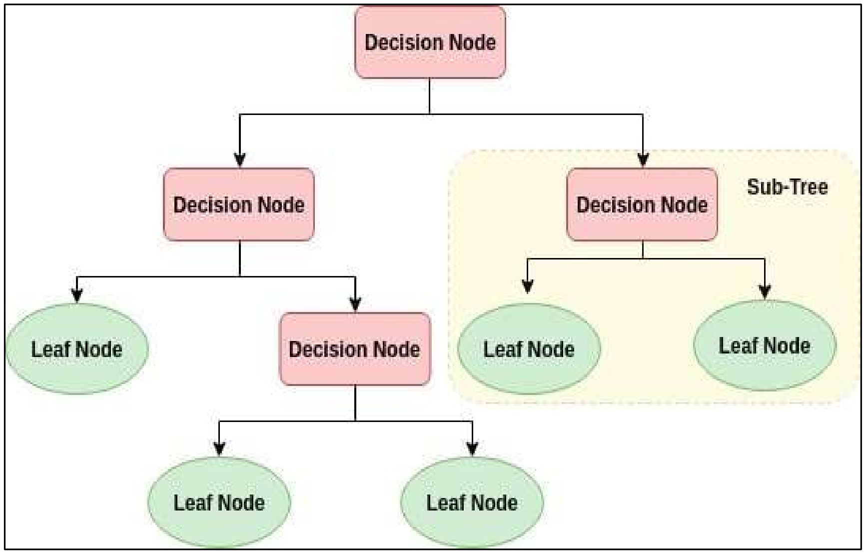
Lab Session-III(c)

(Implementing Decision Tree Classifier)

Decision Tree Classifier- Introduction

- Decision Tree Classifier is a supervised learning algorithm that is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions.
- Decision trees can handle high dimensional data with good accuracy by constructing internal decision-making logic in a form of a decision tree.
- A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome.



ID3 Algorithm

- ID3 stands for ***Iterative Dichotomiser 3*** and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.
- Invented by Ross Quinlan, ID3 uses a **top-down greedy** approach to build a decision tree.
- In simple words, the **top-down** approach means that we start building the tree from the top and the **greedy** approach means that at each iteration we select the best feature at the present moment to create a node.
- The best feature in ID3 is selected using *Entropy and Information Gain* metrics.
- Most generally ID3 is only used for classification problems with nominal features only.

Entropy and Information Gain

- Entropy of dataset (S) is computed as follows:

$$Entropy\ (S) = -\sum_{i=1}^{n} p_i log_2(p_i)$$

Where n is the total number of classes in the target column (in our case n = 2 i.e YES and NO)
p_i is the **probability of class 'i'** or the ratio of "*number of rows with class i in the target column*" to the "*total number of rows*" in the dataset.

