Data Science [ITE4005]

# Programming Assignment #2

: Build a decision tree, and classify the test set

2015004120<br>Software Engineering<br>Park Seonha

## 0. Environment

- OS : Windows
- Language : Python3 (version : 3.4.4)
- Executable file : dt.py (../Programming_Assignment_2/project_dt/dt.py)

## 1. Summary of Algorithm

This programming assignment's goal if "build a decision tree, and classify the test set using it." And, it can be seperated "generate a decision tree" and "get test set and classify with decision tree"

I used gain ratio to select proper attribute to maximize the decrement of entropy, and generated tree recursively with classified in each node.

Generating tree, check the node's attribute name list or attribute list is empty, and if it is, return majority class in that node. And, if all values are in one same class, return that class. If node is not in that case, calculate gain ratio and select proper attribute and make child tree. Child tree has all classified tuples in parent tree.

To calculate Gain ratio, first, calculate expected information and residual information in dataset.

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i) \quad Info_a(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

and get information gain using $Info(D)$ and $Info_a(D)$. Information gain $Gain(A)$ is

$$Gain(A) = Info(D) - Info_a(D)$$

Since this gain is biased towards attributes with large amount of values, to normalize this gain to gain ratio, calculate splitinfo.

$$SplitInfo_a(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2 \frac{|D_j|}{|D|} \ , \ GainRatio(A) = Gain(A)/SplitInfo(A)$$

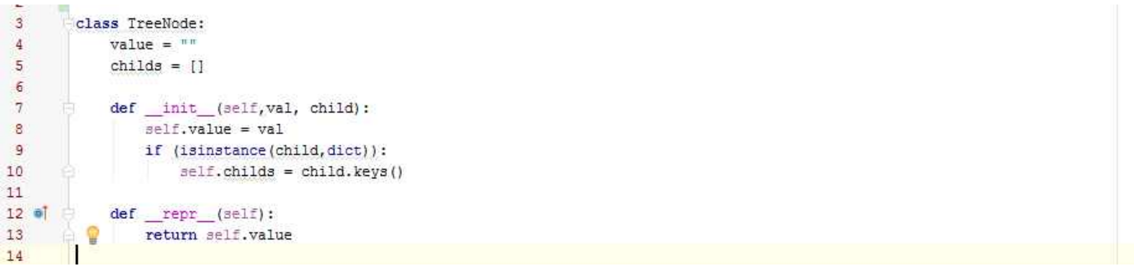
Choose the attribute which has the maximum gain ratio, and classify the tuples with that attribute.

After generate whole decision tree, get test tuples and trace the tree during the answer is found or tree is end and can't get answer. If tracing finished and can't get answer, find the label which has largest value.

## 2. Description of codes

This assignment program has 3 source code files. “DecisionTree.py”, “TreeNode.py”, “dt.py”. “dt.py” has the main routine of this program.

### Treenode.py

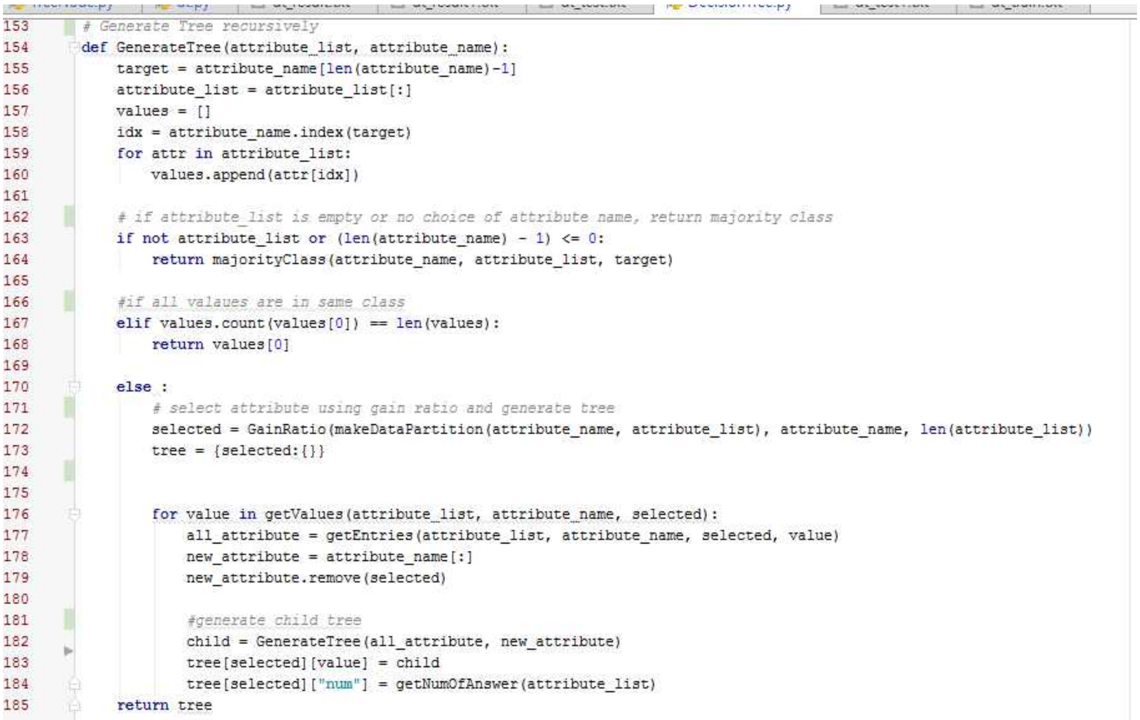


```
class TreeNode:
    value = ""
    childs = []

    def __init__(self,val, child):
        self.value = val
        if (isinstance(child,dict)):
            self.childs = child.keys()

    def __repr__(self):
        return self.value

```

Treenode.py has a class named TreeNode. As its name shows, this contains each node’s attribute name(value) and list of child node(childs). __init__(self, val, child) is initializer of this class, __repr__(self) returns its attribute name.

### DecisionTree.py

This file is implementing the method which generates tree recursively and functions need for implementing that method.

