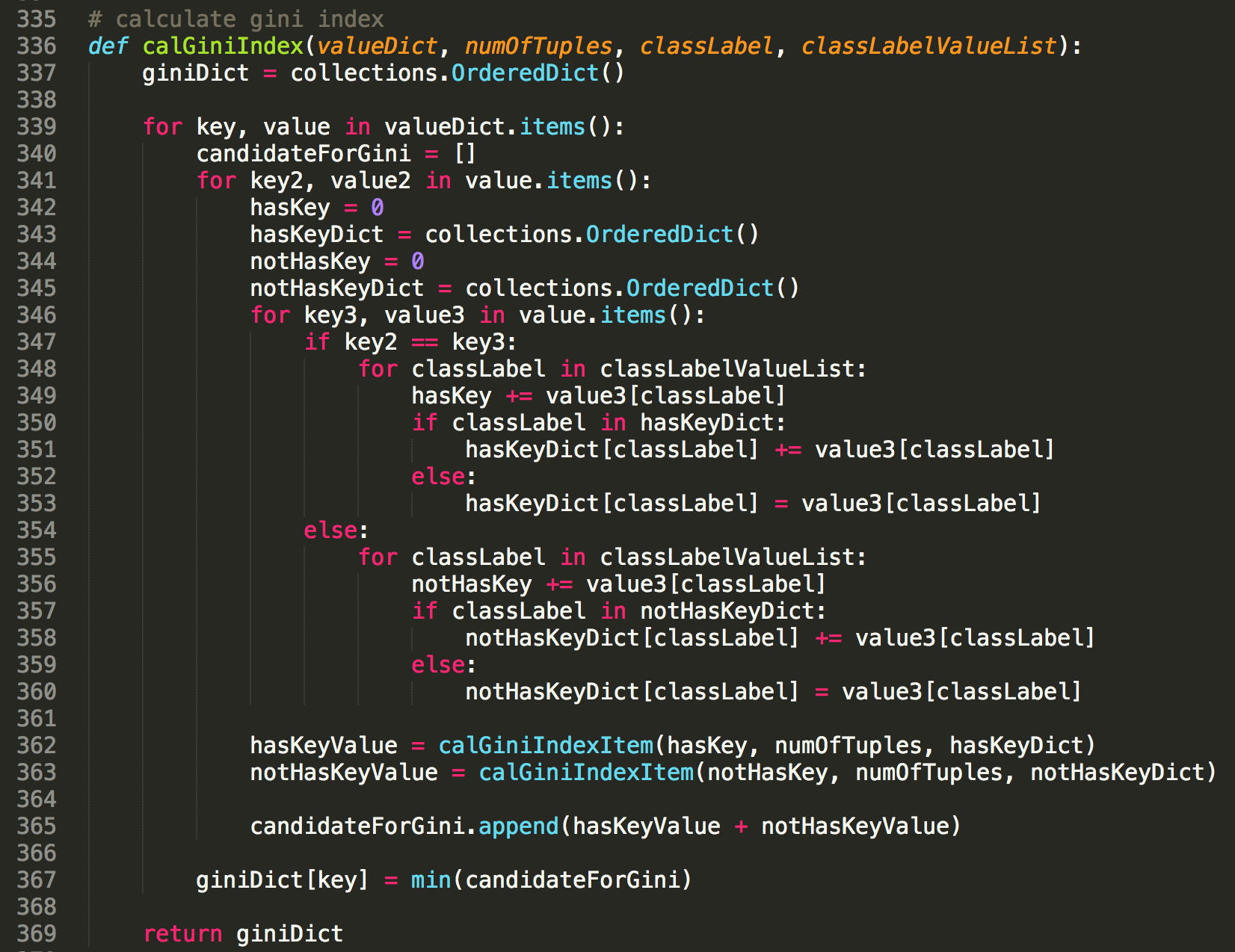
This formula is for split information. We know it is the same the return value of ‘calExpInfo’ function! So, I reused that function. This method uses value dictionary as well.

