Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Data Mining

Lab - 10

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 [5]:
```
import pandas as pd
import numpy as np
```

Create Following Data

In [13]:
```
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 [17]:
```
data
```

Out[17]:

	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 [7]:
```
def entropy(y):
    values,counts=np.unique(y,return_counts=True)
    probabilities=counts/counts.sum()
    entropy=np.sum(-probabilities*np.log2(probabilities))
    return entropy
```

Testing of Above Function -

y = np.array(['Yes', 'No', 'Yes', 'Yes'])
Function Call - > entropy(y))

output - 0.8112781244591328

In [8]:
```
y = np.array(['Yes', 'No','Yes', 'Yes'])
entropy(y)
```

Out[8]:
```
0.8112781244591328
```

Define function to Calculate Information Gain

In [9]:
```
def information_gain(data, split_attribute, 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])
    information_gain=total_entropy-weighted_entropy
    return information_gain
```

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 [10]:
```
data = pd.DataFrame({
    'Weather': ['Sunny','Sunny','Rain','Rain'],
    'Play':    ['Yes', 'No','Yes','Yes']
})

information_gain(data,'Weather','Play')
```

Out[10]:
```
0.31127812445913283
```

Implement ID3 Algo

In [11]:
```
def id3(data, features, target):
    if len(np.unique(data[target])) == 1:
        return np.unique(data[target])[0]

    if len(features) == 0:
        return data[target].mode()[0]

    gains=[information_gain(data,feature,target) for feature in features]
    best_feature=features[np.argmax(gains)]

    tree={best_feature:{}}

    for value in np.unique(data[best_feature]):
        sub_data=data[data[best_feature]==value].drop(best_feature,axis=1)
        subtree=id3(sub_data,[f for f in features if f != best_feature],target)
        tree[best_feature][value]=subtree

    return tree
```

Use ID3

In [14]:
```
features=list(data.columns[:-1])
target='PlayTennis'

tree=id3(data,features,target)
```

Print Tree

In [15]:
```
tree
```

Out[15]:
```
{'Outlook': {'Overcast': 'Yes',
  'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
  'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Extra: Create Predict Function

In [16]:
```
def predict(tree, sample):
    if not isinstance(tree, dict):
        return tree

    root = next(iter(tree))
    feature_value = sample[root]

    if feature_value in tree[root]:
        return predict(tree[root][feature_value], sample)
    else:
        return None
```

Extra: Predict for a sample

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

Your Answer ?

In [18]:
```
features = list(data.columns[:-1])
target = 'PlayTennis'
tree = id3(data, features, target)

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

print("Prediction:", predict(tree, sample))
```

```
Prediction: No
```

In [ ]:
```

```