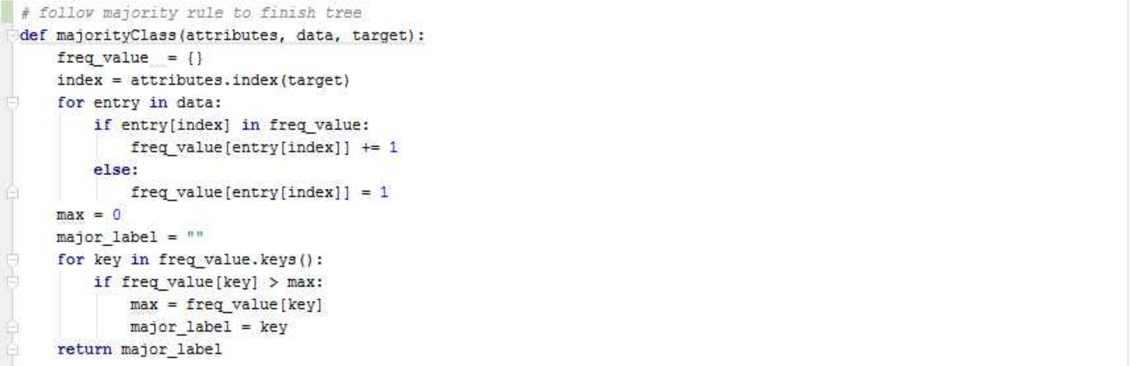


```
# Generate Tree recursively
def GenerateTree(attribute_list, attribute_name):
    target = attribute_name[len(attribute_name)-1]
    attribute_list = attribute_list[:]
    values = []
    idx = attribute_name.index(target)
    for attr in attribute_list:
        values.append(attr[idx])

    # if attribute_list is empty or no choice of attribute name, return majority class
    if not attribute_list or (len(attribute_name) - 1) <= 0:
        return majorityClass(attribute_name, attribute_list, target)

    #if all valaues are in same class
    elif values.count(values[0]) == len(values):
        return values[0]

    else :
        # select attribute using gain ratio and generate tree
        selected = GainRatio(makeDataPartition(attribute_name, attribute_list), attribute_name, len(attribute_list))
        tree = {selected:{}}


        for value in getValues(attribute_list, attribute_name, selected):
            all_attribute = getEntries(attribute_list, attribute_name, selected, value)
            new_attribute = attribute_name[:]
            new_attribute.remove(selected)

            #generate child tree
            child = GenerateTree(all_attribute, new_attribute)
            tree[selected][value] = child
            tree[selected]["num"] = getNumOfAnswer(attribute_list)
    return tree
```

GenerateTree(attribute_list, attribute_name) is the function which generate tree recursively. In this Function, tree is generated in dictionary(key, value) form. Its terminal condition is ○ attribute list (list of all tuples) is empty or attribute name list is empty so can’t choice next node ○ all tuples are in same class. In other case, calculate gain ratio and generate sub-tree with its attribute list and attribute name list except selected just before.

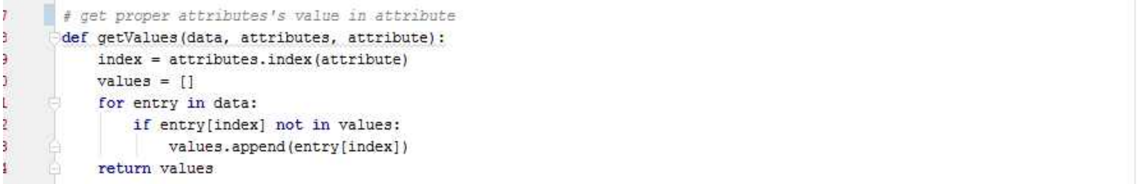
After generate sub-tree, append the key named ‘num’ which has the number of each class label in that node’s attribute list.

To implement this function, I implemented functions majorityClass(attributes, data, target), GainRatio(D,attribute_name, total_attr), getValues(data, attributes, attribute), getEntries(data, attributes, selected, val), getNumOfAnswer(attribute_list).



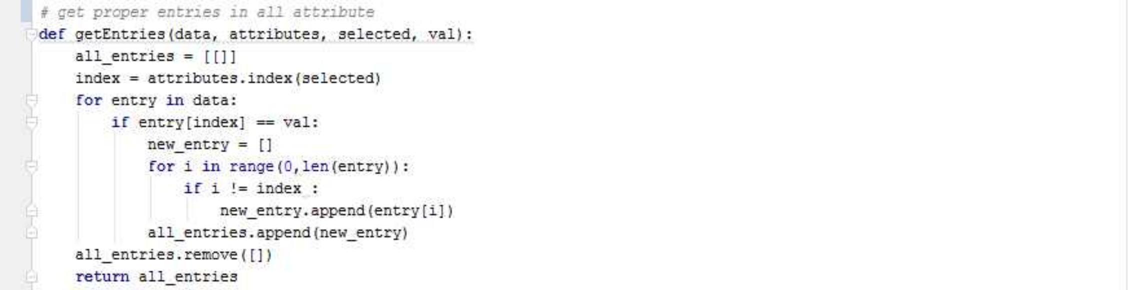
```
# follow majority rule to finish tree
def majorityClass(attributes, data, target):
    freq_value  = {}
    index = attributes.index(target)
    for entry in data:
        if entry[index] in freq_value:
            freq_value[entry[index]] += 1
        else:
            freq_value[entry[index]] = 1
    max = 0
    major_label = ""
    for key in freq_value.keys():
        if freq_value[key] > max:
            max = freq_value[key]
            major_label = key
    return major_label
```

In majorityClass(attribute, data, target) function, get all class label attribute[index]’s number in parameter data, find most frequent class label and return it.



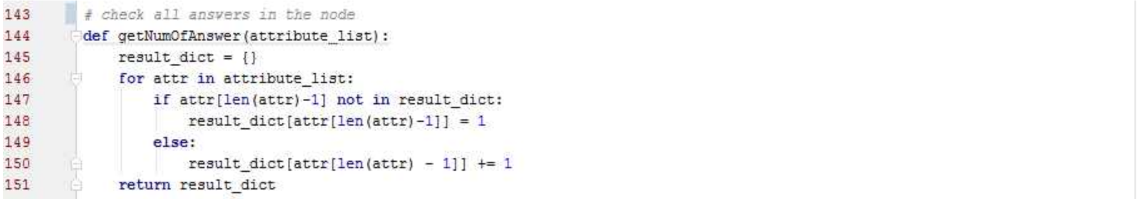
```
# get proper attributes's value in attribute
def getValues(data, attributes, attribute):
    index = attributes.index(attribute)
    values = []
    for entry in data:
        if entry[index] not in values:
            values.append(entry[index])
    return values
```

getValues(data, attributes, attribute) is the function to get class label in specific attribute attributes[index]. It returns like [‘high’, ‘med’, ‘low’], [‘unacc’,‘acc’,‘good’, vgood]