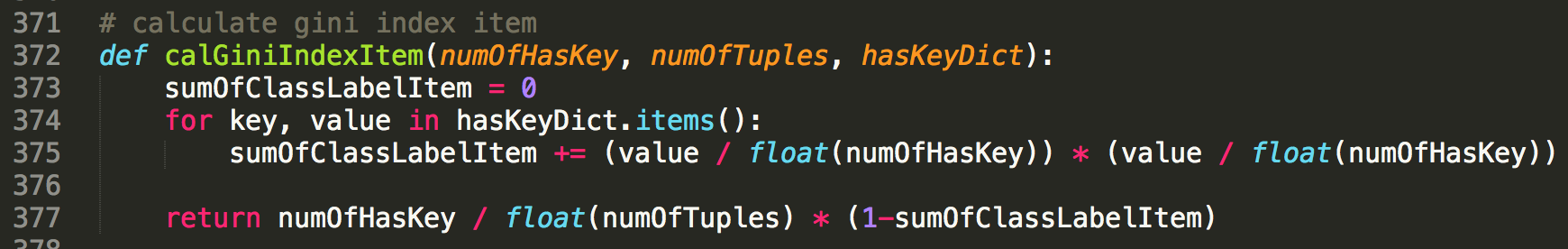
At last, there is Gini index. In Gini index case, we need to calculate Gini Index for every attribute values. After comparing them, we should pick the smallest Gini index. That is the Gini index represented the attribute.



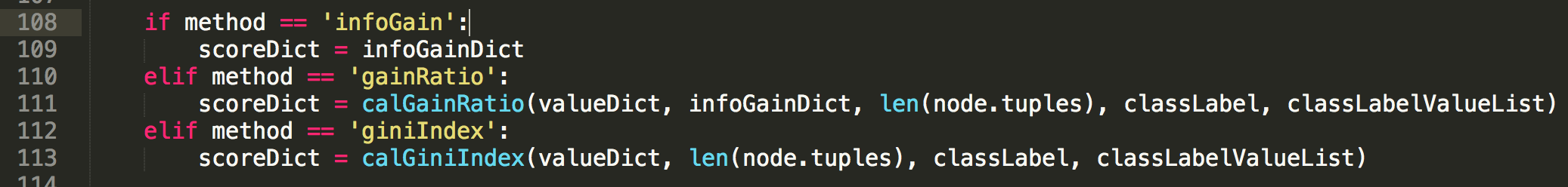


And, for Gini index, we have to split the tuples in two group. One group that has certain attribute value, and another that doesn’t have certain attribute value. I calculated the value in the red box separately. I call this value ‘gini index item’. ‘calGiniIndexItem’ function will do this.

