

Data Science

programming Assignment 2

Decision Tree

Design Document

2015004011
major in Software
Dohyun Kim

1. Summary of my algorithm

This assignment is to implement the 'Decision Tree'. This algorithm uses information gain of ID3 as an attribute selection measure.

I'll briefly explain the sequence.

1. Read the input file and parse it to make a list for each transaction.
2. using recursive function, Build the decision tree by choosing the best dividing attribute through information gain of ID3
3. if best_gain is greater than zero, Recursively calls the buildtree function with a list of rows divided by attributes with best gain.
4. if best_gain is zero, or When 'attri_rows' with row satisfying current condition is less than 2 or tree level is greater than the number of attributes, return decision_node containing result
5. Assuming that all the trees have been created, find the class label value of the test set using the decision tree.
6. Search the tree and return the correct result.
7. If the tree does not have an appropriate result value, use the resulting value.
8. Add the result to the list of test sets and output it to the specified file according to the format.

2. Detailed description of my code

1. main Function

```
if __name__ == '__main__':
    f_train_name = "/Users/dohuni/Desktop/dt.py/"+sys.argv[1]
    f_test_name = "/Users/dohuni/Desktop/dt.py/"+sys.argv[2]
    f_result_name = "/Users/dohuni/Desktop/dt.py/"+sys.argv[3]

    attri_name, attri_list = pre_treatment(f_train_name)
    test_name, test_rows = pre_treatment(f_test_name)
    tree = buildtree(attri_list, 1)
    #print_tree(tree, attri_name)
    result_rows = test_decision(tree, test_rows)
    print_result_f(f_result_name, attri_name, result_rows)
```

First, we create the file address with three arguments that we executed program. Then use the preprocessing function to make list of attribute's values. We then make a decision tree and use the decision tree to add the class labels of the test set. And outputs the result to the result file. The main function was neatly written so that the algorithm could be seen at a glance.

2. pre_treatment()

```
#This function is preprocessing function that read file and make them into lists
def pre_treatment(file_name):
    f = open(file_name)
    def make_clean(str):
        str = str.strip()
        return str
    file_line = [map(make_clean, line.split('\t')) for line in f]
    attri_name = file_line.pop(0) #att_name list
    attri_list = file_line #att list
    f.close()
    return attri_name, attri_list
```

This function is preprocessing function that read file and make them into lists. `strip()` is a built-in function that removes left and right whitespace. The reason for splitting the 'attribute_name' apart is not necessary when creating the tree, but it is stored separately because it is needed for output. The list of attribute names and the list of remaining attribute values are returned in a tuple.

3. class decision_node

```
#The class that contains the information of node in decision tree.
# 'col' is index of standard attribute
# 'level' is level of decision tree.
# 'value_list' has values of standard attribute
# 'child_list' has list of child nodes
# 'result' has result, If it is a leaf node
# 'rp_result' is result that replace the actual result
class decision_node(object):
    def __init__(self, col=-1, level=None, value_list=None, child_list = None,
        result=None, rp_result = None):
        self.col = col
        self.level = level
        self.value_list = value_list
        self.child_list = child_list
        self.result = result
        self.rp_result = rp_result
```

The decision_node class is a class for creating nodes that make up a decision tree.

- 'col' is index of standard attribute
- 'level' is level of decision tree.
- 'value_list' has values of standard attribute
- 'child_list' has list of child nodes
- 'result' has result, If it is a leaf node
- 'rp_result' is result that replace the actual result

4. dividset()

```
#Check the values in a specific column, and return a list of list of rows with
the same value
def divideset(rows, column, col_values):
    div_rowss = []
    for value in col_values :
        #The function checks that the value of a specific column in the row that
        is entered as a parameter is equal to value.
        divid_func = lambda row: row[column] == value
        t_rows = [row for row in rows if divid_func(row)]
        div_rowss.append(t_rows)
    return div_rowss
```

This function is used to create decision trees. Check the value of a specific column in the rows to create a row list with the same elements as col_values.

divid_func returns a boolean indicating whether value is equal to the value of a specific column in the row.

5. count_class()

```
#Counts the values of the class label respectively.
def count_class(rows):
    result = collections.defaultdict(int)
    for row in rows:
        r = row[len(row)-1]
        result[r] +=1
    return dict(result)
```

This function counts the number of values in the class label. It returns a dictionary with the value name of the class label as key and the number as the value.

6. entropy()

```
#Function to calculate entropy
def entropy(rows):
    from math import log
    log2 = lambda x: log(x)/ log(2)
    results = count_class(rows)
    ent = 0.0
    for r in results:
        p = float(results[r])/len(rows)
        ent -= p*log2(p)
    return ent
```

This function is to obtain the entropy value to obtain the information gain.

7. count_rowss()

```
#This function that counts the total number of rows in a list of rows
def count_rowss(rowss):
    cnt =0
    for rows in rowss:
        cnt += len(rows)
    return cnt
```

This function is used to find the number of rows in the list of rows. In the list of rows according to the attribute value, if number of row is one, it make to return the node having the result immediately.