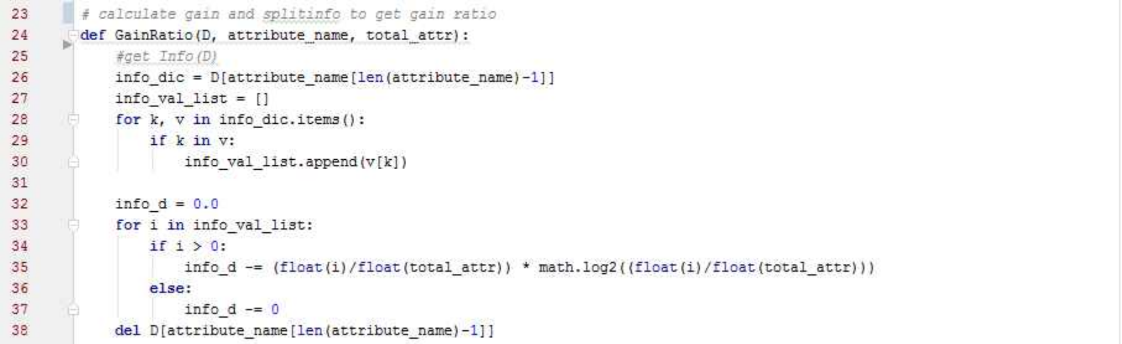
```
# get proper entries in all attribute
def getEntries(data, attributes, selected, val):
    all_entries = [[]]
    index = attributes.index(selected)
    for entry in data:
        if entry[index] == val:
            new_entry = []
            for i in range(0,len(entry)):
                if i != index :
                    new_entry.append(entry[i])
            all_entries.append(new_entry)
    all_entries.remove([])
    return all_entries
```

getEntries(data, attributes, selected, val) is the function to get all proper entries in data. It returns like [‘high’,‘no’,‘excellent’,‘yes’]



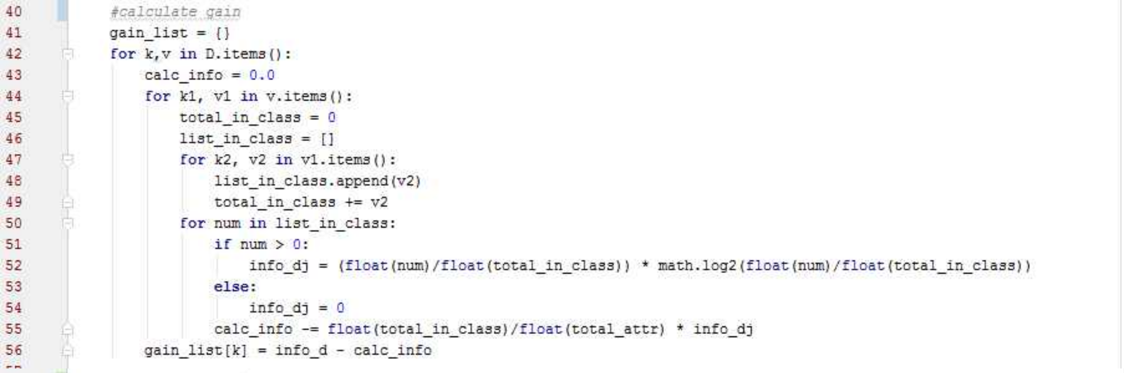
```
# check all answers in the node
def getNumOfAnswer(attribute_list):
    result_dict = {}
    for attr in attribute_list:
        if attr[len(attr)-1] not in result_dict:
            result_dict[attr[len(attr)-1]] = 1
        else:
            result_dict[attr[len(attr) - 1]] += 1
    return result_dict
```

getNumOfAnswer(attribute_list) makes dictionary whose key is last class label like [‘yes’,‘no’], [‘unacc’,‘acc’,‘good’, vgood], and value is the number of each class label. The result of this function will be append in tree as {‘num’: {‘yes’:3, ‘no’:2}}

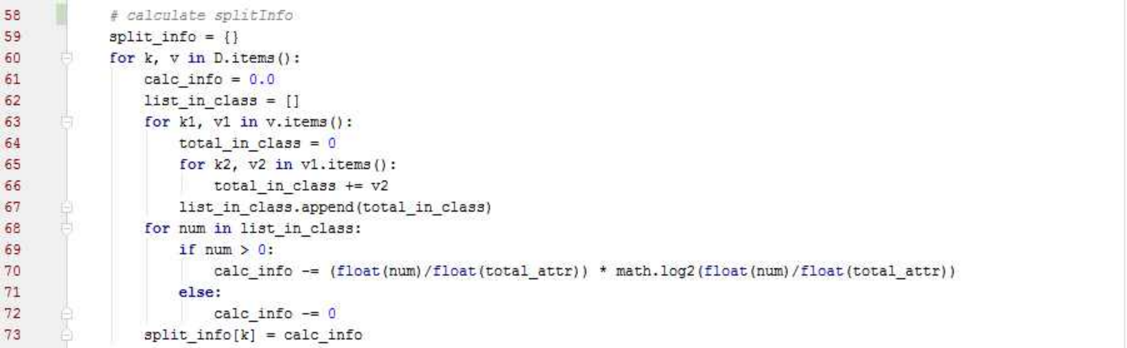


```
# calculate gain and splitinfo to get gain ratio
def GainRatio(D, attribute_name, total_attr):
    #get Info(D)
    info_dic = D[attribute_name[len(attribute_name)-1]]
    info_val_list = []
    for k, v in info_dic.items():
        if k in v:
            info_val_list.append(v[k])

    info_d = 0.0
    for i in info_val_list:
        if i > 0:
            info_d -= (float(i)/float(total_attr)) * math.log2((float(i)/float(total_attr)))
        else:
            info_d -= 0
    del D[attribute_name[len(attribute_name)-1]]
```

GainRatio(D, attribute_name, total_attr) is the function which calculate gain ratio with data partition D, attribute name list, and the number of total attribute. This function returns attribute name which has the maximum gain ratio. So it has several calculation part. First, calculate $Info(D)$.