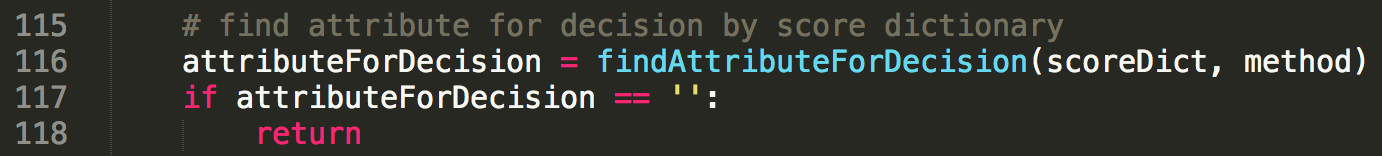


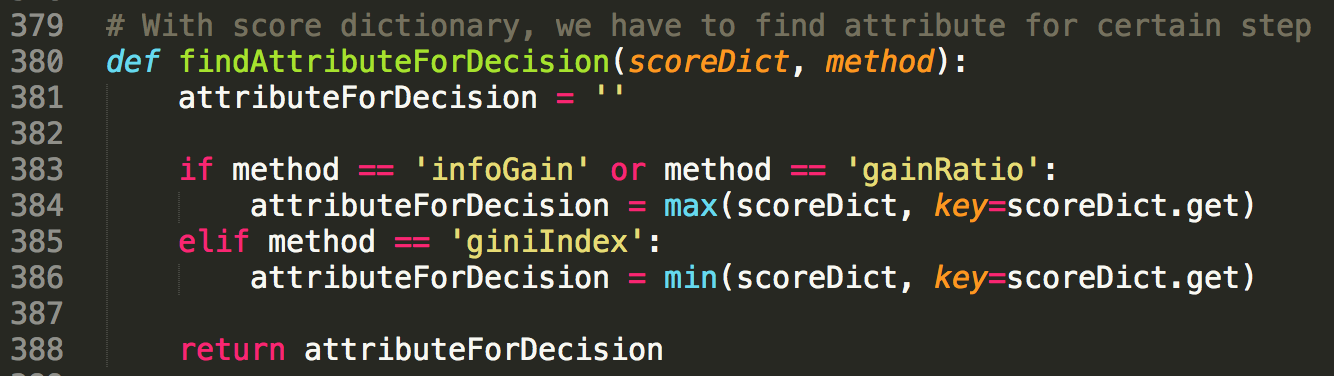


So, we have every measure now! Then, we need to decide the next attribute by these values. I call this index ‘score dictionary.

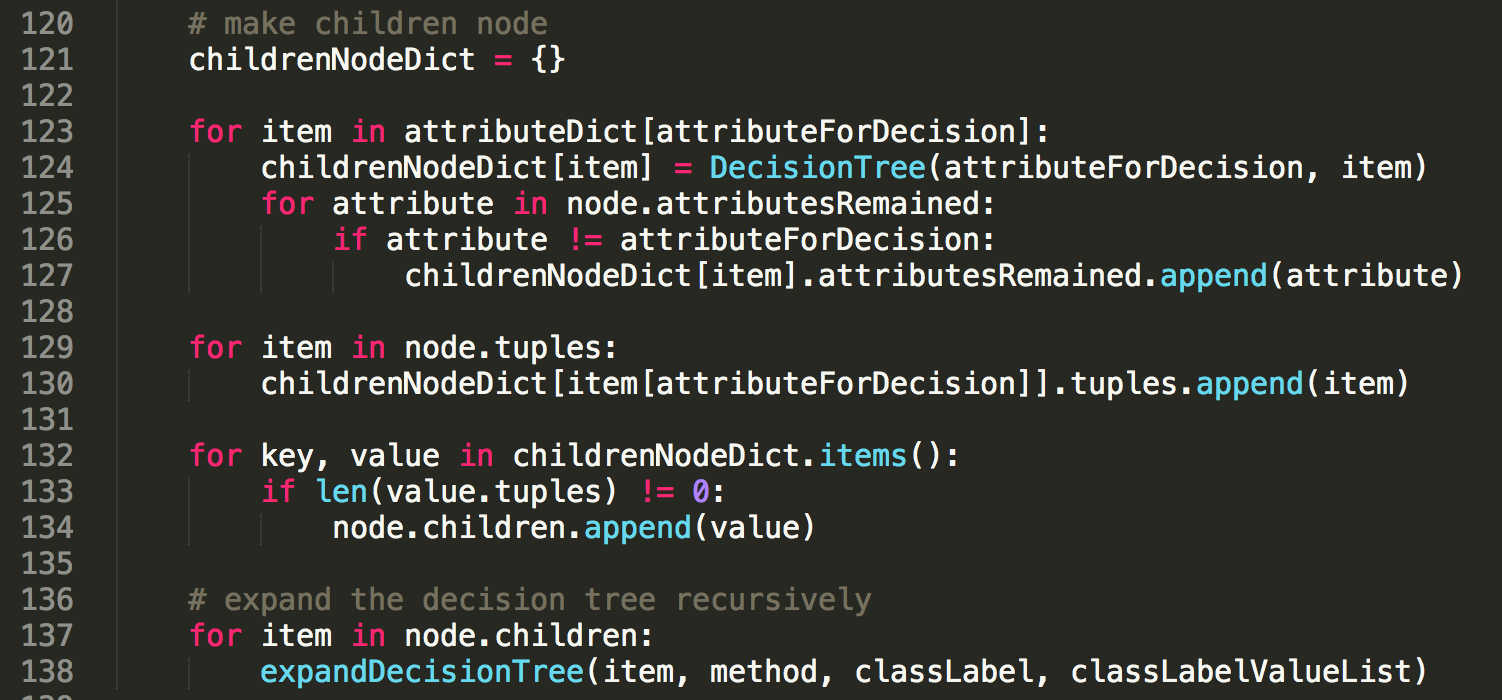


By method name, the program chooses the way to calculate score dictionary.



