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)

NAME : Bhuvi Ghosh **SAPID** : 60009210191

Subject: Machine Learning – I (DJ19DSC402)

AY: 2022-23

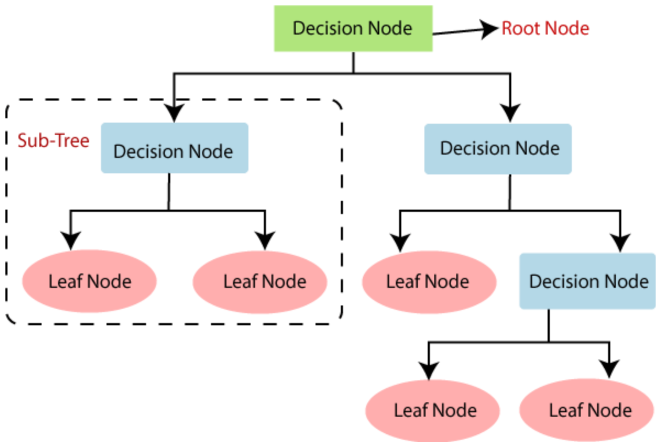
Experiment 2 - 3

(Decision Tree)

Aim: Implement Decision Tree on the given Datasets to build a classifier and Regressor. Apply appropriate pruning method to overcome overfitting.

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).**

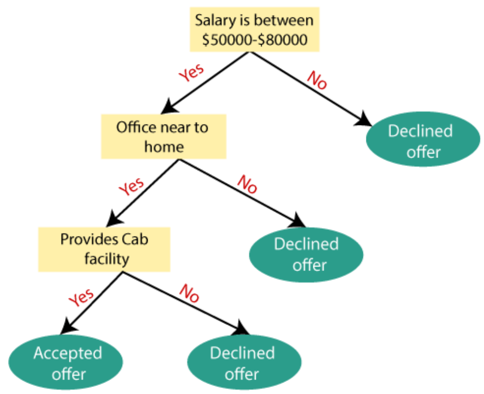
Step-3: Divide the S into subsets that contains possible values for the best attributes.

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:

$$Gini\ Index= 1- \sum_j P_j^2$$

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: Breastcancer.csv
Dataset 4: car prediction.csv

Department of Computer Science and Engineering (Data Science)

1. Implement Decision tree classifier from scratch using Dataset 1.

Code :

```
import numpy as np
import pandas as pd
```

```
[9] data = pd.read_csv("/content/PlayTennis.csv")
    data.head(10)
```

	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
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes

```
[10] class Node():
        def __init__(self, feature_index=None, threshold=None, left=None, right=None, info_gain=None, value=None):
            ''' constructor '''

            # for decision node
            self.feature_index = feature_index
            self.threshold = threshold
            self.left = left
            self.right = right
            self.info_gain = info_gain

            # for leaf node
            self.value = value
```

```
[11] class DecisionTreeClassifier():
        def __init__(self, min_samples_split=2, max_depth=2):
            ''' constructor '''

            # initialize the root of the tree
            self.root = None

            # stopping conditions
            self.min_samples_split = min_samples_split
            self.max_depth = max_depth

        def build_tree(self, dataset, curr_depth=0):
            ''' recursive function to build the tree '''

            X, Y = dataset[:,:-1], dataset[:,-1]
            num_samples, num_features = np.shape(X)

            # split until stopping conditions are met
            if num_samples>=self.min_samples_split and curr_depth<=self.max_depth:
```

