Aakriti Singh 20190802006

Fundamentals of AI & ML

Monsoon Semester V 2021-22

Lab - 4

Date: 23rd October 2021

Topic: Decision Trees

AIM

To write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. How to find the Entropy and Information Gain in Decision Tree Learning

THERORY

Decision Trees

The decision tree algorithm is a supervised learning algorithm for classification or regression problems. Our end goal is to use historical data to predict an outcome. Unlike linear regression, decision trees can pick up nonlinear interactions between variables in the data.

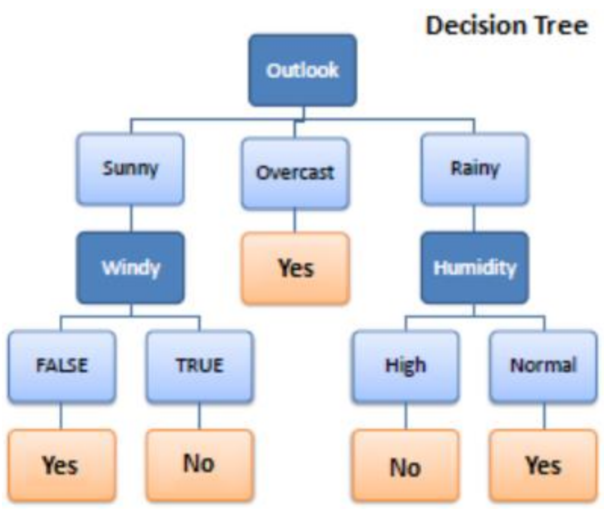


Fig.: Flowchart structure

ENTROPY

There are nine people who wants to play tennis and five people who don't want to play. If we consider how mixed the column playTennis is, we could say it was sort of mixed, with a majority (9/14) of the people wants to play tennis. Entropy gives us a way to quantify the answer" sort of mixed". The more mixed the yes and No in the column are, the higher the entropy. If playTennis had equal amounts of yes and No our entropy would be 1. If playTennis consisted only of yes the entropy would be 0.

We can use the following formula to calculate entropy:

$$-\sum_{c}^{i=1} p(x_i)\log_b p(x_i)$$

<u>Steps to calculate entropy:</u>

Let's go through each step of the formula and calculate the entropy for the Playtennis column.

1. We need to iterate through each unique value in the column and assign it to i. For example we have 2 cases in the playtennis column either yes or No.

2. We then compute the probability of that value occurring in the data. For "Yes" the probability is **9/14**. For case "No" the probability is **4/14**.

3. We take the probability of each case and multiply it by the logarithm base 2 of the probability. 2 is the most common base because entropy is measured in bits For the case of (1), we get **9/14*log2(9/14)**. For the case of (0), we get **4/14*log2(4/14)**.

4. Next we take our product from each test case above and sum it together. Which is **9/14*log2(9/14) + 4/14*log2(4/14)**.

5. Finally we negate the total sum from above , **- (9/14*log2(9/14) + 4/14*log2(4/14))**

6. Once we put the steps all together, we get –

0.9261

INFORMATION GAIN

For Information gain we split our dataset on Humidity as we wanted splits that lower out target column. When we split on Humidity we see that the playtennis entropy goes down to 0.65 on the "normal" side. Information gain will be calculated by the formula,

$$IG(T,A) = Entropy(T) - \sum_{v \epsilon A} \frac{|T_v|}{T} \cdot Entropy(T_v)$$

T = Target column or PlayTennis column.
A = the variable column we are testing, “**Humidity**”.
V = Each value in A, each value in “**Humidity**” column.

Steps to calculate information gain:

1. We’ll first calculate the original entropy for (T) before the split, **0.9261**.

2. Then for each unique value in variable A we compute the number of rows A takes on value v, and divide it by total rows. We get **7/14** for the unique values “**strong**” and **7/14** for unique values “**normal**”.

3. We add all these products together, **7/14* 0.98522 + 7/14 * 0.650022 = 0.817621**

4. When we subtract from overall entropy to get information gain, **0.817621-0.9261 = 0.108479**.

.