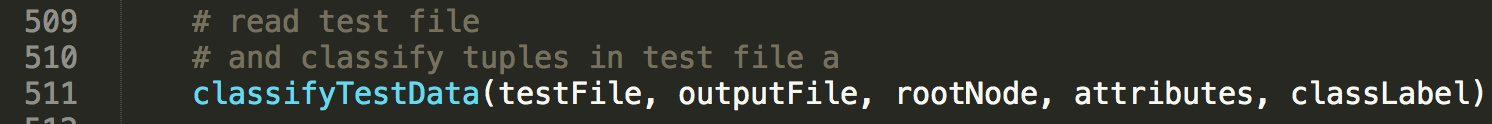


With this score dictionary, we can find the attribute for decision. If method is Information gain or Gain ratio, we have to choose the attribute with the highest score. If method is Gini index, we have to choose the attribute with the lowest score.



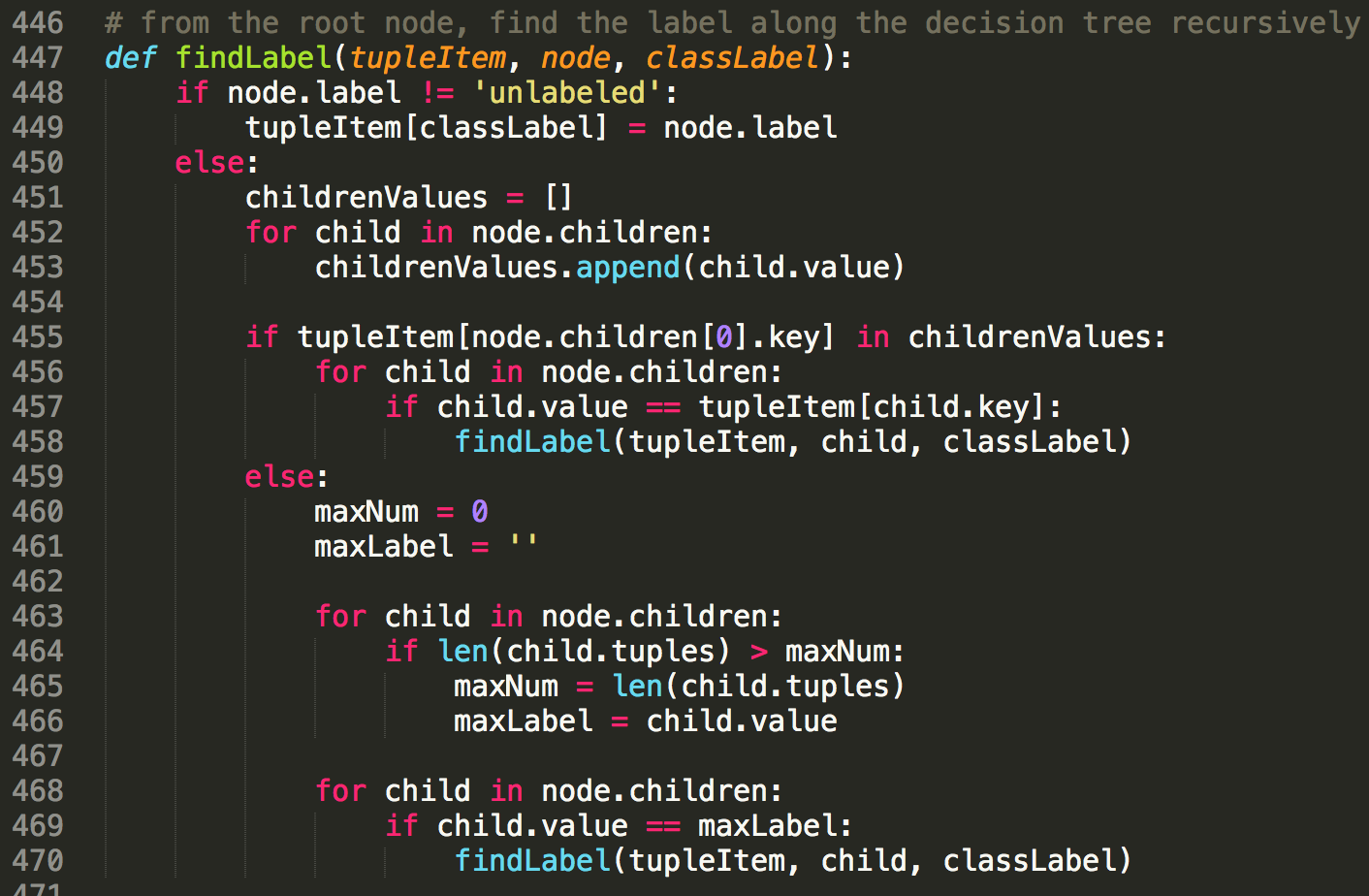
Finally, we decide the next attribute for decision! After this, we have to expand the tree with this attribute. The program generates the children node. The number of the children node is same with the value in the selected attribute. Children node take the ‘attributesRemained’ except the attribute of the parent node. And the tuples will be distributed into the correct children node. If there is no tuple in a child node, that node will be deleted from the list of the children node. And then, ‘expandDecisionTree’ method will be run recursively. This is the end of building the decision tree.

