Data Science – Report #2

I. Development Environment

i. OS: Mac

ii. Programming Language: Python 3.8.2

iii. Captured Picture of Compiling

```
Programming_Assignment2 eunjin$ python dt.py dt_train.txt dt_test.txt dt_result.txt
Programming_Assignment2 eunjin$ python dt.py dt_train1.txt dt_test1.txt dt_result1.txt
```

II. Overview

i. This algorithm creates a tree recursively. First of all, all training samples are at root, and the samples are partitioned as the tree branches on a selected attribute which classifies the samples the best.

ii. It stops with three conditions

  i. All samples for a given node belongs to the same class

  ii. There are no remaining attributes for further partitioning → Use majority voting

  iii. There are no samples left → Use majority voting

iii. Attribute selection measure: information gain

III. Detailed Explanation of Code

```
def openfile(file):
    f = open(file, 'r')
    attributes_name = f.readline().strip().split('\t')
    samples = []; class_label = set(); attributes = defaultdict(set)
    while True:
        dic = {}
        line = f.readline().strip()
        if not line: break
        for idx, val in enumerate(line.split('\t')):
            dic[attributes_name[idx]] = val
            attributes[attributes_name[idx]].add(val)
        class_label.add(dic[attributes_name[idx]])
        samples.append(dic)
    f.close()
    return samples, attributes, class_label
```

i. This function is to open a given training files and save the data in variable 'samples'. I used dictionary to process the given data easily.

```
# Count class-label of train samples
def getCount(samples, class_label):
    class_count = dict.fromkeys(sorted(set(class_label)), 0) # To count the number of classes
    label = list(samples[0].keys())[-1] # To get class label name of attributes
    for sample in samples:
        class_count[sample[label]] += 1
    return class_count

# Majority voting: Return the most heterogeneous class
def getMajority(samples, attributes):
    # Sort counts of class label as a descending order
    class_cnt = Counter(getCount(samples, class_label))
    return class_cnt.most_common()[0][0]
```

ii. These functions are to get a result of majority voting. When the 'getMajority' function is called, it calls 'getCount' function which returns a class's name and its quantity. Then, it uses Counter module to get it as a descending order and to get the most common class label.

```
def getEntropy(count, total):
    info = 0
    for cnt in count.values():
        p = (int(cnt))/total
        info -= p*math.log(p,2)
    return info

# Return the best attribute's name
def selectBestAttribute(samples, attributes):
    total = len(samples) # Number of data
    class_name = list(attributes.keys())[-1] # Get attribute's name of class

    # Get entropy of each attributes ==>info_A(D)
    # Find the minimum entropy which is the best attribute
    min_info = 1000
    best_attr = ""
    for attr_name in list(attributes.keys())[:-1]:
        info_A = 0
        for attr_val in attributes[attr_name]:
            new_samples = [ s for s in samples if s[attr_name] == attr_val ]
            class_cnt = Counter( [s[class_name] for s in new_samples] )
            s_total = sum(class_cnt.values())
            info_A += (s_total/total)*(getEntropy(class_cnt, s_total))

        if min_info >= info_A:
            min_info = info_A
            best_attr = attr_name

    return best_attr
```

iii. 'getEntropy' function is to get entropy to classify a tuple in D.

iv. 'selectBestAttribute' function is to select an attribute which classifies the data the best. It computes entropy after using an attribute A to split D into v partitions as explained in a lecture note. Then, it finds the smallest entropy.

```
# Return True if all samples belong to the same class
def checkClass(samples, attributes):
    class_name = list(attributes.keys())[-1] # Get attribute's name of class
    class_label = set([s[class_name] for s in samples]) # Use a set to figure out the number of existing class

    if len(class_label) == 1:
        return True
    else:
        return False
```

v. This function is to check whether given samples are in a same class or not. If they belong to one class, it returns True. Otherwise, it returns False.

```
# Create a tree recursively
def createTree(samples, attributes, class_label):

    # Stop if all samples belong to the same class
    if checkClass(samples, attributes):
        class_name = list(attributes.keys())[-1] # Get attribute's name of class
        class_label = [s[class_name] for s in samples] # Use a set to figure out the name of a class label
        return class_label[0]

    # Stop if no remaining attributes for partitioning
    if len(attributes) == 1:
        return getMajority(samples, class_label)

    # Select the best attribute and Create a tree
    # The Attribute that best classifies samples
    # This is new root of subtree
    best_attr_name = selectBestAttribute(samples, attributes)
    best_attr_val = attributes[best_attr_name]

    subtree = {}
    subtree[best_attr_name] = {}
    for attr_val in best_attr_val:
        new_samples = [ s for s in samples if s[best_attr_name] == attr_val ]
        new_attributes = copy.deepcopy(attributes)
        del new_attributes[best_attr_name]

        # If there is no more samples, return the majority voting. Otherwise, create a subtree recursively
        if len(new_samples) == 0:
            subtree[best_attr_name][attr_val] = getMajority(samples, class_label)
        else:
            subtree[best_attr_name][attr_val] = createTree(new_samples, new_attributes, class_label)

    return subtree
```

vi. This function is to create a tree recursively. Firstly, it checks whether given samples are in a same class or not. If does, it returns the class. Secondly, it checks whether there are remaining attributes for partitioning or not. If does, it returns the result of majority voting. Then, it finds best attributes and create a subtree. To make the subtree, it processes the samples as it branches on the best attribute. Then, it checks whether there are remaining samples or not. If does, it returns the result of majority voting. Otherwise, it creates a subtree recursively.

```
def openTestfile(file):
    f = open(file, 'r')
    line = f.readline().strip()
    result = line + "\tClass Label\n" # Save a output
    attributes = line.split('\t')
    samples = []
    while True:
        dic = {}
        line = f.readline().strip()
        if not line: break
        for idx, val in enumerate(line.split('\t')):
            dic[attributes[idx]] = val
        samples.append(dic)
    f.close()
    return samples, result
```

vii. This function is to save the given test file.

```
def classify(dt, sample, attributes):
    node = list(dt.keys())[0]
    next_node = sample[node]

    # If the node is leaf, return the value otherwise find the leaf recursively
    if isinstance(dt[node][next_node], str):
        return dt[node][next_node]
    else:
        return classify(dt[node][next_node], sample, attributes)
```

viii. This function is to find a class of a row of given test samples. It finds a result recursively and return the result.

```
def writefile(file, result):
    f = open(file, 'w')
    f.write(result)
    f.close()
    return
```

ix. This function to save the result in a given txt file.

```
if __name__ == "__main__":
    train_samples, attributes, class_label = openfile(sys.argv[1])
    decision_tree = createTree(train_samples, attributes, class_label)

    test_samples, result = openTestfile(sys.argv[2])
    for sample in test_samples:
        result += '\t'.join(list(sample.values())) + '\t'
        result += classify(decision_tree, sample, list(attributes.keys())[:-1]) + '\n'

    writefile(sys.argv[3], result)
```

x. This is main function. First of all, it saves the training samples. Secondly, it makes a decision tree. Then, it saves the test samples and writes the result

IV. Testing Result

i.

```
C:₩Users₩PC₩Desktop₩test>dt_test.exe dt_answer.txt dt_result.txt
5 / 5

C:₩Users₩PC₩Desktop₩test>dt_test.exe dt_answer1.txt dt_result1.txt
323 / 346
```