# Exercise 1

## 1.1 Problem Statement:

Implement CART algorithm for decision tree learning. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

## 1.2 Description of the Algorithm:

Decision Trees are an important type of algorithm for predictive modeling machine learning. Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

CART is a decision tree algorithm and it stands for Classification and Regression Trees. CART can be used for classification or regression predictive modeling problems. The representation for the CART model is a binary tree. It builds decision trees based on the attribute selection measure "Gini's Impurity Index". CART is an umbrella word that refers to the following types of decision trees:

Classification Trees: These are used to forecast the value of a categorical target variable.

Regression trees: These are used to forecast the value of a continuous target variable.

Advantages of Decision Trees:

- Decision trees are simple to understand, interpret, visualize.
- Decision trees implicitly perform variable screening or feature selection.
- Decision trees require little data preparation.
- Decision trees can handle both numerical and categorical data.
- Decision trees can also handle single-class and multi-class classification problems.
- Nonlinear relationships between parameters do not affect tree performance.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

Disadvantages of Decision Trees:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging, boosting etc.
- Predictions of decision trees are neither smooth nor continuous, but are piecewise constant approximations. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

Advantages of CART algorithm:

- CART can handle missing values in the data.
- CART is not significantly impacted by outliers in the input variables.
- CART is non-parametric; Thus it does not depend on information from a certain sort of distribution.
- CART supports growing a full decision tree and further post-pruning the decision tree so that the probability that important structure in the data set will be overlooked by stopping too soon is minimized.
- CART combines both testing with a test data set and cross-validation to more precisely measure the goodness of fit.
- CART allows to utilize the same attributes many times in various regions of the tree. This skill is capable of revealing intricate interdependencies between groups of variables.
- CART can be used in conjunction with other prediction methods to select the input set of variables.

## 1.3 Description of the Dataset:

Title of the dataset: Iris Plants Database

The Iris Dataset contains information of three species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor). The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Data Set Characteristics: Multivariate

Area: Life Sciences

Number of samples (or instances) in the Dataset: 150

Number of attributes (or features) in the Dataset: 05

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   - -- Iris Setosa
   - -- Iris Versicolour
   - -- Iris Virginica
6. Number of samples of each species of iris flowers:

   Class Distribution: 33.3% for each of 3 classes.
   50 (Setosa), 50 (Versicolor), 50 (Virginica)

Predicted Attribute: class of iris plant

Missing Attribute Values: None

Source of Dataset: sklearn.datasets

| Feature Name | Units of measurement | Datatype | Description |
|---|---|---|---|
| sepal length | Centimeters | Real (Numerical) | Length of Iris flower's sepal |
| sepal width length | Centimeters | Real (Numerical) | Width of Iris flower's sepal |
| petal length | Centimeters | Real (Numerical) | Length of Iris flower's petal |
| petal width length | Centimeters | Real (Numerical) | Width of Iris flower's petal |
| variety | Variety of species [Setosa, Virginica, Versicolor] | Object (Categorical) | Variety of the species of Iris flower |

## 1.4 Description of the Machine Learning Library Classes and Methods used:

The **scikit-learn** (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It provides simple and efficient tools for predictive data analysis. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting,k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

## sklearn.model_selection.train_test_split()

The **sklearn.model_selection.train_test_split()** method splits arrays or matrices into random train and test subsets. The parameters for the method are as follows:

sklearn.model_selection.train_test_split(**arrays*, *test_size=None*, *train_size=None*, *random_state=None*, *shuffle=True*, *stratify=None*)

It returns lists containing train-test split of inputs.

## sklearn.tree.DecisionTreeClassifier()

The **sklearn.tree.DecisionTreeClassifier** is a class capable of performing multi-class classification on a dataset. It takes as input two arrays: an array X, holding the training samples, and an array Y holding the class labels for the training samples. In case that there are multiple classes with the same and highest probability, the classifier will predict the class with the lowest index amongst those classes. As an alternative to outputting a specific class, the probability of each class can be predicted, which is the fraction of training samples of the class in a leaf. DecisionTreeClassifier is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification. The parameters of the DecisionTreeClassifier class are as follows:

*class* sklearn.tree.DecisionTreeClassifier(***, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*)

The default values for the parameters controlling the size of the trees (e.g. max_depth, min_samples_leaf, etc.) lead to fully grown and unpruned trees which can potentially be very large on some data sets.

