Experiment No. 5

Classifier Models: Decision Trees (CART), Random Forests

<u>OBJECTIVE:</u>

- To be able to use Python Libraries for Classifier Models.
- To be able to use Classifier Models for supervised machine learning tasks of classification.

Decision Trees:

A decision tree is a flowchart-like representation, with internal nodes representing features, branches representing rules, and leaf nodes representing algorithm results This versatile supervised machine-learning algorithm applies to both classification and regression problems, i.e and power. Decision trees are valued for their interpretability as the rules they generate are easy to understand.

CART: CART is a predictive algorithm used in Machine learning and it explains how the target variable's values can be predicted based on other matters. It is a decision tree where each fork is split into a predictor variable and each node has a prediction for the target variable at the end.

The term CART serves as a generic term for the following categories of decision trees:

Classification Trees: The tree is used to determine which "class" the target variable is most likely to fall into when it is continuous.
Regression trees: These are used to predict a continuous variable's value.

How does CART algorithm works?

Classification and Regression Trees (CART) is a decision tree algorithm that is used for both classification and regression tasks. It is a supervised learning algorithm that learns from labelled data to predict unseen data.

- Tree structure: CART builds a tree-like structure consisting of nodes and branches. The nodes represent different decision points, and the branches represent the possible outcomes of those decisions. The leaf nodes in the tree contain a predicted class label or value for the target variable.
- Splitting criteria: CART uses a greedy approach to split the data at each node. It evaluates all possible splits and selects the one that **best reduces the impurity of the resulting subsets**. For classification tasks, CART uses Gini impurity as the splitting criterion. **<u>The lower the Gini impurity, the more pure the subset is</u>**. For regression tasks, CART uses residual reduction as the splitting criterion. The lower the residual reduction, the better the fit of the model to the data.
- Pruning: To prevent overfitting of the data, pruning is a technique used to remove the nodes that contribute little to the model accuracy. Cost complexity pruning and information gain pruning are two popular pruning techniques. Cost complexity pruning involves calculating the cost of each node and removing nodes that have a negative cost. Information gain

pruning involves calculating the information gain of each node and removing nodes that have a low information gain.

The CART algorithm works via the following process:

1. The best-split point of each input is obtained.
2. Based on the best-split points of each input in Step 1, the new "best" split point is identified.
3. Split the chosen input according to the "best" split point.
4. Continue splitting until a stopping rule is satisfied or no further desirable splitting is available.

CART algorithm uses Gini Impurity to split the dataset into a decision tree. It does that by searching for the best homogeneity for the sub nodes, with the help of the Gini index criterion.

The Gini index is a metric for the classification tasks in CART. **It stores the sum of squared probabilities of each class**. It computes the degree of probability of a specific variable that is wrongly being classified when chosen randomly and a variation of the Gini coefficient. It works on categorical variables, provides outcomes either "successful" or "failure" and hence conducts binary splitting only.

The degree of the Gini index varies from 0 to 1,

- Where 0 depicts that all the elements are allied to a certain class, or only one class exists there.
- Gini index close to 1 means a high level of impurity, where each class contains a very small fraction of elements, and
- A value of 1-1/n occurs when the elements are uniformly distributed into n classes and each class has an equal probability of 1/n. For example, with two classes, the Gini impurity is 1 – 1/2 = 0.5.

Mathematically, we can write Gini Impurity as follows:

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

where pi is the probability of an object being classified to a particular class.

Decision Tree Classifier (Built-in):

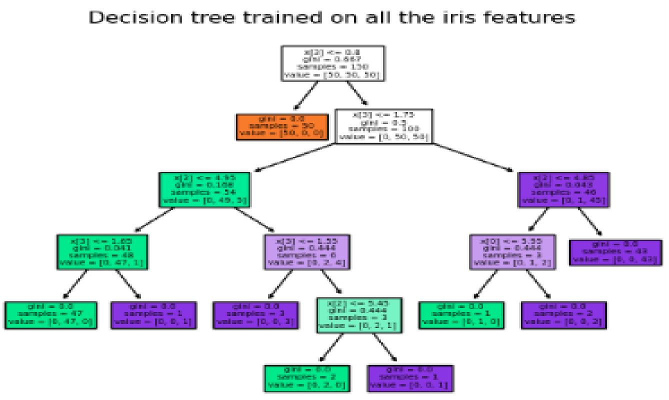
DecisionTreeClassifier is a class capable of performing multi-class classification on a dataset.
As with other classifiers, DecisionTreeClassifier takes as input two arrays: an array X, sparse or dense, of shape (n_samples, n_features) holding the training samples, and an array Y of integer values, shape (n_samples,), holding the class labels for the training samples:

DecisionTreeClassifier is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification.
Using the Iris dataset, we can construct a tree as follows:

```
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, y)
```

Once trained, you can plot the tree with the plot_tree function**:**

```
>>> tree.plot_tree(clf)
[...]
```



Decision Tree Regressor (Built-in):

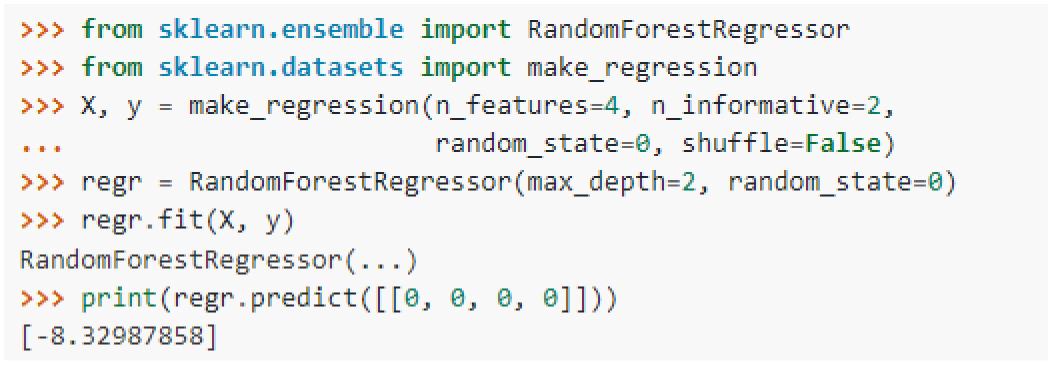
Decision trees can also be applied to regression problems, using the DecisionTreeRegressor class. As in the classification setting, the fit method will take as argument arrays X and y, only that in this case y is expected to have floating point values instead of integer values:

```
>>> from sklearn import tree
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = tree.DecisionTreeRegressor()
>>> clf = clf.fit(X, y)
>>> clf.predict([[1, 1]])
array([0.5])
```

Random Forest:

A random forest is a meta estimator that fits a number of decision tree classifiers/regressors on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                            n_informative=2, n_redundant=0,
...                            random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(...)
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```



```
>>> from sklearn.ensemble import RandomForestRegressor
>>> from sklearn.datasets import make_regression
>>> X, y = make_regression(n_features=4, n_informative=2,
...                        random_state=0, shuffle=False)
>>> regr = RandomForestRegressor(max_depth=2, random_state=0)
>>> regr.fit(X, y)
RandomForestRegressor(...)
>>> print(regr.predict([[0, 0, 0, 0]]))
[-8.32987858]
```

Exercise 1:

Read the given data by using Pandas Library and write the python code (Using Built-in Libraries) to perform the **classification with X1 and X2 as the Input variables and Y as the target.** Use the classification algorithms above and compare the results in terms of the accuracy.

Training Set:

X1	X2	Y
60	22	0
62	25	0
67	24	0
70	20	0
71	15	1
72	14	1
75	14	1
78	11	1

Test Set:

X1	X2	Y
61	23	0
71	19	0
73	15	1
79	13	1

Which algorithm performs the best and why?

Exercise 2:

Read the given data by using Pandas Library and apply CART and Random Forest Regressors to predict the value for the test set.

Training Set:

X1	X2	Y
60	22	140
62	25	155
67	24	159
70	20	179
71	15	192

Test Set:

X1	X2	Y
72	14	200
75	14	212
78	11	215

You may use following to report accuracy metrics

```
#y_pred for Confusion Matrix  :
y_pred = cart_model.predict(X)
```

```
#y_prob for AUC:
y_prob = cart_model.predict_proba(X)[:, 1]
```

```
# Confusion matrix
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       500
           1       1.00      1.00      1.00       268

    accuracy                           1.00       768
   macro avg       1.00      1.00      1.00       768
weighted avg       1.00      1.00      1.00       768
```