Darshan UNIVERSITY

Data Mining

Lab - 10

Name: Harmik Rathod

Enrollment No: 24010101680

Implement Decision Tree(ID3) in python

Uses Information Gain to choose the best feature to split.

Recursively builds the tree until stopping conditions are met.

1. Calculate Entropy for the dataset.
2. Calculate Information Gain for each feature.
3. Choose the feature with maximum Information Gain.
4. Split dataset into subsets for that feature.
5. Repeat recursively until:

All samples in a node have the same label.
No features are left.
No data is left.

Step 2. Import the dataset from this address.

import Pandas, Numpy

In [1]:
```
import pandas as pd
import numpy as np
```

Create Following Data

In [37]:
```
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', '
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'C
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'Hig
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes'
})
```

In [3]:
```
data
```

Out[3]:

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

Now Define Function to Calculate Entropy

In [6]:
```
def entropy(y):
  values,counts = np.unique(y,return_counts=True)
  probabilities = counts / counts.sum()
  return -np.sum(probabilities * np.log2(probabilities))
```

Testing of Above Function -

y = np.array(['Yes', 'No', 'Yes', 'Yes'])

Function Call - > entropy(y))

output - 0.8112781244591328

```
In [14]: y = np.array(['Yes','No','Yes','Yes'])
         print(entropy(y))
```

```
0.8112781244591328
```

Define function to Calculate Information Gain

```
In [29]: def information_gain(data, split_attribute, target):
             # calculate total entropy for target
            total_entropy = entropy(data[target])
            values,counts = np.unique(data[split_attribute],return_counts=True)
            weighted_entropy = 0
            for i in range(len(values)):
                subset = data[data[split_attribute] == values[i]]
                weighted_entropy += (counts[i]/counts.sum()) * entropy(subset[target])
            return total_entropy - weighted_entropy
```

Testing of Above Function-

data = pd.DataFrame({ 'Weather': ['Sunny', 'Sunny', 'Rain', 'Rain'], 'Play': ['Yes', 'No', 'Yes', 'Yes'] })

Function Call - > information_gain(data, 'Weather', 'Play')

Output - 0.31127812445913283

```
In [36]: data = pd.DataFrame({
             'Weather': ['Sunny', 'Sunny', 'Rain', 'Rain'],
             'Play':    ['Yes', 'No',    'Yes', 'Yes']
         })
         print(information_gain(data,'Weather','Play'))
```

```
0.31127812445913283
```

Implement ID3 Algo

```
In [32]: def id3(data, features, target):
             # If all labels are same → return the label
             if len(np.unique(data[target])) == 1:
                 return np.unique(data[target])[0]

             # If no features left → return majority label
             if len(features) == 0:
                 return data[target].mode()[0]
                 #mode returns most common values and return [0] in case of ties

             # Choose best feature
             gains = [information_gain(data,feature,target) for feature in features]
             #compute the information gain for every feature in features

             best_feature = features[np.argmax(gains)]
```

```python
    #finds the index of the maximum value in gains

    tree = {best_feature: {}}
    # stores best feature in the tree

    # For each value of best feature → branch
    for value in np.unique(data[best_feature]): #gets all distinct values of best f

        sub_data = data[data[best_feature] == value].drop(columns = [best_feature])
        sub_tree = id3(sub_data, [f for f in features if f!= best_feature],target)
        tree[best_feature][value] = sub_tree

    return tree
```

Use ID3

In [38]:
```python
features = list(data.columns[:-1])
target = 'PlayTennis'
tree = id3(data, features, target)
```

Print Tree

In [39]:
```python
tree
```

Out[39]:
```
{'Outlook': {'Overcast': 'Yes',
  'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
  'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Extra: Create Predict Function

In [40]:
```python
def predict(tree, sample):
    for feature,branches in tree.items():
        value = sample.get(feature)
        if value in branches:
            result = branches[value]
            if isinstance(result,dict):
                return predict(result,sample)
            else:
                return result
    return None
```

Extra: Predict for a sample

sample = {'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'High', 'Wind': 'Strong'}

Your Answer ?

In [41]:
```python
sample = {'Outlook':'Sunny','Temperature':'Cool','Humidity':'High','Wind':'Strong'}
print('Prediction:', predict(tree,sample))
```

```
Prediction: No
```

In [44]:
```python
sample = {'Outlook':'Sunny','Temperature':'Cool','Humidity':'Normal','Wind':'Weak'}
print('Prediction:', predict(tree,sample))
```

```
Prediction: Yes
```