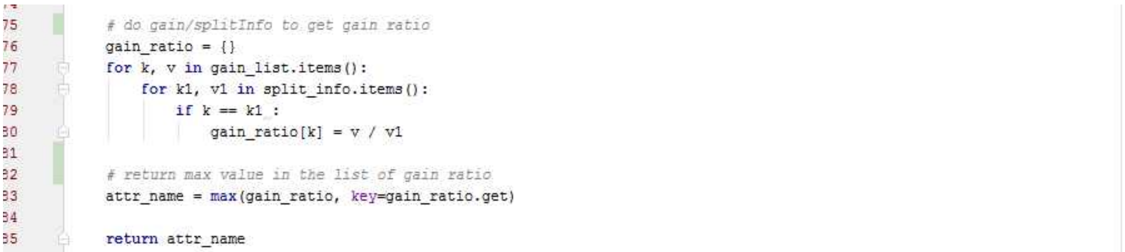


```
    #calculate gain
    gain_list = {}
    for k,v in D.items():
        calc_info = 0.0
        for k1, v1 in v.items():
            total_in_class = 0
            list_in_class = []
            for k2, v2 in v1.items():
                list_in_class.append(v2)
                total_in_class += v2
            for num in list_in_class:
                if num > 0:
                    info_dj = (float(num)/float(total_in_class)) * math.log2(float(num)/float(total_in_class))
                else:
                    info_dj = 0
                calc_info -= float(total_in_class)/float(total_attr) * info_dj
        gain_list[k] = info_d - calc_info
```



```
    # calculate splitInfo
    split_info = {}
    for k, v in D.items():
        calc_info = 0.0
        list_in_class = []
        for k1, v1 in v.items():
            total_in_class = 0
            for k2, v2 in v1.items():
                total_in_class += v2
            list_in_class.append(total_in_class)
        for num in list_in_class:
            if num > 0:
                calc_info -= (float(num)/float(total_attr)) * math.log2(float(num)/float(total_attr))
            else:
                calc_info -= 0
        split_info[k] = calc_info
```

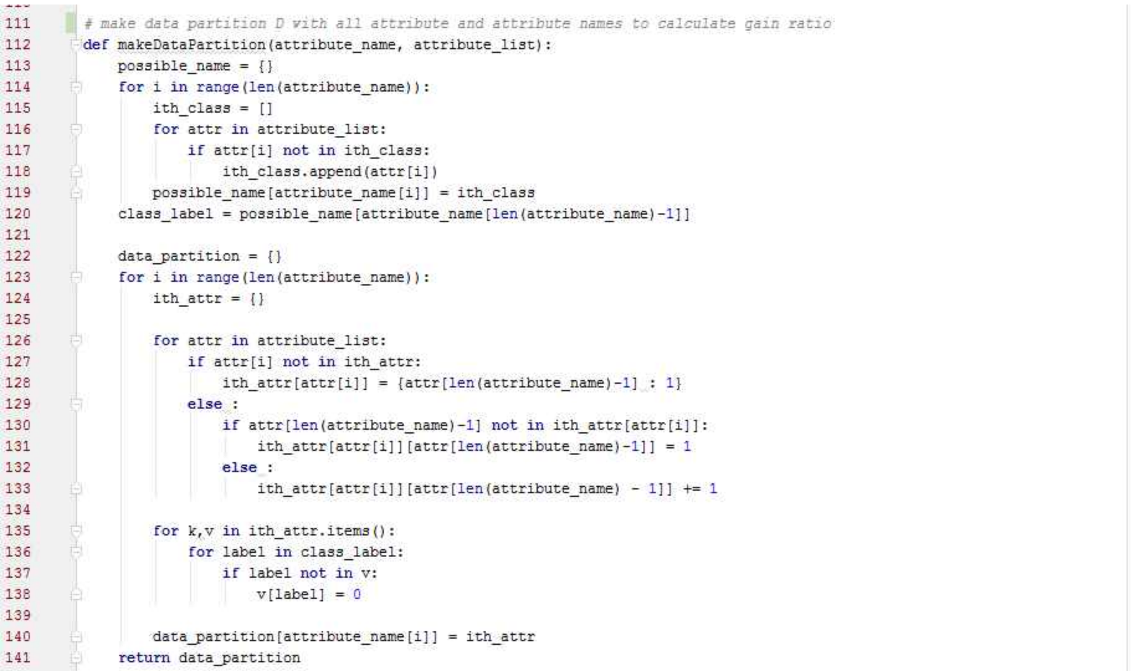
Make the dictionary gain_list whose key is attribute name and value is the gain of each attribute. Same way, calculate splitInfo too.



```
    # do gain/splitInfo to get gain ratio
    gain_ratio = {}
    for k, v in gain_list.items():
        for k1, v1 in split_info.items():
            if k == k1 :
                gain_ratio[k] = v / v1

    # return max value in the list of gain ratio
    attr_name = max(gain_ratio, key=gain_ratio.get)

    return attr_name
```

Bind two dictionary as gain_ratio. find same key in gain_list and split_info, calculate gain ratio and insert the dictionary gain_ratio. Find max value of gain ratio, get its attribute name, and return it.

To implement this function, we need to make data partition D.

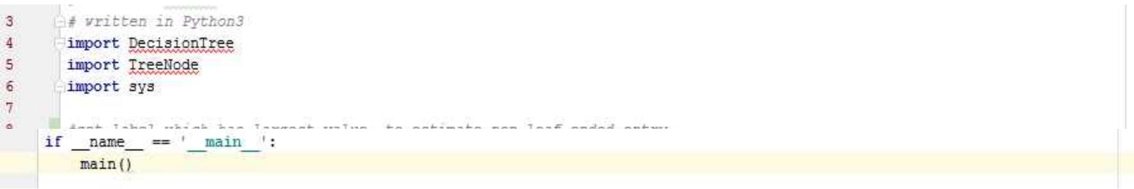


```
# make data partition D with all attribute and attribute names to calculate gain ratio
def makeDataPartition(attribute_name, attribute_list):
    possible_name = {}
    for i in range(len(attribute_name)):
        ith_class = []
        for attr in attribute_list:
            if attr[i] not in ith_class:
                ith_class.append(attr[i])
        possible_name[attribute_name[i]] = ith_class
    class_label = possible_name[attribute_name[len(attribute_name)-1]]

    data_partition = {}
    for i in range(len(attribute_name)):
        ith_attr = {}

        for attr in attribute_list:
            if attr[i] not in ith_attr:
                ith_attr[attr[i]] = {attr[len(attribute_name)-1] : 1}
            else :
                if attr[len(attribute_name)-1] not in ith_attr[attr[i]]:
                    ith_attr[attr[i]][attr[len(attribute_name)-1]] = 1
                else :
                    ith_attr[attr[i]][attr[len(attribute_name) - 1]] += 1

        for k,v in ith_attr.items():
            for label in class_label:
                if label not in v:
                    v[label] = 0

        data_partition[attribute_name[i]] = ith_attr
    return data_partition
```