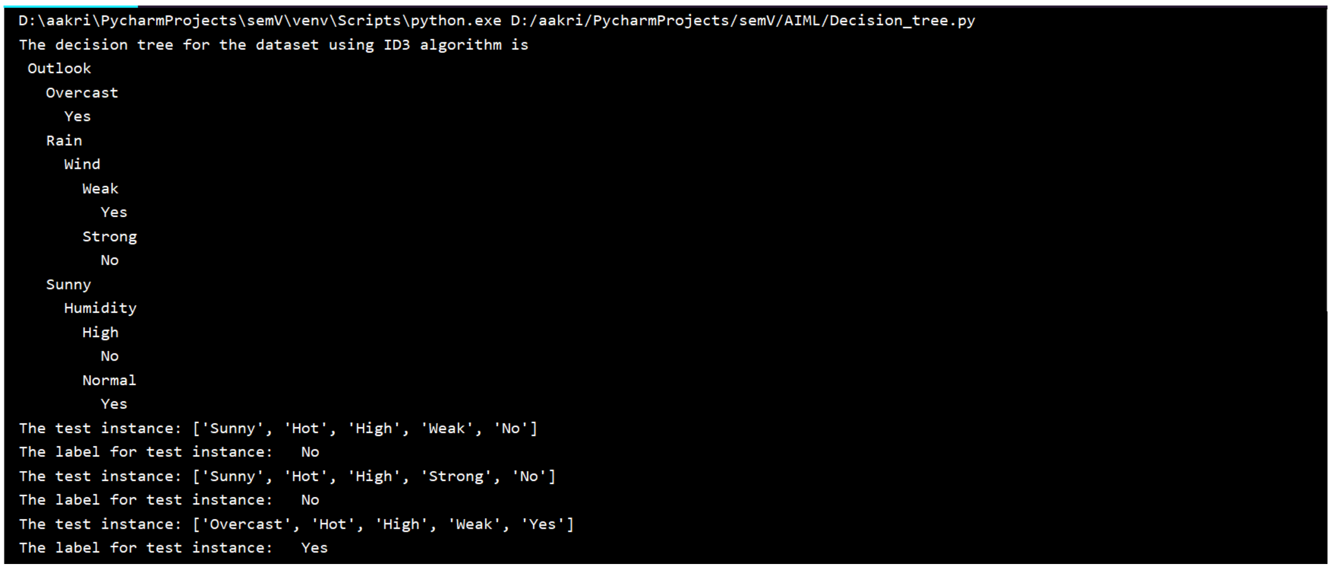
5. We end up with **0.108479** which means that we gain **0.108479** bits of information by splitting our data on “**Humidity**” column. Our information gain is low but its still positive which is because we lowered the entropy on the right side of the split. This process is repeated for every column to calculate the information gain.

PROGRAM CODE

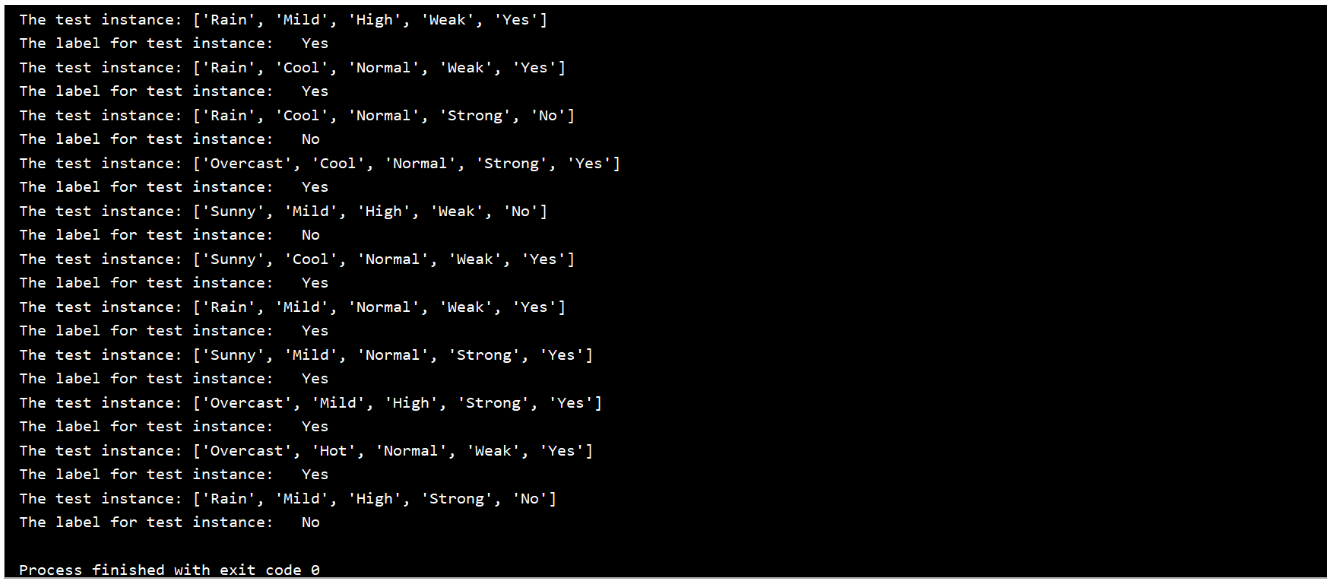
```
import math
import csv


def load_csv(filename):
    lines = csv.reader(open(filename, "r"))
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset, headers


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


def subtables(data, col, delete):
    dic = {}
    coldata = [row[col] for row in data]
    attr = list(set(coldata))

    counts = [0] * len(attr)
    r = len(data)
    c = len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col] == attr[x]:
                counts[x] += 1

    for x in range(len(attr)):
        dic[attr[x]] = [[0 for i in range(c)] for j in range(counts[x])]
        pos = 0
        for y in range(r):
            if data[y][col] == attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos] = data[y]
                pos += 1
    return attr, dic


def entropy(S):
    attr = list(set(S))
    if len(attr) == 1:
        return 0

    counts = [0, 0]
    for i in range(2):
        counts[i] = sum([1 for x in S if attr[i] == x]) / (len(S) * 1.0)

    sums = 0
    for cnt in counts:
        sums += -1 * cnt * math.log(cnt, 2)
    return sums


def compute_gain(data, col):
    attr, dic = subtables(data, col, delete=False)

    total_size = len(data)
    entropies = [0] * len(attr)
    ratio = [0] * len(attr)

    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x] = len(dic[attr[x]]) / (total_size * 1.0)
        entropies[x] = entropy([row[-1] for row in dic[attr[x]]])
        total_entropy -= ratio[x] * entropies[x]
    return total_entropy
```

```
def build_tree(data, features):
    lastcol = [row[-1] for row in data]
    if (len(set(lastcol))) == 1:
        node = Node("")
        node.answer = lastcol[0]
        return node

    n = len(data[0]) - 1
    gains = [0] * n
    for col in range(n):
        gains[col] = compute_gain(data, col)
    split = gains.index(max(gains))
    node = Node(features[split])
    fea = features[:split] + features[split + 1:]

    attr, dic = subtables(data, split, delete=True)

    for x in range(len(attr)):
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))
    return node


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

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


def classify(node, x_test, features):
    if node.answer != "":
        print(node.answer)
        return
    pos = features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos] == value:
            classify(n, x_test, features)


'''Main program'''
dataset, features = load_csv("PlayTennis.csv")
node1 = build_tree(dataset, features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1, 0)
testdata, features = load_csv("PlayTennis.csv")

for xtest in testdata:
    print("The test instance:", xtest)
    print("The label for test instance:", end="   ")
    classify(node1, xtest, features)
```

