Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2021-22

Experiment 2 - 3 (Decision Tree)

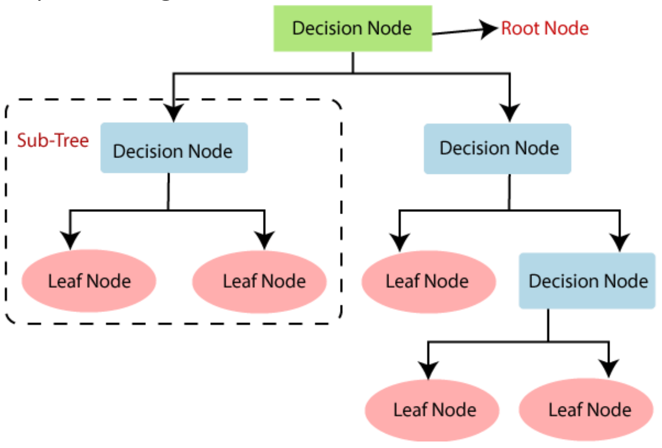
NAME: Dev Patel **SAP ID: 60009200016**

BATCH: K/K1 **DATE: 28/04/2022**

Aim: Implement Decision Tree on the given Dataset to build a classifier and Regressor.

Theory:

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.** In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.**
Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
The decisions or the test are performed on the basis of features of the given dataset.
It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees. Below diagram explains the general structure of a decision tree:

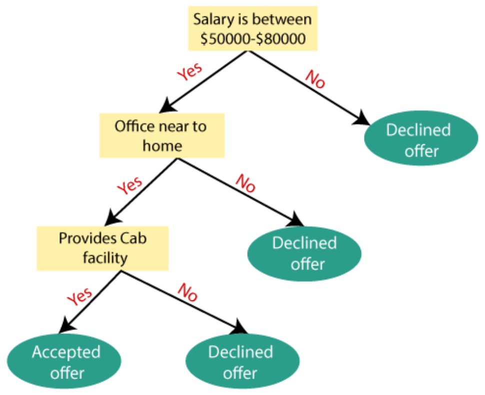


Decision Tree Terminologies

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
Branch/Sub Tree: A tree formed by splitting the tree.
Pruning: Pruning is the process of removing the unwanted branches from the tree.
Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

Steps in building a Tree
Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
Step-3: Divide the S into subsets that contains possible values for the best attributes.
Step-4: Generate the decision tree node, which contains the best attribute.
Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3.
Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures
While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute**

selection measure or ASM. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

1. Information Gain:

Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a class.

According to the value of information gain, we split the node and build the decision tree.

A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$Entropy(s)= -P(yes)\log2 P(yes)- P(no) \log2 P(no)$$

Where,

S= Total number of samples

P(yes)= probability of yes

P(no)= probability of no

2. Gini Index:

Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.

An attribute with the low Gini index should be preferred as compared to the high Gini index.

It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.

Gini index can be calculated using the below formula:

$$\text{Gini Index}= 1- \sum_j P_j^2$$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used: • Cost Complexity Pruning

- Reduced Error Pruning.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:

Dataset 1: PlayTennis.csv

Dataset 2: Iris.csv

Dataset 3: Breast Cancer.csv

Dataset 4: car prediction.csv

Department of Computer Science and Engineering (Data Science)

1. Implement Decision tree classifier from scratch on Dataset 1 by defining Node class and Tree class.
2. Use python libraries to build a decision tree classifier on Dataset 2. Analyze the results with different methods.
3. Discuss about overfitting on Dataset 3 using python libraries.
4. Implement Decision tree regressor from scratch on Dataset 4.

1. Implement Decision tree classifier from scratch on Dataset 1 (PlayTennis.csv)

In [172]:

```
import pandas as pd #for manipulating the csv data
import numpy as np #for mathematical calculation
```

In [173]:

```
train_data = pd.read_csv("PlayTennis.csv")
```

In [174]:

```
train_data.head()
```

Out[174]:

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes

In [175]:

```
def calc_total_entropy(train_data, label, class_list):
    total_row = train_data.shape[0]

    total_entr = 0

    for c in class_list:
        if total_row == 0:
            continue
        total_class_count = train_data[train_data[label] == c].shape[0]
        total_class_entr = - (total_class_count/total_row)*np.log2(total_class_count/tot
al_row)
        total_entr += total_class_entr

    return total_entr
```

In [176]:

```
def calc_entropy(feature_value_data, label, class_list):
    class_count = feature_value_data.shape[0]
    entropy = 0

    for c in class_list:
        label_class_count = feature_value_data[feature_value_data[label] == c].shape[0]

        entropy_class = 0
        if label_class_count != 0:
            probability_class = label_class_count/class_count
            entropy_class = - probability_class * np.log2(probability_class)

        entropy += entropy_class

    return entropy
```