makeDataPartition(attribute_name, attribute_list) do that role in this program. Find the number of all class label in each attribute. It returns dictionary like {'credit_rating': {'fair': {'no': 0, 'yes': 3}, 'excellent': {'no': 2, 'yes': 0}}, 'income': {'medium': {'no': 1, 'yes': 2}, 'low': {'no': 1, 'yes': 1}}, 'student': {'no': {'no': 1, 'yes': 1}, 'yes': {'no': 1, 'yes': 2}}, 'Class:buys_computer': {'no': {'no': 2, 'yes': 0}, 'yes': {'no': 0, 'yes': 3}}}

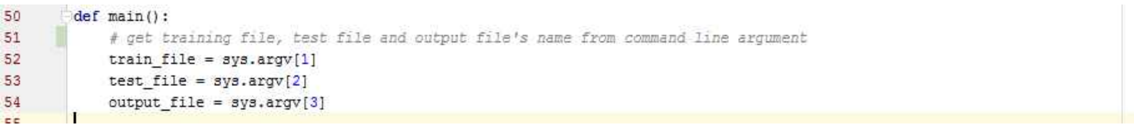
**dt.py**

This file has the main routine of program like file I/O, test program.



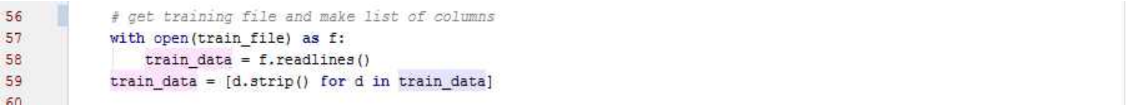
```
# written in Python3
import DecisionTree
import TreeNode
import sys

if __name__ == '__main__':
    main()
```

It includes other two file and sys module, and main() function is in this file.



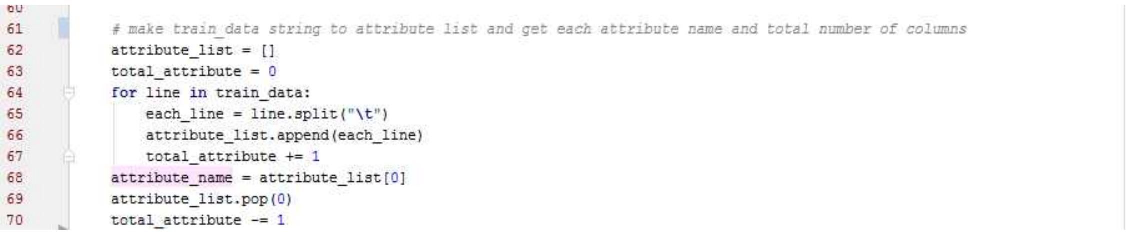
```
def main():
    # get training file, test file and output file's name from command line argument
    train_file = sys.argv[1]
    test_file = sys.argv[2]
    output_file = sys.argv[3]
```

In main function, get command line argument using sys module.



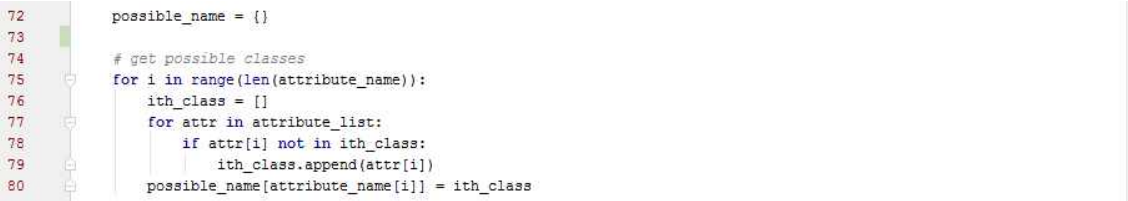
```
    # get training file and make list of columns
    with open(train_file) as f:
        train_data = f.readlines()
    train_data = [d.strip() for d in train_data]
```

open training dataset file.



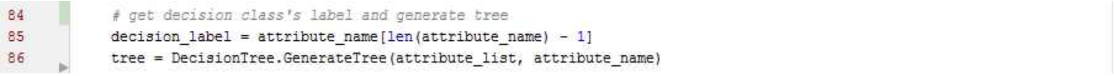
```python
# make train_data string to attribute list and get each attribute name and total number of columns
attribute_list = []
total_attribute = 0
for line in train_data:
    each_line = line.split("\t")
    attribute_list.append(each_line)
    total_attribute += 1
attribute_name = attribute_list[0]
attribute_list.pop(0)
total_attribute -= 1
```

attribute_list is the list of entry tuples. Tuples are splited in list. while generating attribute_list, count total number of attribute too.



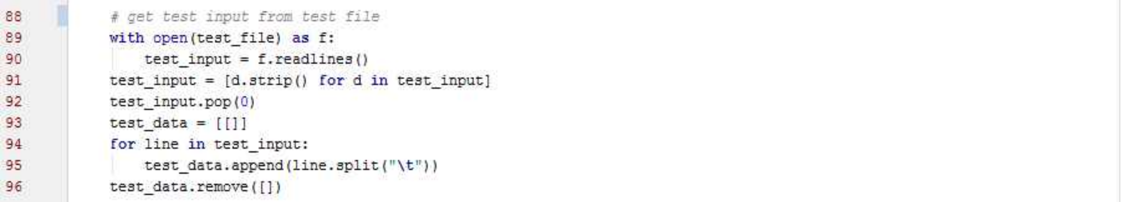
```python
possible_name = {}

# get possible classes
for i in range(len(attribute_name)):
    ith_class = []
    for attr in attribute_list:
        if attr[i] not in ith_class:
            ith_class.append(attr[i])
    possible_name[attribute_name[i]] = ith_class
```

possible name has each attribute and class label. It contains like {'age': ['<=30', '31...40', '>40'], 'credit_rating': ['fair', 'excellent'], 'Class:buys_computer': ['no', 'yes'], 'student': ['no', 'yes'], 'income': ['high', 'medium', 'low']}



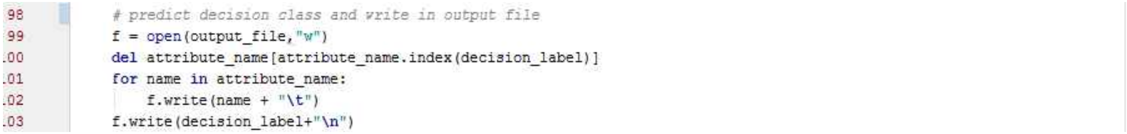
```python
# get decision class's label and generate tree
decision_label = attribute_name[len(attribute_name) - 1]
tree = DecisionTree.GenerateTree(attribute_list, attribute_name)
```

decision_label is the attribute name to predict in testset. and tree is the dictionary which has decision tree.