OUTPUT



```
D:\aakri\PycharmProjects\semV\venv\Scripts\python.exe D:/aakri/PycharmProjects/semV/AIML/Decision_tree.py
The decision tree for the dataset using ID3 algorithm is
 Outlook
   Overcast
     Yes
   Rain
     Wind
       Weak
         Yes
       Strong
         No
   Sunny
     Humidity
       High
         No
       Normal
         Yes
The test instance: ['Sunny', 'Hot', 'High', 'Weak', 'No']
The label for test instance:   No
The test instance: ['Sunny', 'Hot', 'High', 'Strong', 'No']
The label for test instance:   No
The test instance: ['Overcast', 'Hot', 'High', 'Weak', 'Yes']
The label for test instance:   Yes
```



```
The test instance: ['Rain', 'Mild', 'High', 'Weak', 'Yes']
The label for test instance:   Yes
The test instance: ['Rain', 'Cool', 'Normal', 'Weak', 'Yes']
The label for test instance:   Yes
The test instance: ['Rain', 'Cool', 'Normal', 'Strong', 'No']
The label for test instance:   No
The test instance: ['Overcast', 'Cool', 'Normal', 'Strong', 'Yes']
The label for test instance:   Yes
The test instance: ['Sunny', 'Mild', 'High', 'Weak', 'No']
The label for test instance:   No
The test instance: ['Sunny', 'Cool', 'Normal', 'Weak', 'Yes']
The label for test instance:   Yes
The test instance: ['Rain', 'Mild', 'Normal', 'Weak', 'Yes']
The label for test instance:   Yes
The test instance: ['Sunny', 'Mild', 'Normal', 'Strong', 'Yes']
The label for test instance:   Yes
The test instance: ['Overcast', 'Mild', 'High', 'Strong', 'Yes']
The label for test instance:   Yes
The test instance: ['Overcast', 'Hot', 'Normal', 'Weak', 'Yes']
The label for test instance:   Yes
The test instance: ['Rain', 'Mild', 'High', 'Strong', 'No']
The label for test instance:   No

Process finished with exit code 0
```

CONCLUSION

Decision trees can be a useful machine learning algorithm to pick up nonlinear interactions between variables in the data. In this lab, we analysed a decision tree classification ID3 algorithm. We then looked at theory concepts of entropy and information gain. By using these concepts we were able to build a few functions in Python to decide which variables/columns were the most efficient to split on. With a firm grasp on these concepts, we can move forward to build a decision tree.