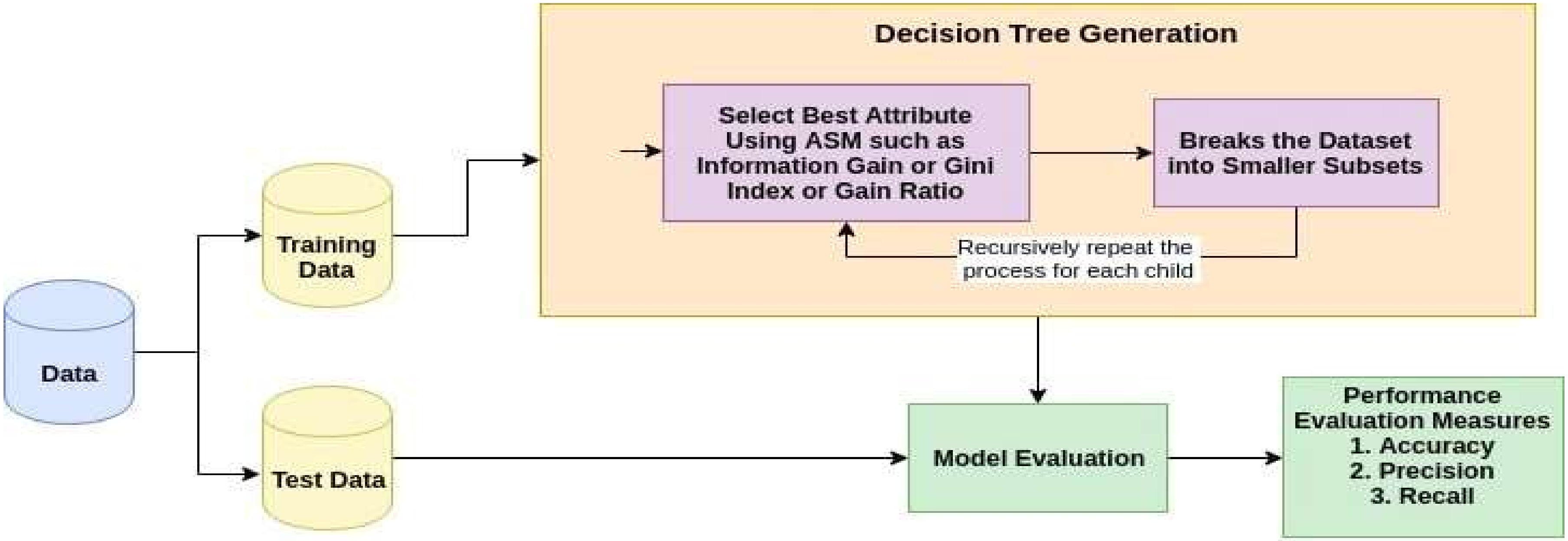
- Information Gain for a feature column **A** is calculated as:

$$Information\ Gain(S, A) = Entropy(S) - \sum_{v=1}^{|v|} \frac{|S_v|}{|S|} Entropy(S_v)$$

where **S_v** is the set of rows in **S** for which the feature column **A** has value **v**, |**S_v**| is the number of rows in **S_v** and likewise |**S**| is the number of rows in **S.**

- **Information Gain calculates the reduction in the entropy and measures how well a given feature separates or classifies the target classes. The feature with the highest Information Gain is selected as the best one.**

ID3 Algorithm



Implementing ID3- Step-by-Step

Following steps are followed to implement ID3 algorithm.

1. Load the dataset
2. Data Preprocessing: Check for Null Values, Class Balancing
3. Split the dataset into train and test
4. Define following functions for constructing decision tree:
   a. Function to compute entropy of any (set) Data Frame
   b. Function to compute entropy of any feature given a set.
   c. Function to find the best split node (on the basis of Information Gain)
   d. Function that returns the sub-table from a given set that meet the condition.
5. Construct the decision tree recursively using the functions computed in Step 4.
6. Predict the label of each new test case.
7. Perform performance evaluation by comparing the predicted and actual values of output target variable.

Steps 1-3

- For implementation of ID3 algorithm, we will be working on the **weather dataset** (discussed in class) in which we have to decide that whether the player should play golf or not on the basis of weather conditions (shown in figure).
- Code:

```
import pandas as pd

import numpy as np

# eps for making value a bit greater than 0 later on

eps = np.finfo(float).eps

from numpy import log2 as log

df=pd.read_csv('C:/Machine Learning/ML_Datasets/weather.csv’)

df1=pd.read_csv('C:/Machine Learning/ML_Datasets/weather_test.csv’)
```

Pre-processing is not required.

Training Data

Outlook	Temp	Humidity	Windy	Play
rainy	hot	high	0	0
rainy	hot	high	1	0
overcast	hot	high	0	1
sunny	mild	high	0	1
sunny	cool	normal	0	1
sunny	cool	normal	1	0
overcast	cool	normal	1	1
rainy	mild	high	0	0
rainy	cool	normal	0	1
sunny	mild	normal	0	1
rainy	mild	normal	1	1
overcast	mild	high	1	1
overcast	hot	normal	0	1
sunny	mild	high	1	0

Test Data

Outlook	Temp	Humidity	Windy	Play
rainy	cool	high	1	0
overcast	mild	normal	0	1

Step 4(a)

