MACHINE LEARNING - LAB - 12 - EXECUTION

Implementation of ID3 algorithm using party dataset

```
import pandas as pd
from sklearn import metrics
df = pd.read_csv('student.csv')
df
```

[11] ✓ 0.1s Python

	Deadline	IsParty	Lazy	Activity
0	Urgent	Yes	Yes	Party
1	Urgent	No	Yes	Study
2	Near	Yes	Yes	Party
3	None	Yes	No	Party
4	None	No	Yes	Pub
5	None	Yes	No	Party
6	Near	No	No	Study
7	Near	No	Yes	TV
8	Near	Yes	Yes	Party
9	Urgent	No	No	Study

```
t = df.keys()[-1]
print('Target Attribute is   → ', t)

attribute_names = list(df.keys())
attribute_names.remove(t)

print('Predicting Attributes → ', attribute_names)
```

[12] ✓ 0.3s Python

```
Target Attribute is   →  Activity
Predicting Attributes →  ['Deadline', 'IsParty', 'Lazy']
```

```
import math
def entropy(probs):
    return sum( [-prob*math.log(prob, 2) for prob in probs])

def entropy_of_list(ls,value):
    from collections import Counter
    total_instances = len(ls)
    cnt = Counter(x for x in ls)
    probs = [x / total_instances for x in cnt.values()]
    return entropy(probs)
```

[13] ✓ 0.3s Python

```
def information_gain(df, split_attribute, target_attribute,battr):
    df_split = df.groupby(split_attribute)
    glist=[]
    for gname,group in df_split:
        glist.append(gname)

    glist.reverse()
    nobs = len(df.index) * 1.0
    df_agg1=df_split.agg({target_attribute:lambda x:entropy_of_list(x, glist.pop())})
    df_agg2=df_split.agg({target_attribute :lambda x:len(x)/nobs})

    df_agg1.columns=['Entropy']
    df_agg2.columns=['Proportion']

    new_entropy = sum( df_agg1['Entropy'] * df_agg2['Proportion'])
    if battr !='S':
        old_entropy = entropy_of_list(df[target_attribute],'S-'+df.iloc[0][df.columns.get_loc(battr)])
    else:
        old_entropy = entropy_of_list(df[target_attribute],battr)
    return old_entropy - new_entropy
```

[14] ✓ 0.1s Python

```python
def id3(df, target_attribute, attribute_names, default_class=None,default_attr='S'):

    from collections import Counter
    cnt = Counter(x for x in df[target_attribute])
    if len(cnt) == 1:
        return next(iter(cnt))
    elif df.empty or (not attribute_names):
        return default_class
    else:
        default_class = max(cnt.keys())
        gainz=[]
        for attr in attribute_names:
            ig= information_gain(df, attr, target_attribute,default_attr)
            gainz.append(ig)

        index_of_max = gainz.index(max(gainz))
        best_attr = attribute_names[index_of_max]
        tree = {best_attr:{}} # Initiate the tree with best attribute as a node
        remaining_attribute_names =[i for i in attribute_names if i != best_attr]


        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,target_attribute, remaining_attribute_names,default_class,best_attr)
            tree[best_attr][attr_val] = subtree
        return tree
```

[15] ✓ 0.8s Python

```python
def entropy_dataset(a_list):
    from collections import Counter

    cnt = Counter(x for x in a_list)
    num_instances = len(a_list)*1.0    # = 14
    print("\nNumber of Instances of the Current Sub-Class is {0}".format(num_instances ))

    # x means no of YES/NO
    probs = [x / num_instances for x in cnt.values()]
    return entropy(probs)
```

[16] ✓ 0.4s Python

```python
print("Entropy calculation for input dataset:\n")
print(df['Activity'])
```

[17] ✓ 0.4s Python

```
Entropy calculation for input dataset:

0    Party
1    Study
2    Party
3    Party
4      Pub
5    Party
6    Study
7       TV
8    Party
9    Study
Name: Activity, dtype: object
```

```python
total_entropy = entropy_dataset(df['Activity'])
print("\nTotal Entropy(S) of Play Dataset→", total_entropy)
print("————————————————————————————————————————————————")
```

[18] ✓ 0.3s Python

```
Number of Instances of the Current Sub-Class is 10.0

Total Entropy(S) of Play Dataset→ 1.6854752972273346
————————————————————————————————————————————————
```

```
from pprint import pprint
tree = id3(df,t,attribute_names)
print("\nThe Resultant Decision Tree is:-\n")
pprint(tree)
```

[19] ✓ 0.1s Python

```
The Resultant Decision Tree is:-

{'IsParty': {'No': {'Deadline': {'Near': {'Lazy': {'No': 'Study', 'Yes': 'TV'}},
                                 'None': 'Pub',
                                 'Urgent': 'Study'}},
             'Yes': 'Party'}}
```

```
attribute = next(iter(tree))
print("\nBest Attribute →",attribute)
print("Tree Keys      →",tree[attribute].keys())
```