<u>Limitations of sklearn.tree.DecisionTreeClassifier class:</u>

- scikit-learn implementation does not support categorical variables for now.
- scikit-learn implementation can build only CART trees for now.

The **DecisionTreeClassifier.fit()** method builds a decision tree classifier from the training set (X, y). The parameters of the fit() method are as follows:

fit(*X*, *y*, *sample_weight=None*, *check_input=True*, *X_idx_sorted='deprecated'*)

It returns an fitted Decision Tree estimator.

The **DecisionTreeClassifier.predict()** method predicts class or regression value for X. The parameters of the predict() method are as follows:

predict(*X*, *check_input=True*)

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

**sklearn.tree.plot_tree()**

The **sklearn.tree** class supports visualization of decision trees. Once a Decision Tree Classifier is built it can be visualized in test representation using**sklearn.tree.export_text()** method. The parameters for the method are as follows:

sklearn.tree.export_text(*decision_tree*, *, *feature_names=None*, *max_depth=10*, *spacing=3*, *decimals=2*, *show_weights=False*)

Alternatively, the decision tree can be plotted **sklearn.tree.plot_tree()** method. The parameters for the method are as follows:

sklearn.tree.plot_tree(*decision_tree*, *, *max_depth=None*, *feature_names=None*, *class_names=None*, *label='all'*, *filled=False*, *impurity=True*, *node_ids=False*, *proportion=False*, *rounded=False*, *precision=3*, *ax=None*, *fontsize=None*)

**sklearn.metrics.accuracy_score()**

Once a Decision Tree Classifier is built, it can be evaluated for performance. The **sklearn.metrics** module implements several loss, score, and utility functions to measure classification performance. The **sklearn.metrics.accuracy_score()** method computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true. The parameters for the method are as follows:

sklearn.metrics.accuracy_score(*y_true*, *y_pred*, *, *normalize=True*, *sample_weight=None*)

## 1.5 Implementation

```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np

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

**o/p:**

```
     sepal.length  sepal.width  petal.length  petal.width    variety
0             5.1          3.5           1.4          0.2     Setosa
1             4.9          3.0           1.4          0.2     Setosa
2             4.7          3.2           1.3          0.2     Setosa
3             4.6          3.1           1.5          0.2     Setosa
4             5.0          3.6           1.4          0.2     Setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica

[150 rows x 5 columns]
```

```
# explore the shape of the dataset
iris.shape

(150, 5)

# explore the information of the dataset

iris.info()
```

**o/p:**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column         Non-Null Count   Dtype
---  -----------    --------------   ---------
 0   sepal.length   150 non-null     float64
 1   sepal.width    150 non-null     float64
 2   petal.length   150 non-null     float64
 3   petal.width    150 non-null     float64
 4   variety        150 non-null     object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

The decision tree model is to be constructed in order to "predict the variety of the iris flowers given their sepal & petal length & width". Hence, "variety" attribute is Target variable. The attributes "sepal.length" , "sepal.width" , "petal.length", "petal.width" are the independent variables.

Extracting Independent variables "sepal.length" , "sepal.width" , "petal.length", "petal.width"

X is a dataframe comprising of Independent variables data.

```
X = iris.iloc[ : , 0:4]

# display X dataframe
X
```

**o/p:**

```
     sepal.length  sepal.width  petal.length  petal.width
0             5.1          3.5           1.4          0.2
1             4.9          3.0           1.4          0.2
2             4.7          3.2           1.3          0.2
3             4.6          3.1           1.5          0.2
4             5.0          3.6           1.4          0.2
..            ...          ...           ...          ...
145           6.7          3.0           5.2          2.3
146           6.3          2.5           5.0          1.9
147           6.5          3.0           5.2          2.0
148           6.2          3.4           5.4          2.3
149           5.9          3.0           5.1          1.8

[150 rows x 4 columns]
```

Extracting Dependent variables "variety"

Y is a dataframe comprising of Dependent variable's data.

