

---

# Data Science Assignment#2

Classify the test set using the decision tree algorithm + Python3

Lee Eunah - 2018년 4월 18일

---

Course name: Data Science (ITE 4005)

Professor: Sang-Wook Kim

TAs: Jangwan Koo

Tae-ri Kim

Student name: Lee Eunah(이 은아)

Student number: 2016025769

Major: Computer Science Engineering

---

## Implementation environment

- OS: Mac OS 10.12.6
- Language: Python 3.6.3

## Summary of my algorithm

Creates a node that has attribute of decision tree and then uses recursion function that inputs training data sets to create decision tree. When the tree is generated, it is important to find out which attribute has priority. So entropy, is called information gain, has to be calculated. The entropy has a value between 0 and 1. It means that data is classified homogeneous when the value is 0. And the larger the entropy, the better attribute for classification. Likewise, when searching the tree from test data sets, we can recursively find out class label.

## Detailed description of my codes

```
import sys    # import for command line
from math import log
```

- Importing sys module for using command line arguments.

```
class Node:
    # create a node constructor
    def __init__(self, index=None, value=None, leaf=None, left=None, right=None)
        self.index = index
        self.value = value
        self.leaf = leaf
        self.left = left
        self.right = right
```

- Create class Node and its constructor.
- index = index of attribute in attribute set
- value = value of attribute
- leaf = stored class label in leaf node
- left, right = pointing next node

---

```
def entropy(datas):
    dataSize = len(datas)
    classLabel = {}
    entropy = 0.0

    for data in datas:
        # only check class label
        currentLabel = data[-1]
        if currentLabel not in classLabel.keys():
            classLabel[currentLabel] = 0
        classLabel[currentLabel] += 1

    # calculate entropy
    for label in classLabel:
        probability = float(classLabel[label]/dataSize)
        entropy -= probability * log(probability, 2)
    return entropy
```

- This function is for calculating information gain(entropy) in given data sets. The entropy is only considered class label of the data sets.

```
def pickAttribute(datas, index, value):
    pickData = []
    notPickData = []

    for data in datas:
        # pick datas that corresponds with value
        if data[index] == value:
            pickData.append(data[:])
        # the rest of datas
        else:
            notPickData.append(data[:])
    return pickData, notPickData
```

- This function is for picking out data sets that correspond with value. Also the data sets that do not correspond with value are stored another list variable.

---

```

def createTree(datas, attributes):
    weightedEntropy = 0.0
    rmAttribute = attributes[:-1]
    bestGain = 0.0

    for i in range(len(rmAttribute)):
        attributeCount = {}

        # find out which value exists
        for data in datas:
            if data[i] not in attributeCount.keys():
                attributeCount[data[i]] = 1

        for value in attributeCount:
            # pick datas that corresponds with value(left)
            # the rest of datas(right)
            # :it functions majority voting when the data set occurs exception.
            left, right = pickAttribute(datas, i, value)

            # calculate gain
            probability = float(len(left)/len(datas))
            weightedEntropy = probability * entropy(left) + (1-probability) * entropy(right)
            # gain of the value
            gain = entropy(datas) - weightedEntropy

            # compare the gains for the best gain value
            if bestGain <= gain:
                bestGain = gain
                bestIndex = i
                bestValue = value
                bestLeft = left
                bestRight = right

    if bestGain > 0:
        leftNode = createTree(bestLeft, attributes)
        rightNode = createTree(bestRight, attributes)
        return Node(bestIndex, bestValue, None, leftNode, rightNode)

    # leaf node (bestGain = 0: when all of class labels are homogeneous)
    else:
        classLabel = {}
        for data in datas:
            currentLabel = data[-1]
            if currentLabel not in classLabel.keys():
                classLabel[currentLabel] = 0
            classLabel[currentLabel] += 1
        return Node(None, None, classLabel, None, None)

```

- This function creates decision tree.
- For making the decision tree, it needs to decide the most classified attribute. So gains that come from each value are compared to find out the best gain value.
- If the best gain value is positive, it proceeds recursive. But if not, it means that node is leaf. When the entropy is 0, class labels of data sets are homogeneous. It is one of the stopping conditions of decision tree.
- There are no data sets when the bestGain is 0. That is, the data sets are not divided anymore or there is no data in given branch. They are also the stopping conditions of decision tree.

---

```
def recursiveTest(data, tree):
    # recur until reaching leaf node
    if tree.leaf == None:
        node = None
        if data[tree.index] == tree.value:
            node = tree.left
        else:
            node = tree.right
        return recursiveTest(data, node)
    else:
        # when reaches leaf node
        return tree.leaf
```

- This function is for searching decision tree when data set inputs.
- If the node is leaf, just returns its class label. But if not, it proceeds recursive.

```
def testTree(testData, tree):
    # store class label value in list
    classLabel = []
    for data in testData:
        tested = recursiveTest(data, tree)
        listTest = list(tested)
        classLabel.append(listTest[0])
    return classLabel
```

- This function is for storing the class labels

```
def readData(fileName):
    datas = []
    inputfile = open("./"+fileName, 'r')
    firstLine = inputfile.readline()
    attributes = firstLine[:-1].split('\t')

    lines = inputfile.readlines()
    for line in lines:
        divide = line[:-1].split('\t')
        datas.append(divide)

    inputfile.close()
    return datas, attributes
```

- Read data function.
- The file data is divided by value datas and attributes.