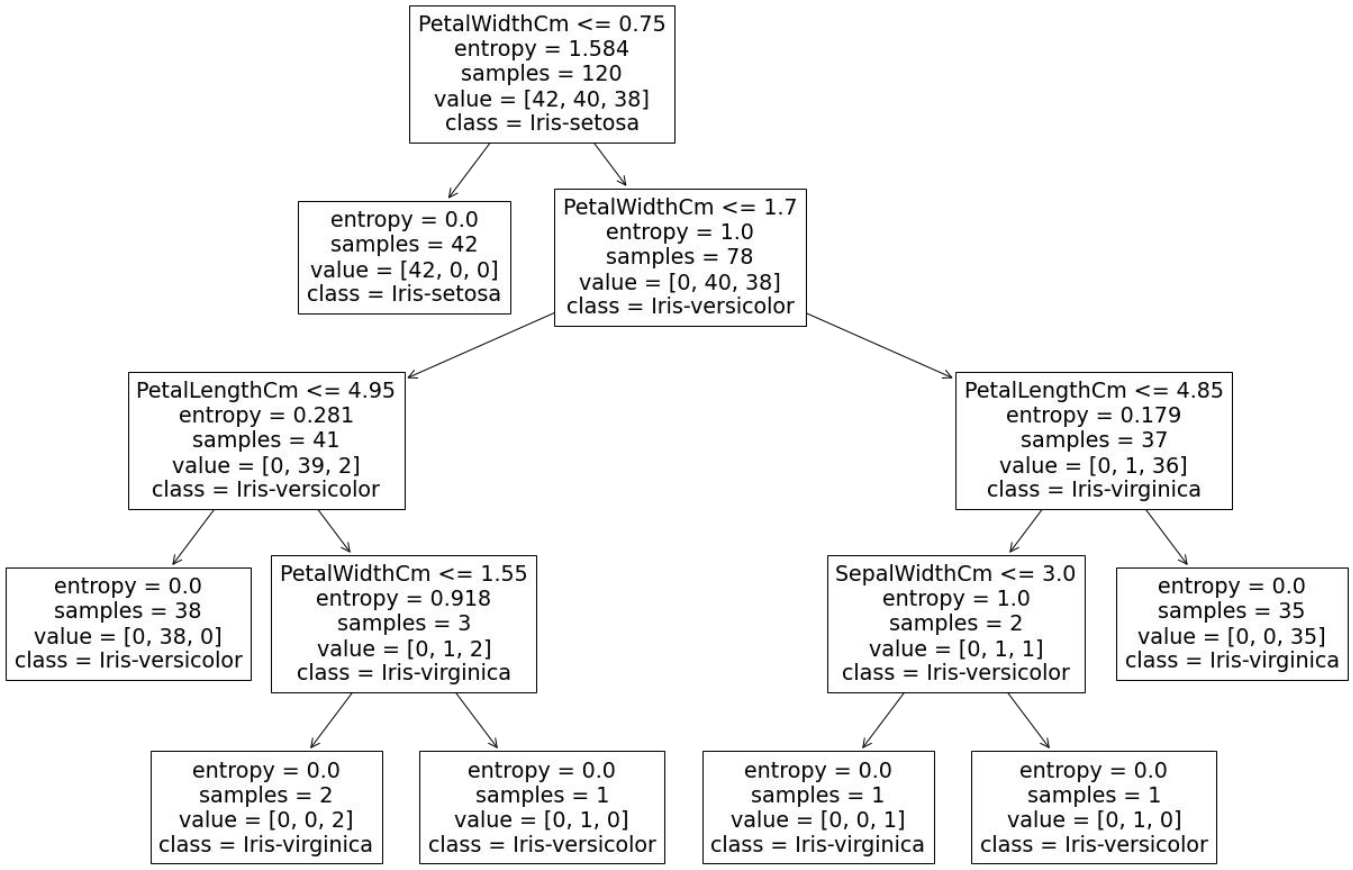
[20] ✓ 0.5s Python

```
Best Attribute → IsParty
Tree Keys      → dict_keys(['No', 'Yes'])
```

Implementation of ID3 algorithm using iris dataset

```
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

data = pd.read_csv('iris.csv')

X = data.iloc[:, 2:5]
y = data.iloc[:, -1]

datasets = train_test_split(X, y,test_size=0.2)

X_train, X_test, y_train, y_test = datasets

model = DecisionTreeClassifier(criterion="entropy")
model.fit(X_train, y_train)

plt.figure(figsize=(22,15))
tree.plot_tree(model, feature_names=X.keys(), class_names=y.unique())
plt.show()
```

[26] ✓ 1.4s Python




```
y_pred = model.predict(X_test)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

y_true = y_test

print('Accuracy = ',accuracy_score(y_pred, y_test))

print('\nConfusion Matrix: \n', confusion_matrix(y_true, y_pred))
```

[28] ✓ 0.8s Python

```
Accuracy =  0.9333333333333333

Confusion Matrix:
 [[ 8  0  0]
 [ 0  9  1]
 [ 0  1 11]]
```

```
# classification report for precision, recall f1-score and accuracy
matrix = classification_report(y_true,y_pred)
print('\nClassification report : \n',matrix)
```

[29] ✓ 0.7s Python

```
Classification report :
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.90      0.90      0.90        10
 Iris-virginica       0.92      0.92      0.92        12

       accuracy                           0.93        30
      macro avg       0.94      0.94      0.94        30
   weighted avg       0.93      0.93      0.93        30
```