# Assignment 3

**Topic: -** Implementation of Tree based Classifiers (Decision tree). Apply K-fold Cross Validation.

**Algorithms used:** Decision tree

**Theory:-**

## 1. Introduction:

Decision trees are a popular and intuitive machine learning technique used for classification and regression tasks. They are powerful tools for modeling complex decision-making processes and are widely employed in various domains such as finance, healthcare, and marketing. In this write-up, we provide an overview of decision trees, including their definition, working principle, types, and methodology.

## 2. What is a Decision Tree and How They Work?

A decision tree is a supervised learning algorithm that learns a series of if-else decision rules based on the features of the input data. It recursively partitions the feature space into subsets, each associated with a specific class label or a numerical value (for regression tasks). At each node of the tree, a decision is made based on the value of a feature, leading to the traversal of the tree until a leaf node is reached, which corresponds to the final decision or prediction.

The process of constructing a decision tree involves selecting the best feature to split the data at each node. This selection is typically based on criteria such as information gain (for classification tasks) or variance reduction (for regression tasks). The goal is to maximize the homogeneity of the subsets produced by the splits, resulting in pure nodes where all instances belong to the same class or have similar numerical values.

## 3. Types of Decision Trees

There are several types of decision trees, including:

a) Classification Trees: Used for predicting categorical class labels.

b) Regression Trees: Used for predicting numerical values.

### (a) Classification Trees:

Classification trees are a type of decision tree used for solving classification problems, where the target variable is categorical. The goal of a classification tree is to partition the feature space into regions that are as homogeneous as possible with respect to the class labels.

**Working Principle:**

Splitting Criteria: At each node of the tree, the algorithm evaluates different splitting criteria (e.g., information gain, Gini impurity) to determine the best feature and threshold for partitioning the data. The goal is to maximize the purity of the resulting subsets, ensuring that each subset contains predominantly instances of the same class.

Recursive Partitioning: The tree-building process continues recursively until certain stopping criteria are met, such as reaching a maximum depth, minimum number of samples per leaf, or no further improvement in impurity reduction.

Leaf Nodes: Once a stopping criterion is reached or no further splits can improve purity significantly, the algorithm assigns a class label to the leaf node based on the majority class of the instances in that node.

**Example:**

Let's consider a binary classification problem where we want to predict whether a person will buy a product based on their age and income level. We have a dataset with the following features: Age (continuous) and Income (continuous), and the target variable is the purchase decision (Yes or No).

Initial Split: The algorithm evaluates different splitting criteria and selects the feature and threshold that best separates the data into two subsets. For example, it may split the data based on age ≤ 40.

Recursive Partitioning: The process continues recursively for each subset, evaluating different splitting criteria until the stopping criteria are met.

Leaf Nodes: Once the tree is constructed, each leaf node represents a decision (e.g., Yes or No) based on the majority class of the instances in that node.

```
                                        Root Node (All data)
```

```
                    /              \
                 Split 1        Split 2
                /                    \
          Age <= 40               Age > 40
         /          \            /          \
Income<=30k      Inc>30k    Inc<=30k     Inc>30k
  /  \             /  \        /  \         /  \
Buy  NoBuy    Buy  NoBuy  Buy  NoBuy Buy  No Buy
```

(Leaf nodes with majority class label)

**Working Principle:**

i) Start with the root node containing all data.

ii) Choose the feature and threshold that best separates the data into two subsets based on the splitting criteria (e.g., information gain). In this example, we split on Age <= 40.

iii) Repeat steps 2 and 3 recursively for each child node until a stopping criterion is met (e.g., maximum depth, minimum samples per leaf).

iv) Each leaf node represents a region with mostly instances of the same class. Assign the majority class label to the leaf node.

(b) Regression Trees:

Regression trees are another type of decision tree used for solving regression problems, where the target variable is continuous. The goal of a regression tree is to partition the feature space into regions that minimize the variance of the target variable within each region.