- Function to compute entropy of any (set) Data Frame
- It is computed as follows

$$Entropy\ (S) = -\sum_{i=1}^{n} p_i log_2(p_i)$$

Code: #Function to calculate the entropy of the label

```
def find_entropy(df):
    Class = df.keys()[-1]
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy
```

Step 4(b)

- Function to compute entropy of any feature given a set.
- Information Gain for a feature column **A** is calculated as:

$$Entropy(A) = \sum_{v=1}^{|v|} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $\mathbf{S_v}$ is the set of rows in **S** for which the feature column **A** has value **v**, $|\mathbf{S_v}|$ is the number of rows in $\mathbf{S_v}$ and likewise $|\mathbf{S}|$ is the number of rows in **S.**

Code: #Function to calculate the entropy of all features.

```
def find_entropy_attribute(df,attribute):
    Class = df.keys()[-1]
    target_variables = df[Class].unique()
    variables = df[attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
            num = len(df[attribute][df[attribute]==variable][df[Class] ==target_variable])
            den = len(df[attribute][df[attribute]==variable])
            fraction = num/(den+eps)
            entropy += -fraction*log(fraction+eps)
        fraction2 = den/len(df)
        entropy2 += -fraction2*entropy
    return abs(entropy2)
```

Step 4(c)

- Function to find the best split node (on the basis of Information Gain)

$$Information\ Gain(S, A) =$$

$$Entropy(S) - \sum_{v=1}^{|v|} \frac{|S_v|}{|S|} Entropy(S_v)$$

- **The feature with the highest Information Gain is selected as the best one.**

Code:

```
#Function to find the feature with the highest information gain.

def find_winner(df):

    IG = [ ]

    for key in df.keys()[:-1]:

            IG.append(find_entropy(df)-find_entropy_attribute(df,key))

    return df.keys()[:-1][np.argmax(IG)]
```

Step 4(d)

- Function that returns the sub-table from a given set that meet the condition.
- Code:

```
#Function to get a subtable of met conditions, node: Column name, value: Unique value of the column
def get_subtable(df, node, value):
    return df[df[node] == value].reset_index(drop=True)
```

Step 5: Construct the decision tree

Code:

```
    #Function to build the ID3 Decision Tree
def buildTree(df,tree=None):
    Class = df.keys()[-1]
    node = find_winner(df) #Get attribute with maximum information gain
    attValue = np.unique(df[node]) #Get distinct value of that attribute
    #Create an empty dictionary to create tree
    if tree is None:
        tree={}
        tree[node] = {}
   #We make loop to construct a tree by calling this function recursively. In this we check if the subset is pure stops if it is pure.
    for value in attValue:
        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable['Play'],return_counts=True)
        if len(counts)==1:#Checking purity of subset
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable) #Calling the function recursively
    return tree
tree = buildTree(df)
```

Step 6: Predict the labels

Code : #Function to predict for any input variable

```
def predict(inst,tree):
   #Recursively we go through the tree that we built earlier
   for nodes in tree.keys():
       value = inst[nodes]
       tree = tree[nodes][value]
       prediction = 0
       if type(tree) is dict:
           prediction = predict(inst, tree)
       else:
           prediction = tree
           break;
   return prediction
```

```
Y_label=[]

for i in range(len(df1)):

    inst =df1.iloc[i,:]

    prediction = predict(inst,tree)

    Y_label.append(prediction)
```

Step 7: Performance Evaluation

- We can check the performance using classification report and confusion metrics.

Code:

```
from sklearn import metrics
print(metrics.classification_report(Y_test,Y_label))
print(metrics.confusion_matrix(Y_test,Y_label))
```

Decision Tree Classifier- In built Function

- **Load Dataset**

```
from sklearn.datasets import load_iris
iris = load_iris()
X, Y = iris.data, iris.target
```

- **Train_Test_split**

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.8,random_state=42)
```

- **Fit Decision Tree model on Training set**

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='entropy')
clf = clf.fit(X_train, Y_train)
```

- **Predict Labels on Test Set**

```
Y_label1=clf.predict(X_test)
```

- **Performance Evaluation**

```
from sklearn import metrics
print(metrics.classification_report(Y_test,Y_label1))
```