Department of Computer Science and Engineering (Data Science)

```python
        if num_samples>=self.min_samples_split and curr_depth<=self.max_depth:
            # find the best split
            best_split = self.get_best_split(dataset, num_samples, num_features)
            # check if information gain is positive
            if best_split["info_gain"]>0:
                # recur left
                left_subtree = self.build_tree(best_split["dataset_left"], curr_depth+1)
                # recur right
                right_subtree = self.build_tree(best_split["dataset_right"], curr_depth+1)
                # return decision node
                return Node(best_split["feature_index"], best_split["threshold"],
                            left_subtree, right_subtree, best_split["info_gain"])

        # compute leaf node
        leaf_value = self.calculate_leaf_value(Y)
        # return leaf node
        return Node(value=leaf_value)

    def get_best_split(self, dataset, num_samples, num_features):
        ''' function to find the best split '''

        # dictionary to store the best split
        best_split = {}
        max_info_gain = -float("inf")    #sets max_info_gain to -ve infinity

        # loop over all the features
        for feature_index in range(num_features):
            feature_values = dataset[:, feature_index]
            possible_thresholds = np.unique(feature_values)
            # loop over all the feature values present in the data
            for threshold in possible_thresholds:
                # get current split
                dataset_left, dataset_right = self.split(dataset, feature_index, threshold)    #1st stump
```

```python
                # check if childs are not null
                if len(dataset_left)>0 and len(dataset_right)>0:
                    y, left_y, right_y = dataset[:, -1], dataset_left[:, -1], dataset_right[:, -1]
                    # compute information gain
                    curr_info_gain = self.information_gain(y, left_y, right_y)
                    # update the best split if needed
                    if curr_info_gain>max_info_gain:
                        best_split["feature_index"] = feature_index
                        best_split["threshold"] = threshold
                        best_split["dataset_left"] = dataset_left
                        best_split["dataset_right"] = dataset_right
                        best_split["info_gain"] = curr_info_gain
                        max_info_gain = curr_info_gain

        # return best split
        return best_split

    def split(self, dataset, feature_index, threshold):
        ''' function to split the data '''

        dataset_left = np.array([row for row in dataset if row[feature_index]<=threshold])    #binary split
        dataset_right = np.array([row for row in dataset if row[feature_index]>threshold])
        return dataset_left, dataset_right

    def information_gain(self, parent, l_child, r_child):
        ''' function to compute information gain '''

        weight_l = len(l_child) / len(parent)
        weight_r = len(r_child) / len(parent)
        gain = self.entropy(parent) - (weight_l*self.entropy(l_child) + weight_r*self.entropy(r_child))
        return gain
```

```
[11]  def entropy(self, y):
          ''' function to compute entropy '''

          class_labels = np.unique(y)
          entropy = 0
          for cls in class_labels:
              p_cls = len(y[y == cls]) / len(y)
              entropy += -p_cls * np.log2(p_cls)
          return entropy


      def calculate_leaf_value(self, Y):
          ''' function to compute leaf node '''

          Y = list(Y)
          return max(Y, key=Y.count)



      def fit(self, X, Y):
          ''' function to train the tree '''

          dataset = np.concatenate((X, Y), axis=1)
          self.root = self.build_tree(dataset)

      def predict(self, X):
          ''' function to predict new dataset '''

          preditions = [self.make_prediction(x, self.root) for x in X]
          return preditions

      def make_prediction(self, x, tree):
          ''' function to predict a single data point '''
```

```
[11]      ''' function to predict a single data point '''

          if tree.value!=None: return tree.value
          feature_val = x[tree.feature_index]
          if feature_val<=tree.threshold:
              return self.make_prediction(x, tree.left)
          else:
              return self.make_prediction(x, tree.right)
```

```
[12] X = data.iloc[:, :-1].values
     Y = data.iloc[:, -1].values.reshape(-1,1)
     from sklearn.model_selection import train_test_split
     X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.2, random_state=41)
```

```
[13] classifier = DecisionTreeClassifier(min_samples_split=3, max_depth=3)

     classifier.fit(X_train,Y_train)
```

```
[14] Y_pred = classifier.predict(X_test)
     from sklearn.metrics import accuracy_score
     accuracy_score(Y_test, Y_pred)

     0.6666666666666666
```

Q2)

[69]
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import tree
```

[70]
```
df=pd.read_csv('/content/Iris.csv')
```