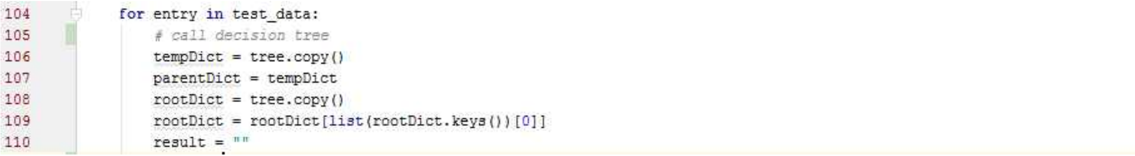
```python
# get test input from test file
with open(test_file) as f:
    test_input = f.readlines()
test_input = [d.strip() for d in test_input]
test_input.pop(0)
test_data = [[]]
for line in test_input:
    test_data.append(line.split("\t"))
test_data.remove([])
```

open test_file and get test input in list.



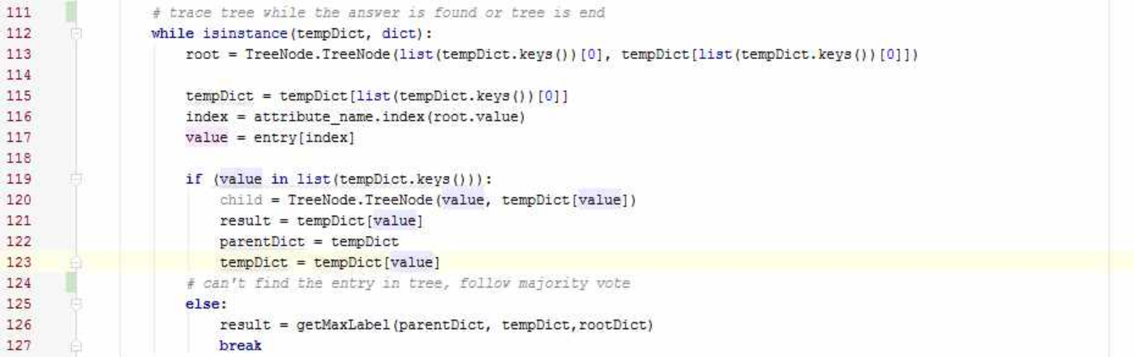
```python
# predict decision class and write in output file
f = open(output_file,"w")
del attribute_name[attribute_name.index(decision_label)]
for name in attribute_name:
    f.write(name + "\t")
f.write(decision_label+"\n")
```

open output file and write test input’s attribute name.



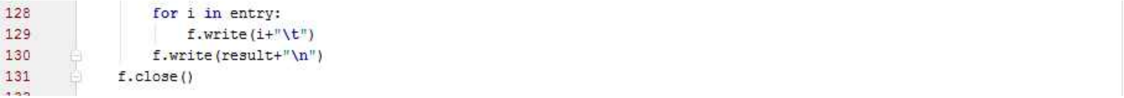
```python
for entry in test_data:
    # call decision tree
    tempDict = tree.copy()
    parentDict = tempDict
    rootDict = tree.copy()
    rootDict = rootDict[list(rootDict.keys())[0]]
    result = ""
```

for each testset tuple, copy decision tree dictionary and make it as TreeNode,



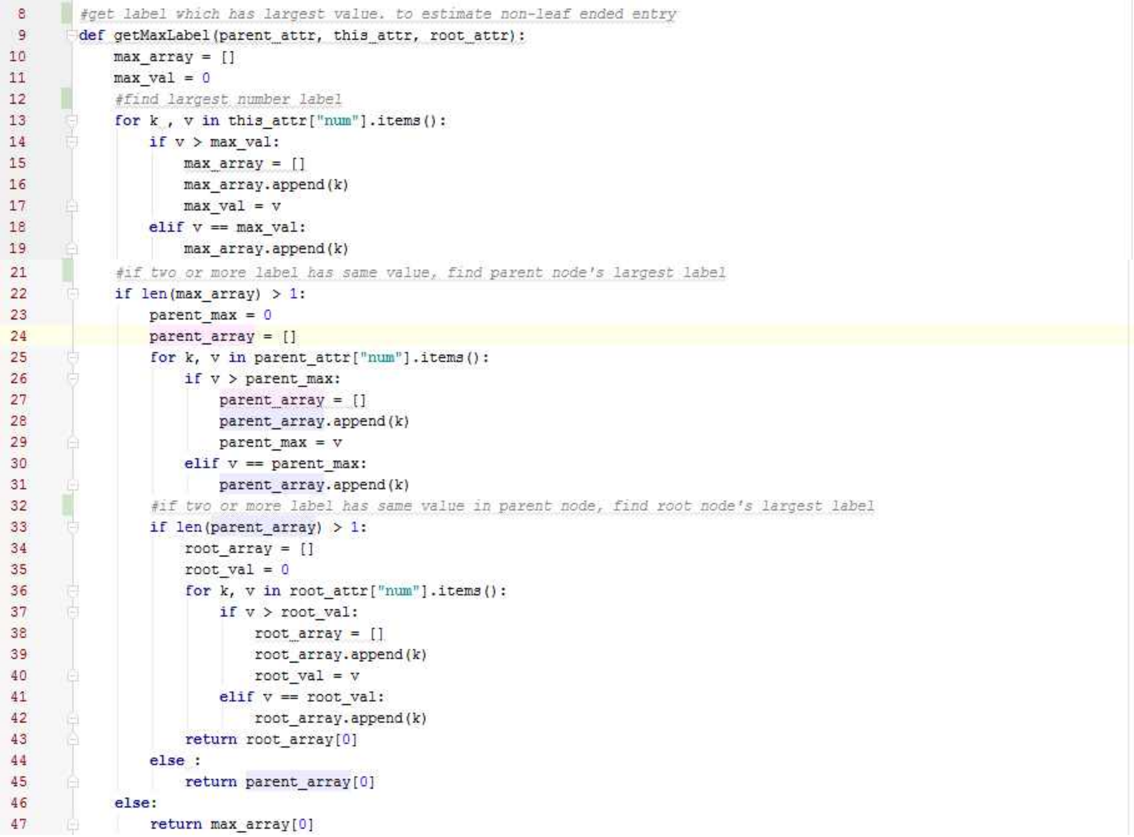
```
        # trace tree while the answer is found or tree is end
        while isinstance(tempDict, dict):
            root = TreeNode.TreeNode(list(tempDict.keys())[0], tempDict[list(tempDict.keys())[0]])

            tempDict = tempDict[list(tempDict.keys())[0]]
            index = attribute_name.index(root.value)
            value = entry[index]

            if (value in list(tempDict.keys())):
                child = TreeNode.TreeNode(value, tempDict[value])
                result = tempDict[value]
                parentDict = tempDict
                tempDict = tempDict[value]
            # can't find the entry in tree, follow majority vote
            else:
                result = getMaxLabel(parentDict, tempDict,rootDict)
                break
```

