

3.103.

30.09.2020.

Harsh R. Sah
186174081

Program:

```
import numpy as np  
import math
```

```
from data_loader import read_data
```

```
class Node:
```

```
    def __init__(self, attribute):  
        self.attribute = attribute  
        self.children = []  
        self.cans = ''
```

```
    def __str__(self):  
        return self.attribute
```

```
def subtables(data, col):
```

```
    dict = {}
```

```
    item = np.unique(data[:, col])
```

```
    count = np.zeros([item.shape[0], 1], dtype=int)
```

```
    for x in range(item.shape[0]):
```

```
        for y in range(data.shape[0]):
```

```
            if data[y, col] == item[x]:
```

```
                count[x] += 1
```

```
    for x in range(item.shape[0]):
```

```
        dict[item[x]] = np.empty([int(count[x]), data.shape[0]])
```

```
    dtype = 'S32'
```

```
    pos = 0
```

```
    for y in range(data.shape[0]):
```


TITLE : _____

Expt. No. : _____

Date : _____

if data[y, col] == items[x]:

dict(items[x])['pos'] = data[y]

pos += 1

dict(items[x]) = np.delete(dict(items[x]), col, 1)

return items, dict

Harsh R
IBH17CS001

def entropy(s):

items = np.unique(s)

if items.size == 1:

return 0

count = np.zeros((items.shape[0], 1))

sums = 0

for x in range(items.shape[0]):

count[x] = sum(s == item[x]) / (s.size)

for count in counts:

sums += -1 * count * math.log(count, 2)

return sums.

gain_ratio(data, col):

item, dict = subtables(data, col)

total_size = data.shape[0]

entropies = np.zeros((items.shape[0], 1))

Marks : _____

Staff : _____

P1
11
in
f
c

```
for x in range(items.shape[0]):
    ratio = dict(items[x]).shape[0] / (total_size)
    entropies[x] = ratio * entropy(dict(items[x]))
```

```
total_entropy = entropy(data[:, -1])
```

```
for x in range(entropies.shape[0]):
```

```
    total_entropy -= entropies[x]
```

```
return total_entropy
```

```
def creat_node(data, metadata):
```

```
    if (np.unique(data[:, -1]).shape[0] == 1):
```

```
        node = Node("")
```

```
        node.answer = np.unique(data[:, -1])[0]
```

```
        return node.
```

```
    gains = np.zeros((data.shape[1] - 1, 1))
```

```
    for col in range(data.shape[1] - 1):
```

```
        gains[col] = gain_ratio(data, col)
```

```
    split = np.argmax(gains)
```

```
    node = Node(metadata[split])
```

```
    metadata = np.delete(metadata, split, '0')
```

```
    items, dict = sublabels(data, split)
```

```
    for x in range(items.shape[0]):
```

```
        child = creat_node(dict(items[x]), metadata)
```

```
        node.children.append((items[x], child))
```

```
    return node.
```


TITLE : _____

Expt. No. : _____

Date : _____

def empty (size) :

s = ""

for n in range (size)

st = " "

return s

Harsh P Jadh

13617CS031

def print-tree (node, level) :

if node.answer != ""

print (empty (level), node.answer)

return

print (empty (level), node.attribute)

for value, n in node.children :

print (empty (level+1), value)

print-tree (n, level+1)

metadata, traindata = read_data ("training examples.csv")

data = np.array (traindata)

node = create_node (data, metadata)

print-tree (node, 0)

Marks : _____

Staff : _____

Output 3.

stay

Ramy

b'no'

seemy

b'yes'