```
df.head()
```

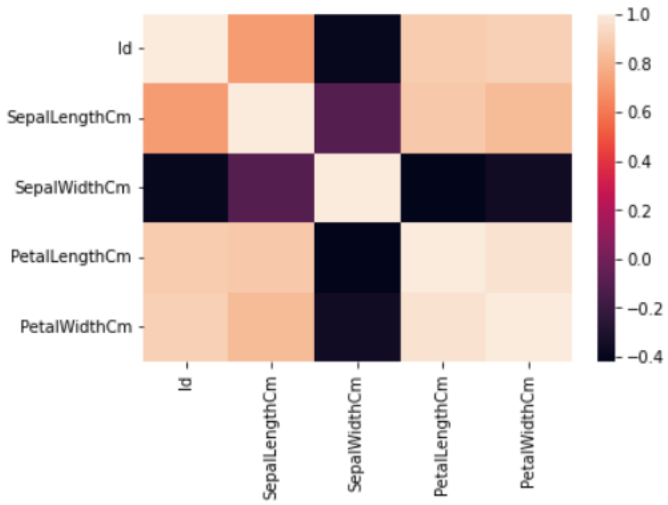
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

[72]
```
df.isnull().sum()
```

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

[73]
```
sns.heatmap(df.corr())
```

<AxesSubplot:>




```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['Species']= label_encoder.fit_transform(df['Species'])
```

[76]
```
x=df.iloc[:,1:5].values
y=df.iloc[:,5].values
```

[77]
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=23)
```

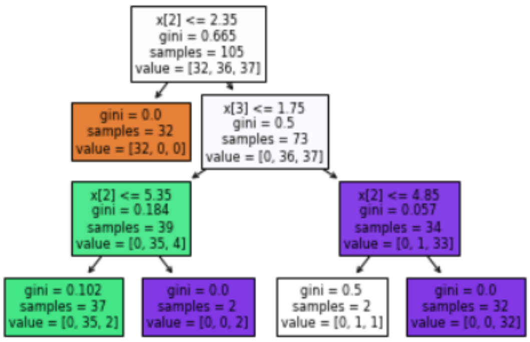
[78]
```
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier(max_depth=3)
```

[79]
```
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
```

[80]
```
from sklearn.metrics import accuracy_score,confusion_matrix
accuracy_score(y_test,y_pred)
```

```
0.9555555555555556
```

[81]
```
from sklearn import tree
tree1=tree.plot_tree(clf,filled=True)
```



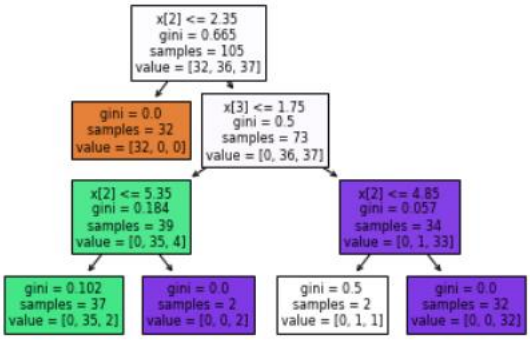
[82]
```
confusion_matrix(y_test,y_pred)
```

```
array([[18,  0,  0],
       [ 0, 14,  0],
       [ 0,  2, 11]])
```

[83]
```
y_pred=clf.predict(x_train)
accuracy_score(y_train,y_pred)
```

```
0.9714285714285714
```

```
[84] from sklearn import tree
     tree1= tree.plot_tree(clf,filled=True)
```




```
confusion_matrix(y_train,y_pred)
```

```
array([[32,  0,  0],
       [ 0, 36,  0],
       [ 0,  3, 34]])
```

Q3)

```
[86] df=pd.read_csv('/content/breastcancer.csv')
```

```
[87] df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280

5 rows × 33 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   id                        569 non-null    int64
 1   diagnosis                 569 non-null    object
 2   radius_mean               569 non-null    float64
```

```
[90] df.drop(columns='Unnamed: 32',axis=1,inplace=True)
```

```
[91] from sklearn import preprocessing

     # label_encoder object knows how to understand word labels.
     label_encoder = preprocessing.LabelEncoder()

     # Encode labels in column 'species'.
     df['diagnosis']= label_encoder.fit_transform(df['diagnosis'])
```