trace tree while the value is in tree and find the result. Find the attribute value in tree. If it is not exist call the function getMaxLabel(parent_attr, this_attr, root_attr) to choose result.



```
        for i in entry:
            f.write(i+"\t")
        f.write(result+"\n")
    f.close()
```

After choice result, write entry in output file and repeat with next entry. After predict all entry, close output file. The program is end in this part.



```
#get label which has largest value. to estimate non-leaf ended entry
def getMaxLabel(parent_attr, this_attr, root_attr):
    max_array = []
    max_val = 0
    #find largest number label
    for k , v in this_attr["num"].items():
        if v > max_val:
            max_array = []
            max_array.append(k)
            max_val = v
        elif v == max_val:
            max_array.append(k)
    #if two or more label has same value, find parent node's largest label
    if len(max_array) > 1:
        parent_max = 0
        parent_array = []
        for k, v in parent_attr["num"].items():
            if v > parent_max:
                parent_array = []
                parent_array.append(k)
                parent_max = v
            elif v == parent_max:
                parent_array.append(k)
        #if two or more label has same value in parent node, find root node's largest label
        if len(parent_array) > 1:
            root_array = []
            root_val = 0
            for k, v in root_attr["num"].items():
                if v > root_val:
                    root_array = []
                    root_array.append(k)
                    root_val = v
                elif v == root_val:
                    root_array.append(k)
            return root_array[0]
        else :
            return parent_array[0]
    else:
        return max_array[0]
```

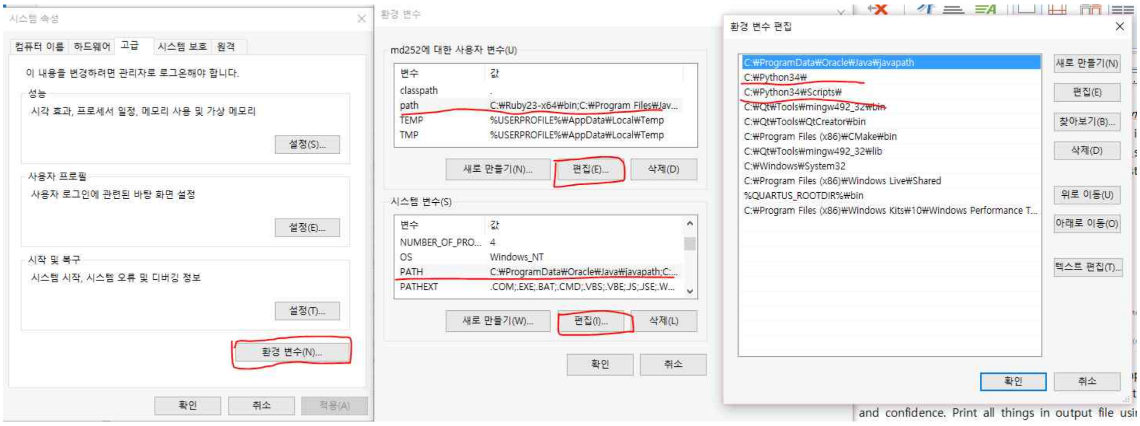
getMaxLabel(parent_attr, this_attr, root_attr) is the function follows the majority rule. use the dictionary whose key is 'num'. and find max value in num. If max value is duplicate, find same thing in parent node's 'num', and if parent node's max value is duplicate, find in root node.

## 3. Instructions for compiling this code

This code is written in Python3 and tested in Python 3.4.4. Since this code has **log2**(to calculate gain ratio), and it only exist in Python3 math module, to run this code, we need python3 interpreter.

We can get python 3.4.4 in ([https://www.python.org/downloads/release/python-344/](https://www.python.org/downloads/release/python-344/))

If python2 is already installed in PC, change path Python2x to Python34 in advanced system settings – environment variables.


시스템 속성

이 내용을 변경하려면 관리자로 로그온해야 합니다.

성능

시각 효과, 프로세서 일정, 메모리 사용 및 가상 메모리

사용자 프로필

사용자 로그인에 관련된 바탕 화면 설정

시작 및 복구

시스템 시작, 시스템 오류 및 디버깅 정보

환경 변수

md252에 대한 사용자 변수(U)

| 변수 | 값 |
|---|---|
| classpath | . |
| path | C:₩Ruby23-x64₩bin;C:₩Program Files₩Jav... |
| TEMP | %USERPROFILE%₩AppData₩Local₩Temp |
| TMP | %USERPROFILE%₩AppData₩Local₩Temp |