---

```
def writeData(fileName, attributes, testDatas, classLabel):
    word = ""
    file = open("./" + fileName, "w")
    for i in range (len(attributes)):
        word += attributes[i]
        word += '\t'
        if i == len(attributes) -1:
            word += '\n'

    for j in range (len(testDatas)):
        for i in range (len(testDatas[j])):
            word += testDatas[j][i]
            word += '\t'
            if i == len(testDatas[j]) -1:
                word += classLabel[j]
                word += '\n'

    file.write(word)
    file.close()
```

- Write data function.
- Write datas in accordance with given form.
- Main function

```
## main








# command line exception
if len(sys.argv) != 4:
    print('Please fill in the command form.
Executable_file minimum_support inputfile outputfile')
    exit()

trainFile = sys.argv[1]
testFile = sys.argv[2]
outputFile = sys.argv[3]

datas, attributes = readData(trainFile)
tree = createTree(datas, attributes)
testDatas, testAttributes = readData(testFile)
classLabel = testTree(testDatas, tree)
writeData(outputFile, attributes, testDatas, classLabel)
```

- Check command line exception

## Compiling my code


 dt\_answer.txt  
 dt\_answer1.txt  
 dt\_test.txt      -put all of files in same directory.  
 dt\_test1.txt  
 dt\_train.txt  
 dt\_train1.txt  
 dt.py

```
Decision_tree — -bash — 80x24
[Eunahui-MacBook-Pro:Decision_tree leeeunah$ ls
dt.py          dt_answer1.txt  dt_test1.txt    dt_train1.txt
dt_answer.txt  dt_test.txt     dt_train.txt
Eunahui-MacBook-Pro:Decision_tree leeeunah$
```

- move to that directory.

```
Eunahui-MacBook-Pro:Decision_tree leeeunah$ python dt.py dt_train.txt dt_test.txt dt_answer.txt
```

- When I enter that command, the result .txt is...

 dt\_answer.txt

age	income	student	credit_rating	Class:buys_computer
<=30	low	no	fair	no
<=30	medium	yes	fair	yes
31...40	low	no	fair	yes
>40	high	no	fair	yes
>40	low	yes	excellent	no

```
Eunahui-MacBook-Pro:Decision_tree leeeunah$ python dt.py dt_train1.txt dt_test1.txt dt_answer1.txt
```

- When I enter that command, the result .txt is...

dt_answer1.txt							
buying	maint	doors	persons	lug_boot	safety	car_evaluation	
med	vhigh	2	4	med	med	unacc	
low	high	4	4	small	low	unacc	
high	vhigh	4	4	med	med	unacc	
high	vhigh	4	more	big	low	unacc	
low	high	3	more	med	low	unacc	
med	high	2	more	small	high	unacc	
vhigh	low	3	2	med	high	unacc	
med	high	2	4	small	low	unacc	
med	low	5more	4	small	med	acc	
med	low	5more	2	big	med	unacc	
med	low	4	more	big	high	vgood	
low	low	4	2	big	high	unacc	
low	low	3	more	med	low	unacc	
high	med	2	2	big	high	unacc	
high	low	4	more	small	low	unacc	
med	vhigh	3	4	med	med	unacc	
low	low	3	more	small	high	good	
vhigh	med	2	more	med	med	unacc	
vhigh	low	4	more	big	high	acc	
vhigh	low	2	2	small	high	unacc	
high	high	5more	2	big	med	unacc	
high	med	5more	2	med	low	unacc	
med	med	3	2	big	high	unacc	
low	vhigh	5more	2	small	low	unacc	
vhigh	vhigh	3	more	small	high	unacc	
low	high	2	more	big	med	acc	
vhigh	vhigh	5more	more	small	high	unacc	
high	low	5more	4	small	med	unacc	
low	med	4	2	big	low	unacc	

## Any other specification about my code

- It is difficult to catch exceptions that decide the tree is stopped.
- It is possible that all of test data sets do not exist. So the result is not exact.
- At first, I tried to implement decision tree by using dictionary type. But it occurred lots of errors. So I changed my mind to use node of class.
- My code satisfy stopping conditions of decision tree:
  1. There is no attribute to divide.
  2. All class label of data sets is homogenous
  3. There are no tuples for a given branch.
- It does not consider gain ratio for attribute selection. The entropy is calculated on the basis of value. So the number of branch is one. The gain ratio for attribute selection is applied two or more branches.