8.best_result

```
#Returns the attribute value that has the largest number at class lable
def best_result(result_dict):
    result = None
    best_cnt = 0
    for r in result_dict.keys():
        if result_dict[r] > best_cnt:
            best_cnt = result_dict[r]
            result = r
    return result
```

This function returns the attribute value that has the largest number at class lable. This function is used to obtain the result of the detection.

9.buildtree

```
#This recursive function that builds the tree by choosing the best dividing
criteria for the current set
def buildtree(attri_rows, level) :
    final_result = collections.defaultdict(int)
    result_cnt = count_class(attri_rows)

    if len(attri_rows)<2 or level > len(attri_rows[0]) :
        return decision_node(result = result_cnt)

    rp_result = best_result(result_cnt)
    cur_entropy = entropy(attri_rows)

    best_gain = 0.0
    best_col = None
    best_value_list = None
    best_div_list = None

    col_last = len(attri_rows[0])-1
    for col in range(0,col_last):
        col_values = list(set([row[col] for row in attri_rows]))

        #Attribute selection measure is ID3
        div_rowss = divideset(attri_rows, col, col_values)
        if count_rowss(div_rowss) >1 :
            gain = cur_entropy
            div_rowss_cnt = count_rowss(div_rowss)
            for i in range(0,len(div_rowss)):
                p = float(len(div_rowss[i])) / div_rowss_cnt
                gain -= p * entropy(div_rowss[i])
            if gain > best_gain :
                best_gain = gain
                best_col = col
                best_value_list = col_values
                best_div_list = div_rowss

    if best_gain >0:
        node_list = []
        for rows in best_div_list:
            new_node = buildtree(rows, level+1)
            node_list.append(new_node)
        return decision_node(col=best_col, value_list = best_value_list,
            child_list = node_list, rp_result=rp_result)
    else :
        return decision_node(result =count_class(attri_rows))
```

This function is a function that creates a decision tree that is the core function of this program. The attribute selection measure is ID3, So I use information gain. if best_gain is greater than zero, Recursively calls the buildtree function with a list of rows divided by attributes with best gain. if best_gain is zero, or When 'attri_rows' with row satisfying current condition is less than 2 or tree level is greater than the number of attributes, return decision_node containing result

10.test_decision

```
#The function that determines the class label value of the test set
def test_decision(tree,test_rows):
    results_rows = []
    for row in test_rows:
        result = classify(tree,row)
        row.append(result)
        results_rows.append(row)
    return results_rows
```

This function searches the decision tree that has been created and find it have the class label value of the test set.

11. classify

```
#Search the decision tree and return class label value
def classify(tree,test_row):
    if tree.result != None :
        return tree.result.keys()[0]
    else :
        val = test_row[tree.col]
        #If there is a child node that matches value
        if val in tree.value_list:
            for i in range(0,len(tree.value_list)):
                if val == tree.value_list[i]:
                    return classify(tree.child_list[i],test_row)
        #If there is no child node matching value, use 're_result'.
        else:
            return tree.re_result
```

This function search the decision tree and return class label value. This function recursively searches the tree. If the current node is a leaf node, the result value is result of present node. If there is a child node that matched value, The child node is searched. If there is no child node matching value, use 're_result' that replace the real result.

12. print_tree

```
#check the decision tree at consol window
def print_tree(tree, attri_name):
    if tree.result != None:
        print tree.result
    else:
        print attri_name[tree.col] + '?'
        for i in range(0, len(tree.value_list)):
            print ' ' + tree.value_list[i] + '->'
            print_tree(tree.child_list[i], attri_name)
```

It is not directly related to this assignment, but you can see how the tree was created.

13. print_result_f

```
#Functions that print the results to a file
def print_result_f(f_result_name, attri_name, result_rows):
    f = open(f_result_name, 'w')
    str = "\t".join(attri_name) + '\n'
    f.write(str)
    for row in result_rows:
        str = "\t".join(row) + '\n'
        f.write(str)
    f.close()
```

This function prints the result of a test set created by test_decision function to a specific result file.