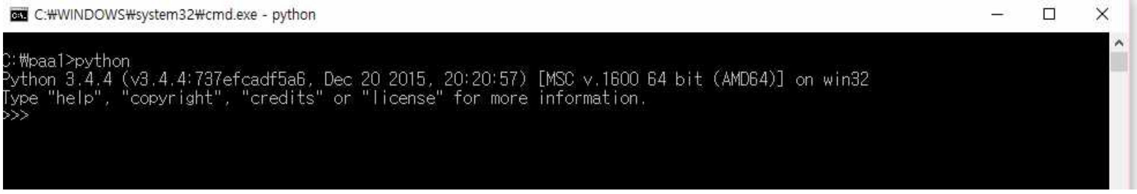
시스템 변수(S)

| 변수 | 값 |
|---|---|
| NUMBER_OF_PRO... | 4 |
| OS | Windows_NT |
| PATH | C:₩ProgramData₩Oracle₩Java₩javapath;C:... |
| PATHEXT | .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.W... |

환경 변수 편집

C:₩ProgramData₩Oracle₩Java₩javapath
C:₩Python34₩
C:₩Python34₩Scripts₩
C:₩Qt₩Tools₩mingw492_32₩bin
C:₩Qt₩Tools₩QtCreator₩bin
C:₩Program Files (x86)₩CMake₩bin
C:₩Qt₩Tools₩mingw492_32₩lib
C:₩Windows₩System32
C:₩Program Files (x86)₩Windows Live₩Shared
%QUARTUS_ROOTDIR%₩bin
C:₩Program Files (x86)₩Windows Kits₩10₩Windows Performance T...

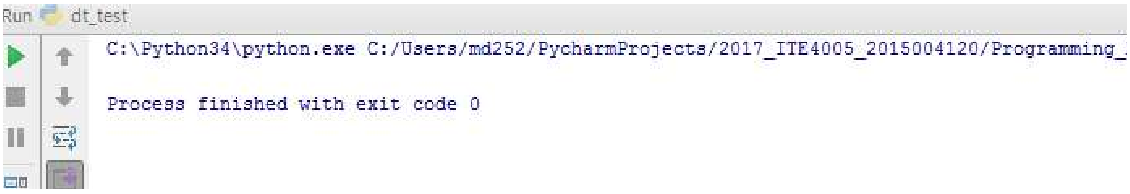
and confidence. Print all things in output file usi


C:₩WINDOWS₩system32₩cmd.exe - python

```
:₩paa1>python
ython 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win32
ype "help", "copyright", "credits" or "license" for more information.
>>
```

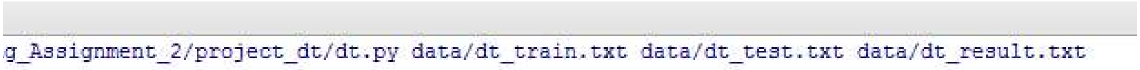
After environment variables setting, we can get python3 default in cmd.

In code directory(python file is in /project_dt), we can run this program on cmd “python dt.py [training data file name] [test set file name] [output file name]”.



```
C:\Python34\python.exe C:/Users/md252/PycharmProjects/2017_ITE4005_2015004120/Programming_

Process finished with exit code 0
```

continue in next picture

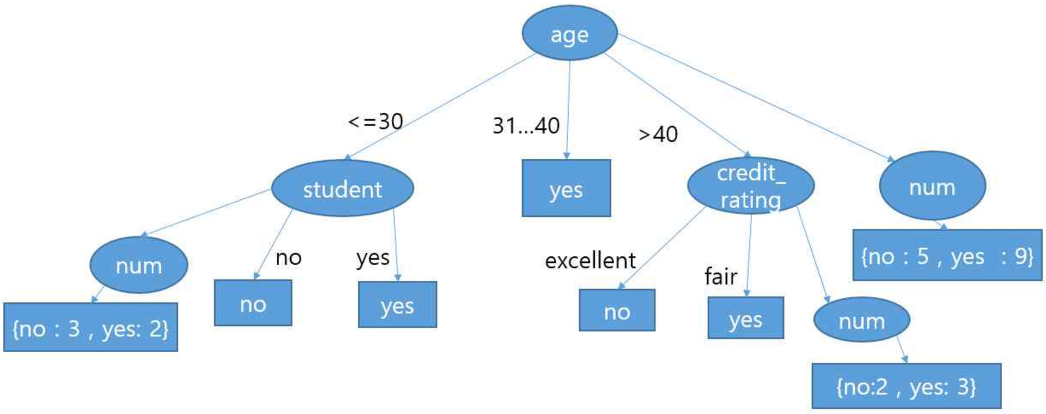


```
g_Assignment_2/project_dt/dt.py data/dt_train.txt data/dt_test.txt data/dt_result.txt
```

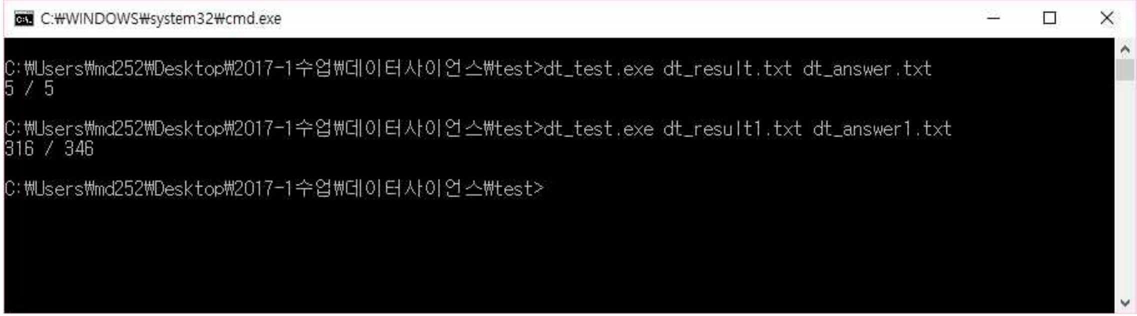
## 4. Other Specifications

I implemented this tree with all attribute node should have the number of its class labels as child of themselves. It was used to classify by majority vote which can't reach leaf node. So tree looks like below.

In dictionary shaped tree, they all have key named 'num' and it contains number of class labels the node has.



tree generated by dt_train.txt



```
C:₩Users₩md252₩Desktop₩2017-1수업₩데이터사이언스₩test>dt_test.exe dt_result.txt dt_answer.txt
5 / 5

C:₩Users₩md252₩Desktop₩2017-1수업₩데이터사이언스₩test>dt_test.exe dt_result1.txt dt_answer1.txt
316 / 346

C:₩Users₩md252₩Desktop₩2017-1수업₩데이터사이언스₩test>
```

the accuracy of this program