2014000082 구희강

# Data Science

## PROGRAMMING ASSIGNMENT #2 – DECISION TREE

### Code Description and Algorithm

**<main>**

```
if __name__ == "__main__":
    train_file = sys.argv[1]
    test_file = sys.argv[2]
    result_file = sys.argv[3]
    train_data, attr, labels, attr_string = readTrainData(train_file)
    test_data = readData(test_file)
    dTree = buildTree(train_data,attr,labels)
    classify(dTree,test_data,attr,labels)
    writeData(result_file,test_data,attr_string)
```

Make train data list(train_data), create attribute list, attribute values. With this value build decision tree(dTree) by buildTree() function. Then labels class attribute using decision tree.

**<readTrainData>**

```
def readTrainData(name):
    fin = open(name,'r')
    attr_string = fin.readline().strip('\n')
    attr = list(attr_string.split('\t'))
    attr.pop()
    data = []
    while True:
        tmp = fin.readline()
        if not tmp:
            break
```

```
        tmp = list(tmp.strip('\n').split('\t'))
        data.append(tmp)
    labels = []
    for i in range(len(attr)):
        labels.append([])
    for i in range(len(attr)):
        for tup in range(len(data)):
            if data[tup][i] not in labels[i]:
                labels[i].append(data[tup][i])
    fin.close()
    return (data, attr, labels, attr_string)
```

readTrainData() function parses input files and makes train data list, attribute list, attribute label list.

**<readData>**

```
def readData(name):
    fin = open(name,'r')
    fin.readline()
    data = []
    while True:
        tmp = fin.readline()
        if not tmp:
            break
        tmp = list(tmp.strip('\n').split('\t'))
        data.append(tmp)
    fin.close()
    return data
```

readData() function simply parse input files. This function is for test data.

**<writeData>**

```
def writeData(name,data,attr_string):
    fout = open(name,'w+')
    tmp = []
    tmp.append(attr_string)
    for tup in data:
        tmp.append('\t'.join(tup))
    fout.write('\n'.join(tmp))
    fout.close()
```

writeData() function combines given data and writes as a text file.

**<getEntropy>**

```
def getEntropy(data):
    t_num = len(data)
    target = len(data[0]) -1
    tmp = {}
    entropy = 0
    for i in range(t_num):
        label = data[i][target]
        if label not in tmp:
            tmp[label] = 1
        else:
            tmp[label] = tmp[label] + 1
    for key in tmp.keys():
        p = tmp[key]/t_num
        entropy += -1*p*math.log2(p)
    return entropy
```

getEntropy() function returns entropy value of given list of tuples

**<getSplitInfo>**

```
def getSplitInfo(label, num):
    ret = 0
    for key in label:
        p = label[key] / num
        if p != 0:
            ret += -1*p*math.log2(p)
    return ret
```

getSplitInfo() function also returns entropy value of given labels (each number has been counted ahead)

**<majorityVote>**

```
def majorityVote(data,target):
    d = {}
    for tup in data:
        label = tup[target]
        if label not in d:
            d[label] = 1
        else:
            d[label] += 1
    count = 0
    major = ''
    for label in d:
        if count < d[label]:
            count = d[label]
```

```
            major = label
    return major
```

majorityVote() function returns most major value in the given data by counting each value.

**<Node Class>**

```
class Node:
    def __init__(self,attr,leaf):
        self.attr = attr
        self.leaf = leaf
        if not leaf:
            self.child = {}
    def add(self,key,node):
        self.child[key] = node
    def isLeaf(self):
        return self.leaf
```

Each node has attribute name, and its child is also node. Child nodes is in saved in dictionary type with its attribute value(key). And node could be a leaf node if the leaf option is True.

**<buildTree>**

```
def buildTree(data,attr,labels):
    #tuple number
    t_num = len(data)
    #attribute number
    a_num = len(data[0]) - 1
    # class attribute index
    target = len(data[0]) - 1
    #parent info
    parent_info = getEntropy(data)
    if parent_info == 0:
        node = Node(data[0][target],True)
        return node
    # majority vote
    elif len(attr) == 0:
        value = majorityVote(data,target)
        node = Node(value,True)
        return node
```

buildTree() makes decision tree by recursive function call. parent_info is entropy value before split by certain attribute. If parent_info is zero, it means all of class

attribute in given data has same value, so it returns leaf node which is label with certain value. And if there is no more attribute left, returns leaf node which is labeled with majority voting.

```
    max = -1
    winner_index = -1
    for a_index in range(a_num):
        child_info = 0
        label = {}
        for key in labels[a_index] :
            label[key] = 0
        #attribute count
        for i in range(t_num):
            key = data[i][a_index]
            label[key] += 1
        splitinfo = getSplitInfo(label,t_num)
        for key in label.keys():
            p_key = label[key] / t_num
            new_list = []
            for i in range(t_num):
                if data[i][a_index] == key:
                    new_list.append(data[i])
            if len(new_list) > 0:
                child_info += getEntropy(new_list) * p_key
        info_gain = parent_info - child_info
        gain_ratio = info_gain/splitinfo
        if gain_ratio >= max:
            max = gain_ratio
            winner_index = a_index
    winner_attr = attr[winner_index]
    winner_labels = labels[winner_index]
    node = Node(winner_attr,False)
    for key in winner_labels:
        new_data = []
        new_attr = copy.deepcopy(attr)
        new_labels = copy.deepcopy(labels)
        del new_attr[winner_index]
        del new_labels[winner_index]
        for tup in data:
            if tup[winner_index] == key:
                new_tup = copy.deepcopy(tup)
                new_tup.pop(winner_index)
                new_data.append(new_tup)
        if len(new_data) == 0:
```

```
            label = majorityVote(data,target)
            child = Node(label,True)
            node.add(key,child)
        else:
            child = buildTree(new_data,new_attr,new_labels)
            node.add(key,child)
    return node
```

child_info is entropy value after split by certain attribute. splitinfo is also entropy value. Splitting decision tree by attribute that has large gain_ratio (info_gain/splitinfo) among remaining attributes, it means the tree tends to split unbalanced and subsets contain instances with similar values. Tree splits the decision tree by winner attribute.

**<classify>**

```
def classify(dTree,data,attr,labels):
    for tup in data:
        node = dTree
        while True:
            if node.leaf:
                tup.append(node.attr)
                break
            current_attr = node.attr
            attr_index = attr.index(current_attr)
            key = tup[attr_index]
            try :
                node = node.child[key]
            except:
                tup.append("error")
                break
```

In classify() function, by traveling across the decision tree, attribute class is labeled.

## Test Result

Usage :

**>> python priori.py [train file name] [test file name] [result file name]**

<example>

```
C:\Users\GUR\Desktop\goni
λ python dt.py dt_train1.txt dt_test1.txt dt_result1.txt
```

Test environment

- Windows 10

Test requirement

- Python 3

Python program estimated class attribute label by decision tree.

<result>

```
C:\Users\GUR\Desktop\goni
λ dt_test.exe dt_result1.txt dt_answer1.txt
318 / 346
```