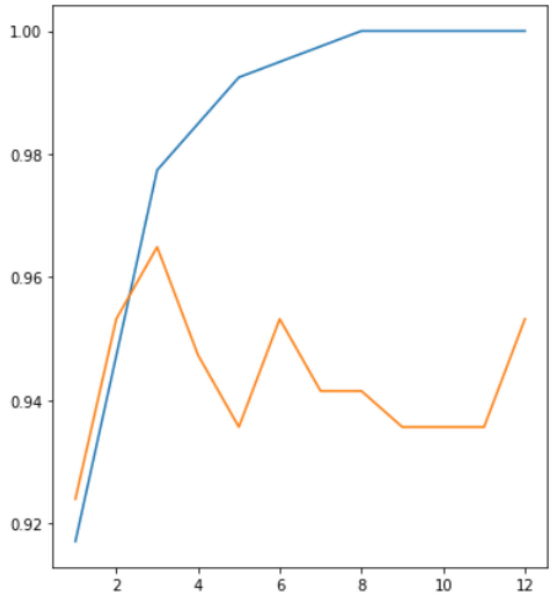
```
[92] x=df.drop('diagnosis',axis=1)
     y=df['diagnosis']
```

```
[93] l=[]
     x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=23)
     for i in range(1,13):
       model=DecisionTreeClassifier(max_depth=i)
       model.fit(x_train,y_train)
       y_pred=model.predict(x_test)
       l.append(accuracy_score(y_test,y_pred))
```

```
[95] z=[]
     for i in range(1,13):
       model=DecisionTreeClassifier(max_depth=i)
       model.fit(x_train,y_train)
       y_pred=model.predict(x_train)
       z.append(accuracy_score(y_train,y_pred))
```

Variation between training & testing accuracy:

```
plt.figure(figsize=(6,7))
plt.plot([i for i in range(1,13)],z)
plt.plot([i for i in range(1,13)],l)
plt.show()
```



Q4)

```
[150] df=pd.read_csv('/content/CarPrice_Assignment.csv')
```

```
[151] df.head()
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4

5 rows × 26 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
```

```
[153] df.drop(columns='car_ID',axis=1,inplace=True)
```

```
[154] categorical=df.select_dtypes(include='object').columns
```

```
[155] numeric=df.select_dtypes(include='int64').columns
```

```
[156] label_encoder = preprocessing.LabelEncoder()
```

```
[157] for i in categorical:
        df[i]= label_encoder.fit_transform(df[i])
```

```
df.isnull().sum()
```

```
symboling         0
CarName           0
fueltype          0
aspiration        0
doornumber        0
carbody           0
drivewheel        0
enginelocation    0
wheelbase         0
carlength         0
carwidth          0
carheight         0
curbweight        0
enginetype        0
cylindernumber    0
enginesize        0
```

```
[168] from sklearn.metrics import r2_score
```

```
[169] x=df.drop(columns='price',axis=1)
      y=df['price'].values
```

```
[170] x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=50,test_size=0.33)
```

```
[190] from sklearn.tree import DecisionTreeRegressor
      regressor = DecisionTreeRegressor(random_state=50,max_depth=5,min_samples_split=4)
      regressor.fit(x_train,y_train)
      y_pred=regressor.predict(x_test)
```

```
[191] r2_score(y_test,y_pred)

      0.874578268797094
```

```
[192] y_pred=regressor.predict(x_train)
      r2_score(y_train,y_pred)

      0.9705197556286744
```

```
from sklearn.model_selection import GridSearchCV
params = {'max_leaf_nodes': list(range(3,13)), 'min_samples_split': [2, 3, 4]}
grid_search_cv = GridSearchCV(DecisionTreeRegressor(random_state=42), params, verbose=1, cv=3)
grid_search_cv.fit(x_train, y_train)
```

```
[200] Fitting 3 folds for each of 30 candidates, totalling 90 fits
      ▸ GridSearchCV
      ▸ estimator: DecisionTreeRegressor
        ▸ DecisionTreeRegressor
```

```
[202] print(grid_search_cv.best_estimator_)

      DecisionTreeRegressor(max_leaf_nodes=5, random_state=42)
```

```
[212] regressor= DecisionTreeRegressor(random_state=50,max_depth=5)
      regressor.fit(x_train,y_train)
      y_pred=regressor.predict(x_test)
```

```
[213] r2_score(y_test,y_pred)

      0.8738548381655884
```

```
[214] y_pred=regressor.predict(x_train)
      r2_score(y_train,y_pred)

      0.9731762119701888
```

The model overfits as there is a significant difference between the r square score of train & test set.

```
[216] plt.figure(figsize=(20,10))
      tree1=tree.plot_tree(regressor,filled=True)
```

```
tree1=tree.plot_tree(regressor,filled=True)
plt.show()
```