```
Y = iris.iloc[ : , 4: ]

# display Y dataframe
Y.variety.unique()
```

**o/p:**

```
array(['Setosa', 'Versicolor', 'Virginica'], dtype=object)
```

**Implementing CART algorithm for decision tree learning.**

```
# construct the decision tree model
# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X ,
                                                    Y,
                                                    test_size = 0.25,
                                                    random_state = 5)
X_train
```

**o/p:**

```
     sepal.length  sepal.width  petal.length  petal.width
40            5.0          3.5           1.3          0.3
115           6.4          3.2           5.3          2.3
142           5.8          2.7           5.1          1.9
69            5.6          2.5           3.9          1.1
17            5.1          3.5           1.4          0.3
..            ...          ...           ...          ...
8             4.4          2.9           1.4          0.2
73            6.1          2.8           4.7          1.2
144           6.7          3.3           5.7          2.5
118           7.7          2.6           6.9          2.3
99            5.7          2.8           4.1          1.3

[112 rows x 4 columns]
```

```
# construct the decision tree model

# create an instance "DecisionTreeClassifier" class
# set the criterion to be "entropy"


from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state = 1234, criterion = 'entropy')


# fit() method will construct the decision tree
# by fitting the given training dataset.

clf.fit(X_train , Y_train)


DecisionTreeClassifier(criterion='entropy', random_state=1234)

# Visualize the decision tree as text representation

from sklearn import tree

text_representation = tree.export_text(clf)

print(text_representation)
```

**o/p:**

```
|--- feature_2 <= 2.45
|   |--- class: Setosa
```

```
|--- feature_2 >  2.45
|   |--- feature_3 <= 1.75
|   |   |--- feature_3 <= 1.45
|   |   |   |--- class: Versicolor
|   |   |--- feature_3 >  1.45
|   |   |   |--- feature_1 <= 2.60
|   |   |   |   |--- feature_0 <= 6.10
|   |   |   |   |   |--- class: Virginica
|   |   |   |   |--- feature_0 >  6.10
|   |   |   |   |   |--- class: Versicolor
|   |   |   |--- feature_1 >  2.60
|   |   |   |   |--- feature_0 <= 7.05
|   |   |   |   |   |--- class: Versicolor
|   |   |   |   |--- feature_0 >  7.05
|   |   |   |   |   |--- class: Virginica
|   |--- feature_3 >  1.75
|   |   |--- class: Virginica
```

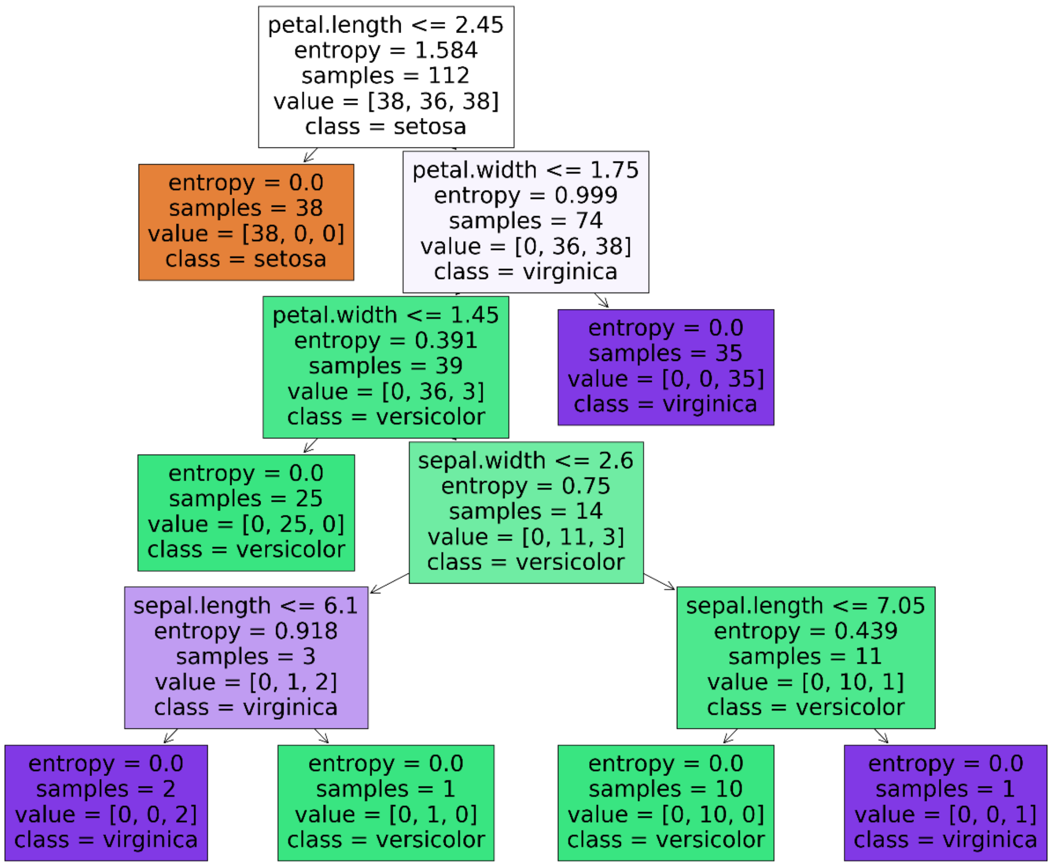
plot_tree() will work only from sklearn version 0.21 and above.