**Working Principle:**

Splitting Criteria: Similar to classification trees, regression trees evaluate different splitting criteria (e.g., variance reduction) to determine the best feature and threshold for partitioning the data. The goal is to minimize the variance of the target variable within each resulting subset.

Recursive Partitioning: The tree-building process continues recursively until certain stopping criteria are met, such as reaching a maximum depth, minimum number of samples per leaf, or no further reduction in variance.

Leaf Nodes: Once a stopping criterion is reached or no further splits can significantly reduce variance, the algorithm assigns a numerical value to the leaf node based on the average (or median) of the target variable in that node.

**Example:**

Consider a regression problem where we want to predict the price of a house based on its features such as the number of bedrooms, bathrooms, and square footage. We have a dataset with the following features: Bedrooms (integer), Bathrooms (integer), Square Footage (continuous), and the target variable is the Price (continuous).

Initial Split: The algorithm evaluates different splitting criteria and selects the feature and threshold that best separates the data into two subsets, minimizing the variance of the target variable within each subset.

Recursive Partitioning: The process continues recursively for each subset, evaluating different splitting criteria until the stopping criteria are met.

Leaf Nodes: Once the tree is constructed, each leaf node represents a predicted price based on the average (or median) of the target variable in that node.

***Working:**

i) Start with the root node containing all data.

ii) Choose the feature and threshold that best splits the data into two subsets, minimizing the variance (or another metric) of the target variable within each subset. In this example, we split on Bedrooms <= 2.

iii) Repeat steps 2 and 3 recursively for each child node until a stopping criterion is met.

iv) Each leaf node represents a region with low variance in the target variable. Assign the average (or median) value of the target variable to the leaf node.

## 4. Methodology

The methodology for building a decision tree involves the following steps:

Step 1: Data Preparation: Preprocess the dataset by handling missing values, encoding categorical variables, and splitting the data into training and testing sets.

Step 2: Tree Construction: Recursively partition the feature space by selecting the best split at each node based on a chosen criterion (e.g., information gain, Gini impurity).

Step 3: Stopping Criteria: Define conditions for terminating the tree-growing process, such as reaching a maximum depth, minimum number of samples per node, or no further improvement in impurity reduction.

Step 4: Pruning (Optional): Simplify the decision tree by removing nodes that do not contribute significantly to its predictive performance, thereby reducing overfitting.

Step 5: Evaluation: Evaluate the performance of the decision tree model using appropriate metrics such as accuracy, precision, recall, or mean squared error, depending on the task (classification or regression).

Step 6: Tuning: Fine-tune the hyperparameters of the decision tree algorithm (e.g., maximum depth, minimum samples per leaf) using techniques such as cross-validation or grid search to optimize performance.

## 5. Conclusion

Decision trees are versatile and interpretable machine learning models that offer a straightforward approach to solving classification and regression problems. By recursively partitioning the feature space based on simple if-else decision rules, decision trees can effectively capture complex decision boundaries and make accurate predictions. Understanding the principles, types, and methodology of decision trees is essential for effectively applying them in real-world machine learning tasks.

In conclusion, decision trees are powerful tools in the machine learning toolbox, offering a balance between interpretability and predictive performance. As with any machine learning algorithm, careful consideration of data preprocessing, model selection, and parameter tuning is crucial for achieving optimal results.

**Advantages:**

1. Compared to other algorithms decision trees requires less effort for data preparation during pre-processing.
2. A decision tree does not require normalization of data.
3. A decision tree does not require scaling of data as well.
4. Missing values in the data also do NOT affect the process of building a decision tree to any considerable extent.
5. A Decision tree model is very intuitive and easy to explain to technical teams as well as stakeholders.

**Disadvantages:**

1. A small change in the data can cause a large change in the structure of the decision tree causing instability.
2. For a Decision tree sometimes calculation can go far more complex compared to other algorithms.
3. Decision tree often involves higher time to train the model.
4. Decision tree training is relatively expensive as the complexity and time has taken are more.
5. The Decision Tree algorithm is inadequate for applying regression and predicting continuous values.