PROGRAM 3

By Harsh R Jain 1BG17CS031

```
import numpy as np
import math
from data_loader import read_data

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def __str__(self):
        return self.attribute

def subtables(data, col):
    dict = {}
    #unique values of a particular attribute
    items = np.unique(data[:, col])
    #initializes the count of an attribute value in the training data to zero
    count = np.zeros((items.shape[0], 1), dtype=int)
    #counts the no. of occurrence of an attribute value in the training data
    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]),
dtype="S32")
        pos = 0
        #create a dict containing key as the attribute value and value as the
list of instances with attribute value
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
```

```
    #from the dict created above remove the value which matches with the
dict key
```

```
    dict[items[x]] = np.delete(dict[items[x]], col, 1)
    return items, dict
```

```
def entropy(S):
```

```
    #the no. of target attribute values
```

```
    items = np.unique(S)
```

```
    #if the collection contains only 1 element the entropy value is zero
```

```
    if items.size == 1:
```

```
        return 0
```

```
    #initializes the count of instances with the target attribute
values(yes/no) to zero
```

```
    counts = np.zeros((items.shape[0], 1))
```

```
    sums = 0
```

```
    #proportion of positive and negative instances
```

```
    for x in range(items.shape[0]):
```

```
        counts[x] = sum(S == items[x]) / (S.size)
```

```
    #computing entropy
```

```
    for count in counts:
```

```
        sums += -1 * count * math.log(count, 2)
```

```
    return sums
```

```
def gain_ratio(data, col):
```

```
    #subtables function returns the possible attribute values and a
dictionary mapping a value to the instances having that value
```

```
    items, dict = subtables(data, col )
```

```
    total_size = data.shape[0]
```

```
    entropies = np.zeros((items.shape[0], 1))
```

```
    #compute info gain of each attribute
```

```
    for x in range(items.shape[0]):
```

```
        #ratio=count of instances having a attribute value/total no. of
instances
```

```
        ratio = dict[items[x]].shape[0]/(total_size)
```

```
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
```

```
    total_entropy = entropy(data[:, -1])
```

```

    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy

def create_node(data, metadata):

    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

    #gain of each attribute initialized as zero
    gains = np.zeros((data.shape[1] - 1, 1))

    #compute gain of each attribute
    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    #index of attribute having maximum gain
    split = np.argmax(gains)
    #attribute having max gain forms a node of the tree
    node = Node(metadata[split])
    #remove the attribute from metadata after making it a node in the tree
    metadata = np.delete(metadata, split, 0)
    #items-possible values of the attribute with max gain, dict- mapping of
    each value to instances having that value
    items, dict = subtables(data, split)

    #for each attribute value find the next best attribute
    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

    return node

def empty(size):
    s = ""
    for x in range(size):
        s += "    "

```



```

        return s

def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return

    print(empty(level), node.attribute)

    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)

metadata, traindata = read_data("trainingexamples.csv")

data = np.array(traindata)

node = create_node(data, metadata)

print_tree(node, 0)

```

The screenshot shows a Python IDE with two windows. The left window, titled 'ID3.py - D:\7th Sem\ML Lab\Program 3\3-ID3\ID3.py (3.8.2)', contains the following code:

```

import numpy as np
import math
from data_loader import read_data

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def __str__(self):
        return self.attribute

def subtables(data, col):
    dict = {}
    #unique values of a particular attribute
    items = np.unique(data[:, col])
    #initializes the count of an attribute value in the training data to zero
    count = np.zeros((items.shape[0], 1), dtype=int)
    #counts the no. of occurrence of an attribute value in the training data
    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="S32")
        pos = 0
        #create a dict containing key as the attribute value and value as the 11
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
        #from the dict created above remove the value which matches with the dic
        dict[items[x]] = np.delete(dict[items[x]], col, 1)
    return items, dict

def entropy(S):
    #the no. of target attribute values
    items = np.unique(S)

```

The right window, titled 'Python 3.8.2 Shell', shows the execution output:

```

Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\7th Sem\ML Lab\Program 3\3-ID3\ID3.py =====
Sky
Rainy
b'No'
Sunny
b'Yes'
>>>

```

The taskbar at the bottom shows the Windows search bar and various application icons. The system clock indicates 12:35 on 30-Sep-20.