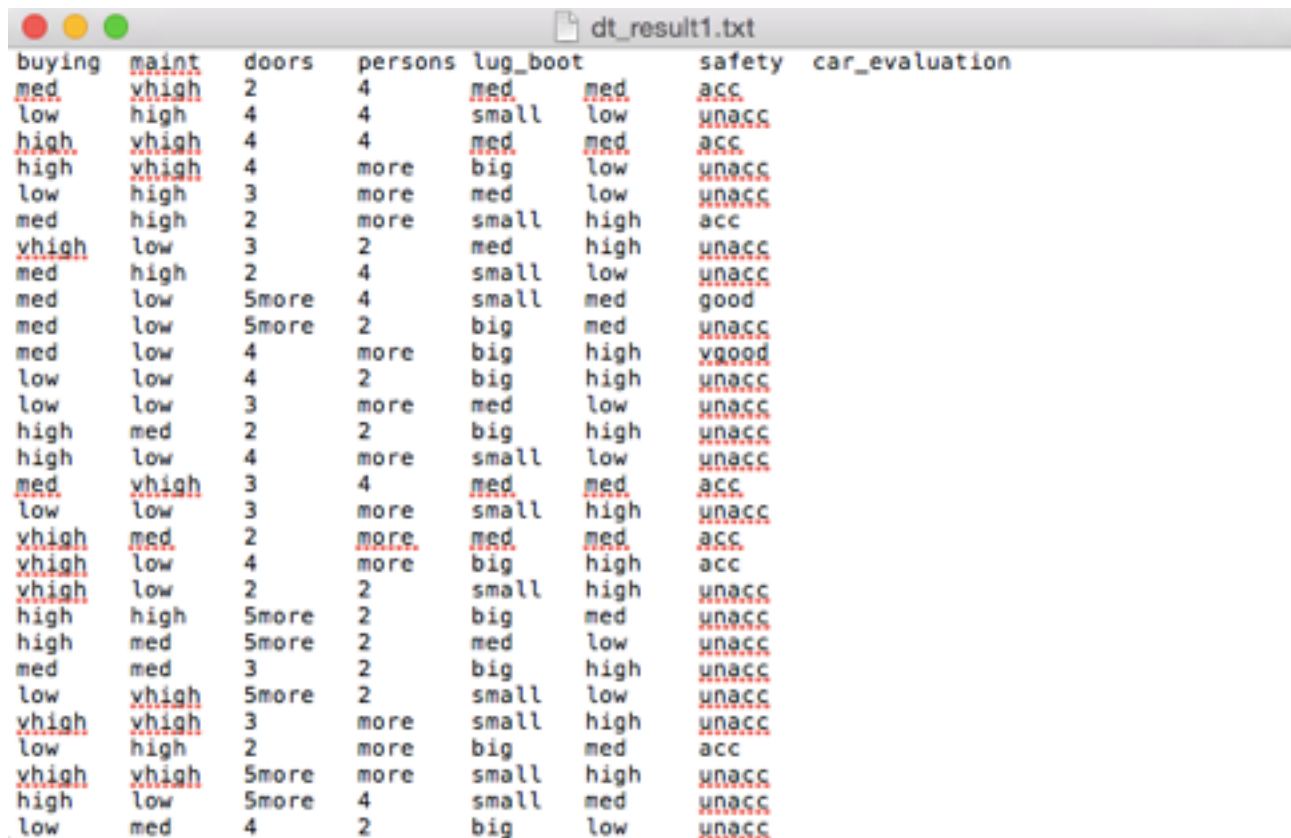
3. Instruction for compiling my source code

you need to make some changes to the source code before you run.

If you scroll down, You need to modify the absolute addresses of 'f_train_name', 'f_test_name' and 'f_result_name'. The line number is 168,169 and 170.

My code was written in Python so when you run it you need to input like this "dt.py dt_train.txt dt_test.txt dt_result.txt". Enter python program name first, followed by the train set filename, test set filename, result set filename.

4. test result



buying	<u>maint</u>	doors	persons	lug_boot	safety	car_evaluation
<u>med</u>	<u>vhigh</u>	2	4	<u>med</u>	<u>med</u>	<u>acc</u>
low	high	4	4	small	low	<u>unacc</u>
<u>high</u>	<u>vhigh</u>	4	4	<u>med</u>	<u>med</u>	<u>acc</u>
<u>high</u>	<u>vhigh</u>	4	more	big	low	<u>unacc</u>
low	high	3	more	med	low	<u>unacc</u>
med	high	2	more	small	high	<u>acc</u>
<u>vhigh</u>	low	3	2	med	high	<u>unacc</u>
med	high	2	4	small	low	<u>unacc</u>
med	low	5more	4	small	med	good
med	low	5more	2	big	med	<u>unacc</u>
med	low	4	more	big	high	<u>vgood</u>
low	low	4	2	big	high	<u>unacc</u>
low	low	3	more	med	low	<u>unacc</u>
high	med	2	2	big	high	<u>unacc</u>
high	low	4	more	small	low	<u>unacc</u>
<u>med</u>	<u>vhigh</u>	3	4	<u>med</u>	<u>med</u>	<u>acc</u>
low	low	3	more	small	high	<u>unacc</u>
<u>vhigh</u>	<u>med</u>	2	<u>more</u>	<u>med</u>	<u>med</u>	<u>acc</u>
<u>vhigh</u>	low	4	more	big	high	<u>acc</u>
<u>vhigh</u>	low	2	2	small	high	<u>unacc</u>
high	high	5more	2	big	med	<u>unacc</u>
high	med	5more	2	med	low	<u>unacc</u>
med	med	3	2	big	high	<u>unacc</u>
low	<u>vhigh</u>	5more	2	small	low	<u>unacc</u>
<u>vhigh</u>	<u>vhigh</u>	3	more	small	high	<u>unacc</u>
low	high	2	more	big	med	<u>acc</u>
<u>vhigh</u>	<u>vhigh</u>	5more	more	small	high	<u>unacc</u>
high	low	5more	4	small	med	<u>unacc</u>
low	med	4	2	big	low	<u>unacc</u>

I have created a program that compares my results file with the correct answer file and found that I got 91% correct answer rate.

```
318 / 346
91.9075144509
```