### Classifying test data





Let’s classify the test data. Read the test file. And save it forms of the list of dictionaries like the ‘tupleList’. And then, the data put it into ‘findLabel’ function one by one. It gives the appropriate label to test tuple items. After getting the label, the program writes the data of test tuples into the output file.



This is the function ‘findLabel’. This function is run recursively as well. The tuple follows the tree from the root node along the children node. If there is a child node that has the same attribute value with the attribute value of the tuple, the tuple will go to that child node. If there is not that child node, the tuple will choose the child node that has the biggest number of the tuples. If the label of the node is not ‘unlabeled’, the tuple is labeled with the label of the current node, and returns.

### Closing files



At the end of the program, it closes the files.

# 4. Instructions for executing the source code

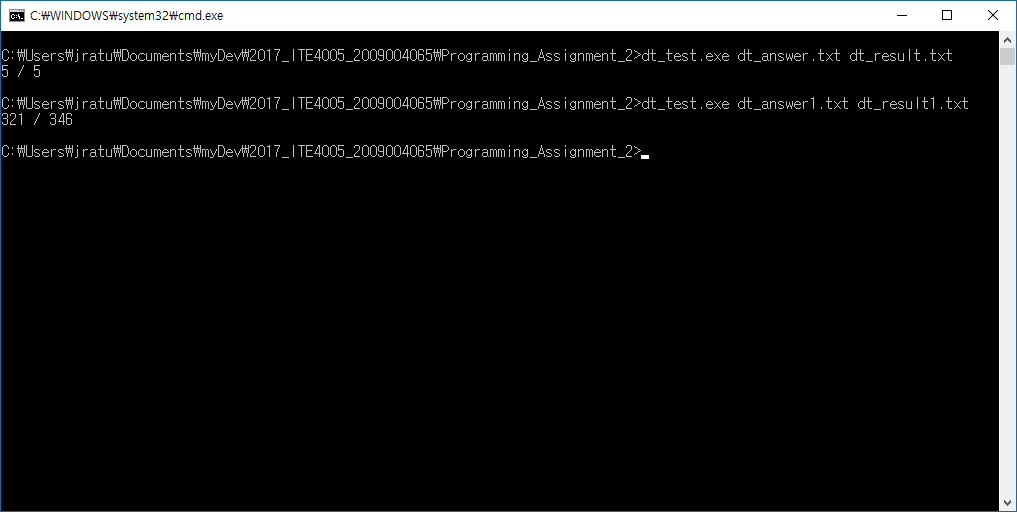
I used OS X, and python. So, you don’t need to compile it into exe file. You can execute it just by python command.



Just input this command in the directory that has dt.py file, training file, and test file.

$ python dt.py [training\_file\_name] [test\_file\_name] [output\_file\_name]

# 5. Result of test



This program has 100% accuracy (5 / 5) for dt\_test.txt file, and 92.77%(321 / 346) for dt\_test.txt file.