In [177]:

```
def calc_info_gain(feature_name, train_data, label, class_list):
    feature_value_list = train_data[feature_name].unique()
    total_row = train_data.shape[0]
    feature_info = 0.0

    for feature_value in feature_value_list:
        feature_value_data = train_data[train_data[feature_name] == feature_value]
        feature_value_count = feature_value_data.shape[0]
        feature_value_entropy = calc_entropy(feature_value_data, label, class_list)
        feature_value_probability = feature_value_count/total_row
        feature_info += feature_value_probability * feature_value_entropy

    return calc_total_entropy(train_data, label, class_list) - feature_info
```

In [178]:

```
def find_most_informative_feature(train_data, label, class_list):
    feature_list = train_data.columns.drop(label)
    max_info_gain = -1
    max_info_feature = None

    for feature in feature_list:
        feature_info_gain = calc_info_gain(feature, train_data, label, class_list)
        if max_info_gain < feature_info_gain:
            max_info_gain = feature_info_gain
            max_info_feature = feature

    return max_info_feature
```

In [179]:

```
def generate_sub_tree(feature_name, train_data, label, class_list):
    feature_value_count_dict = train_data[feature_name].value_counts(sort=False)
    tree = {}

    for feature_value, count in feature_value_count_dict.iteritems():
        feature_value_data = train_data[train_data[feature_name] == feature_value]

        assigned_to_node = False
        for c in class_list:
            class_count = feature_value_data[feature_value_data[label] == c].shape[0]

            if class_count == count:
                tree[feature_value] = c
                train_data = train_data[train_data[feature_name] != feature_value]
                assigned_to_node = True
        if not assigned_to_node:
            tree[feature_value] = "?"

    return tree, train_data
```

In [180]:

```
def make_tree(root, prev_feature_value, train_data, label, class_list):
    if train_data.shape[0] != 0:
        max_info_feature = find_most_informative_feature(train_data, label, class_list)
        tree, train_data = generate_sub_tree(max_info_feature, train_data, label, class_
list)
        next_root = None

        if prev_feature_value != None:
            root[prev_feature_value] = dict()
            root[prev_feature_value][max_info_feature] = tree
            next_root = root[prev_feature_value][max_info_feature]
        else:
            root[max_info_feature] = tree
            next_root = root[max_info_feature]

        for node, branch in list(next_root.items()):
            if branch == "?":
                feature_value_data = train_data[train_data[max_info_feature] == node]
                make_tree(next_root, node, feature_value_data, label, class_list)
```

In [181]:

```
def id3(train_data_m, label):
    train_data = train_data_m.copy()
    tree = {}
    class_list = train_data[label].unique()
    make_tree(tree, None, train_data_m, label, class_list)

    return tree
```

In [182]:

```
def predict(tree, instance):
    if not isinstance(tree, dict):
        return tree
    else:
        root_node = next(iter(tree))
        feature_value = instance[root_node]
        if feature_value in tree[root_node]:
            return predict(tree[root_node][feature_value], instance)
        else:
            return None
```

In [183]:

```
def evaluate(tree, test_data_m, label):
    correct_preditct = 0
    wrong_preditct = 0
    for index, row in test_data_m.iterrows():
        result = predict(tree, test_data_m.iloc[index])
        if result == test_data_m[label].iloc[index]:
            correct_preditct += 1
        else:
            wrong_preditct += 1
    accuracy = correct_preditct / (correct_preditct + wrong_preditct)
    return accuracy
```

In [184]:

```
tree = id3(data,'play')
tree
```

Out[184]:

```
{'outlook': {'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}},
  'rainy': {'windy': {False: 'yes', True: 'no'}},
  'overcast': 'yes'}}
```

2. Use python libraries to build a decision tree classifier on Dataset 2. Analyze the results with different methods

In [185]:

```
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
import sklearn.tree
```

In [186]:

```
iris = pd.read_csv("Iris.csv")
```

In [187]:

```
iris.head()
```

Out[187]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species

0	1	5.1	3.5	1.4	0.2	Iris-setosa
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [188]:

```
iris = iris.drop(columns ="Id")
```

In [189]:

```
iris.shape
```

Out[189]:

```
(150, 5)
```

In [190]:

```
X = iris.values[:,:4]
Y = iris.values[5]
```

In [191]:

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
# 70% training and 30% test
```

In [192]:

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

In [193]:

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9555555555555556
```

In [194]:

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9555555555555556
```

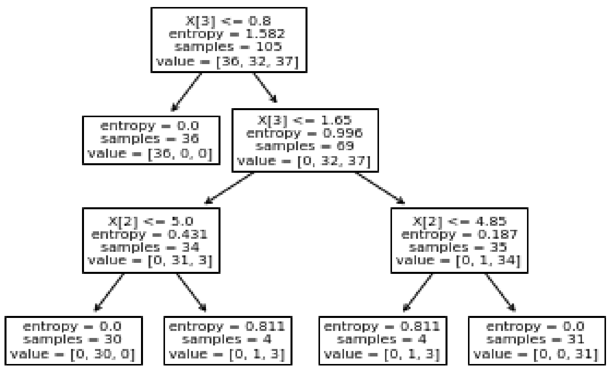
In [195]:

```
sklearn.tree.plot_tree(clf)
```

Out[195]:

```
[illegible]
```

```
[Text(125.55000000000001, 190.26, 'X[3] <= 0.8\nentropy = 1.582\nsamples = 105\nvalue = [
36, 32, 37]'),
 Text(83.7, 135.9, 'entropy = 0.0\nsamples = 36\nvalue = [36, 0, 0]'),
 Text(167.4, 135.9, 'X[3] <= 1.65\nentropy = 0.996\nsamples = 69\nvalue = [0, 32, 37]'),
 Text(83.7, 81.53999999999999, 'X[2] <= 5.0\nentropy = 0.431\nsamples = 34\nvalue = [0, 3
1, 3]'),
 Text(41.85, 27.180000000000007, 'entropy = 0.0\nsamples = 30\nvalue = [0, 30, 0]'),
 Text(125.55000000000001, 27.180000000000007, 'entropy = 0.811\nsamples = 4\nvalue = [0,
1, 3]'),
 Text(251.10000000000002, 81.53999999999999, 'X[2] <= 4.85\nentropy = 0.187\nsamples = 35
\nvalue = [0, 1, 34]'),
 Text(209.25, 27.180000000000007, 'entropy = 0.811\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(292.95, 27.180000000000007, 'entropy = 0.0\nsamples = 31\nvalue = [0, 0, 31]')]
```



In [196]:

```python
# Create Decision Tree classifer object using Gini Index
clf = DecisionTreeClassifier(criterion="gini", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

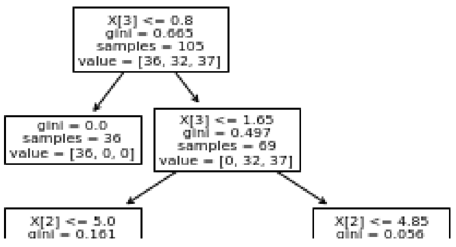
```
Accuracy: 0.9555555555555556
```

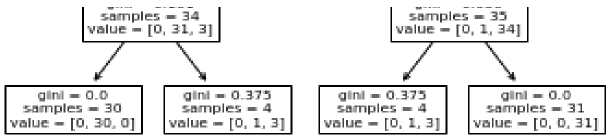
In [197]:

```python
sklearn.tree.plot_tree(clf)
```

Out[197]:

```
[Text(125.55000000000001, 190.26, 'X[3] <= 0.8\ngini = 0.665\nsamples = 105\nvalue = [36,
32, 37]'),
 Text(83.7, 135.9, 'gini = 0.0\nsamples = 36\nvalue = [36, 0, 0]'),
 Text(167.4, 135.9, 'X[3] <= 1.65\ngini = 0.497\nsamples = 69\nvalue = [0, 32, 37]'),
 Text(83.7, 81.53999999999999, 'X[2] <= 5.0\ngini = 0.161\nsamples = 34\nvalue = [0, 31,
3]'),
 Text(41.85, 27.180000000000007, 'gini = 0.0\nsamples = 30\nvalue = [0, 30, 0]'),
 Text(125.55000000000001, 27.180000000000007, 'gini = 0.375\nsamples = 4\nvalue = [0, 1,
3]'),
 Text(251.10000000000002, 81.53999999999999, 'X[2] <= 4.85\ngini = 0.056\nsamples = 35\nv
alue = [0, 1, 34]'),
 Text(209.25, 27.180000000000007, 'gini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(292.95, 27.180000000000007, 'gini = 0.0\nsamples = 31\nvalue = [0, 0, 31]')]
```




samples = 34
value = [0, 31, 3]
samples = 35
value = [0, 1, 34]
gini = 0.0
samples = 30
value = [0, 30, 0]
gini = 0.375
samples = 4
value = [0, 1, 3]
gini = 0.375
samples = 4
value = [0, 1, 3]
gini = 0.0
samples = 31
value = [0, 0, 31]