```
# visualizing the decision tree pictorially

fig = plt.figure(figsize = (25 , 20) , dpi = 200.0)

_ = tree.plot_tree(clf,
                   feature_names = ['sepal.length' ,'sepal.width',
'petal.length', 'petal.width'],
                   class_names = ['setosa', 'versicolor', 'virginica'],
                   filled = True)
```

**o/p:**



```
# Now, Let us test the accuracy of the decision tree model
# on the training data and on test data.
# predict() method predicts classes for test dataset
# accuracy_score() will take original class labels,  predicted class labels
and
#  computes accuracy of the model.


# Let us first test the accuracy of the model on the training data itself.


from sklearn.metrics import accuracy_score

pred_train = clf.predict(X_train)

accuracy_train = accuracy_score(Y_train, pred_train)

print('% of Accuracy on training data: ', accuracy_train * 100 )



# Let us test the accuracy of the model on the test data (or new data or
unseen data).

pred_test = clf.predict(X_test)

accuracy_test = accuracy_score(Y_test, pred_test)

print('% of Accuracy on test data: ', accuracy_test * 100 )
```

**o/p:**

```
% of Accuracy on training data:  100.0
% of Accuracy on test data:  92.10526315789474
```

It can be observed that the model performs almost perfect on the training dataset. But its ability to make predictions on new data is less.

**Applying the Decision Tree model to classify a new sample.**

```
# Use the decision tree model to predict class label of new sample i.e.,
classify new sample.

# creating new iris flower's data and loading into Dataframe

new_data = {'sepal.length' : [3.7],
            'sepal.width' :  [3.0],
            'petal.length' : [2.2],
            'petal.width' : [1.3] }

new_df = pd.DataFrame(new_data)

new_df.head()
```

**o/p:**

```
   sepal.length  sepal.width  petal.length  petal.width
0           3.7          3.0           2.2          1.3
```

```
new_pred = clf.predict(new_df)

print('Prediction: The variety of the iris flower is ' , new_pred)
```

**o/p:**

```
Prediction: The variety of the iris flower is  ['Setosa']
```

### 1..6 Results and Discussion

- ✓ CART algorithm has been implemented for decision tree learning using scikit-learn machine learning library.
- ✓ Iris dataset has been used and decision tree machine learning model has been constructedwith a height of 5 and number of nodes in the tree were 13.
- ✓ The accuracy score of the decision tree was 92.1%.

Hence, a decision tree model has been trained using the iris flowers data that predicts the species of the new iris flower