## Methodology:

We will start by loading the dataset and splitting it into training and testing sets using a 80/20 split. We will then fit a decision tree classifier to the training data using the default hyperparameters in scikit-learn. We will evaluate the performance of our model on the testing data using the accuracy score and the confusion matrix.

**Implementation:-**

1. Read the .csv file of dataset
2. Display few observations
3. Perform data preprocessing(handling missing data, etc)
4. Create the independent and dependent variables
5. Standardization of data
6. Plot few graphs to understand/explore the data.
7. Perform feature importance and find the most significant features.
8. Split the data into training and test sets.
9. Create the objects of classifiers.
10. Fit the data in model to train it.
11. Analyze the performance of the classifiers.

## **k-Fold Cross-Validation:**

k-Fold Cross-Validation is a widely used technique in machine learning for evaluating the performance of models. By systematically partitioning the dataset into multiple subsets and iteratively training and evaluating the model, k-Fold Cross-Validation provides a robust estimate of its performance and generalization ability. This approach helps in identifying potential issues such as overfitting or underfitting and enables practitioners to make more informed decisions during the model development process.

In k-Fold Cross-Validation, we assess the performance of a machine learning model by partitioning the dataset into k subsets (or folds). The model is trained and evaluated k times, with each fold serving as the testing set once while the remaining k-1 folds are used for training. This process helps in estimating the model's performance more reliably compared to a single train-test split.

Methodology of k-Fold Cross-Validation:

Data Partitioning:

The dataset is divided into k approximately equal-sized subsets (folds).
Each fold contains a similar distribution of instances from the original dataset, ensuring representative sampling.
If necessary, the dataset may be shuffled prior to partitioning to avoid any ordering bias.
Training and Evaluation:

For each iteration i (1 to k):
One fold is designated as the testing set, denoted as fold i.
The model is trained on the remaining k-1 folds, which collectively form the training set.
The trained model is then evaluated on the instances in fold i, using appropriate evaluation metrics.

This process is repeated k times, each time using a different fold as the testing set and the remaining folds as the training set.
Performance Aggregation:

After completing the k iterations, k performance scores are obtained, one for each fold.
These performance scores are typically averaged to obtain the final performance estimate of the model.
The averaged performance metric provides a more reliable estimate of the model's performance compared to a single train-test split.

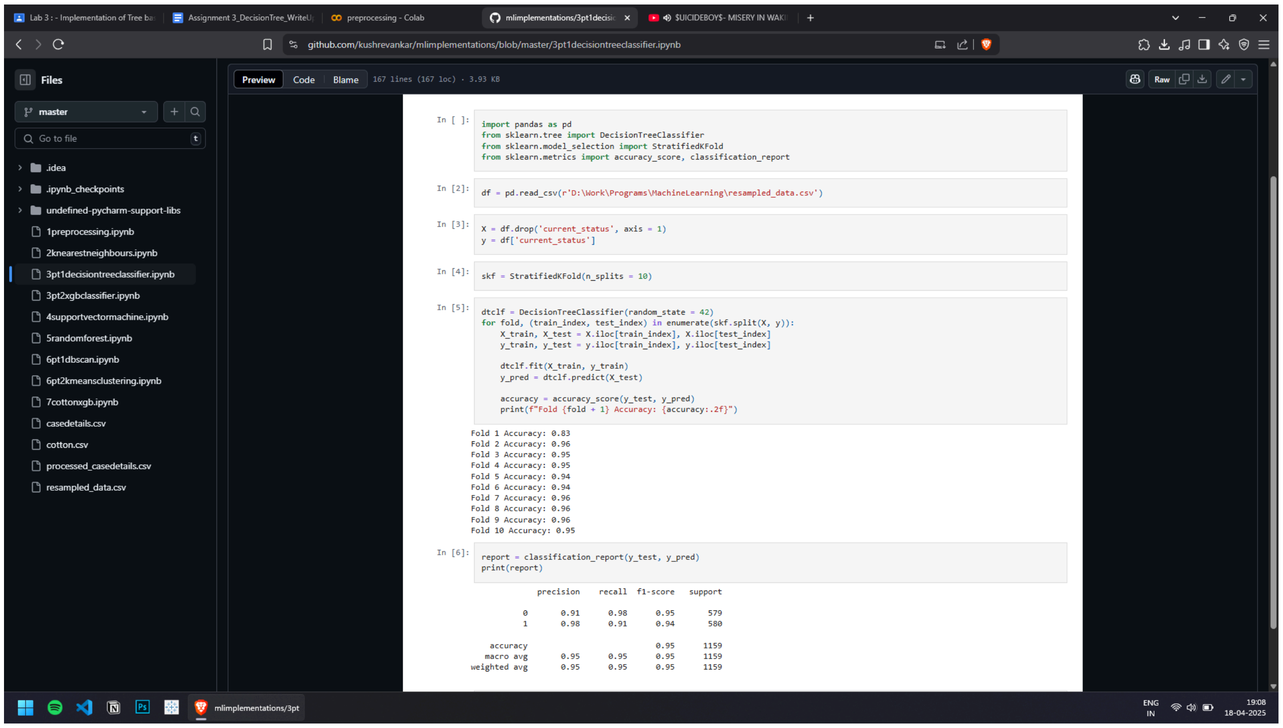
**Conclusion:**
In this lab, we implemented a decision tree classifier to predict whether a person is likely to purchase a product or not based on their age, income. performance analysis of decision Tree classifier was done.

**FAQs:**

1) What is the Decision Tree classifier?
2) What are some advantages of decision trees?
3) How does a decision tree work?
4) How do you prevent overfitting in a decision tree?
5) What is pruning in decision trees?

Code & Output:

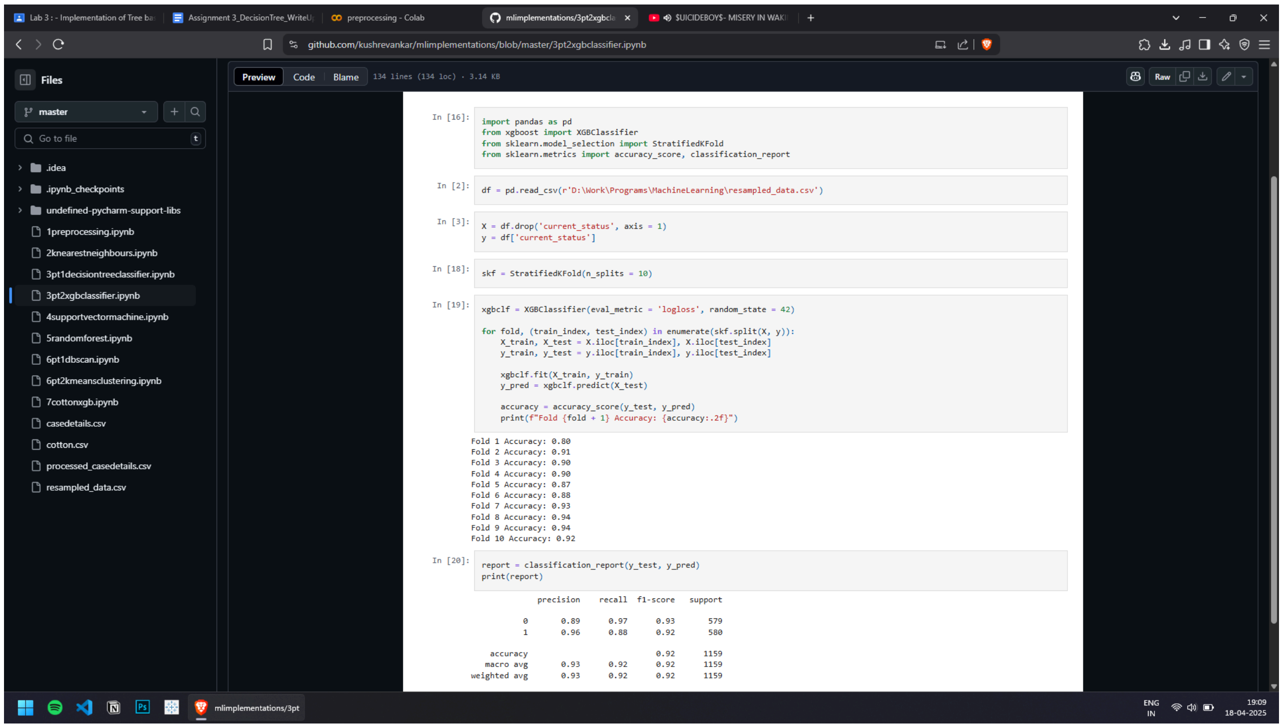
Decision Tree Classifier



github.com/kushrevankar/mlimplementations/blob/master/3pt1decisiontreeclassifier.ipynb

```
In [ ]:  import pandas as pd
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.model_selection import StratifiedKFold
         from sklearn.metrics import accuracy_score, classification_report

In [2]:  df = pd.read_csv(r'D:\Work\Programs\MachineLearning\resampled_data.csv')

In [3]:  X = df.drop('current_status', axis = 1)
         y = df['current_status']

In [4]:  skf = StratifiedKFold(n_splits = 10)

In [5]:  dtclf = DecisionTreeClassifier(random_state = 42)
         for fold, (train_index, test_index) in enumerate(skf.split(X, y)):
             X_train, X_test = X.iloc[train_index], X.iloc[test_index]
             y_train, y_test = y.iloc[train_index], y.iloc[test_index]

             dtclf.fit(X_train, y_train)
             y_pred = dtclf.predict(X_test)

             accuracy = accuracy_score(y_test, y_pred)
             print(f"Fold {fold + 1} Accuracy: {accuracy:.2f}")

        Fold 1 Accuracy: 0.83
        Fold 2 Accuracy: 0.96
        Fold 3 Accuracy: 0.95
        Fold 4 Accuracy: 0.95
        Fold 5 Accuracy: 0.94
        Fold 6 Accuracy: 0.94
        Fold 7 Accuracy: 0.96
        Fold 8 Accuracy: 0.96
        Fold 9 Accuracy: 0.96
        Fold 10 Accuracy: 0.95

In [6]:  report = classification_report(y_test, y_pred)
         print(report)

                      precision    recall  f1-score   support

                   0       0.91      0.98      0.95       579
                   1       0.98      0.91      0.94       580

            accuracy                           0.95      1159
           macro avg       0.95      0.95      0.95      1159
        weighted avg       0.95      0.95      0.95      1159
```

## XGBoost Classifier



```
In [16]:
import pandas as pd
from xgboost import XGBClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report
```

```
In [2]:
df = pd.read_csv(r'D:\Work\Programs\MachineLearning\resampled_data.csv')
```

```
In [3]:
X = df.drop('current_status', axis = 1)
y = df['current_status']
```

```
In [18]:
skf = StratifiedKFold(n_splits = 10)
```

```
In [19]:
xgbclf = XGBClassifier(eval_metric = 'logloss', random_state = 42)

for fold, (train_index, test_index) in enumerate(skf.split(X, y)):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    xgbclf.fit(X_train, y_train)
    y_pred = xgbclf.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"Fold {fold + 1} Accuracy: {accuracy:.2f}")
```

```
Fold 1 Accuracy: 0.80
Fold 2 Accuracy: 0.91
Fold 3 Accuracy: 0.90
Fold 4 Accuracy: 0.90
Fold 5 Accuracy: 0.87
Fold 6 Accuracy: 0.88
Fold 7 Accuracy: 0.93
Fold 8 Accuracy: 0.94
Fold 9 Accuracy: 0.94
Fold 10 Accuracy: 0.92
```

```
In [20]:
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       579
           1       0.96      0.88      0.92       580

    accuracy                           0.92      1159
   macro avg       0.93      0.92      0.92      1159
weighted avg       0.93      0.92      0.92      1159
```