# Assignment 3

**Title : Assignment on Classification technique**

**Aim**: Every year many students give the GRE exam to get admission in foreign Universities. The data set contains GRE Scores (out of 340), TOEFL Scores (out of 120), University Rating (out of 5), Statement of Purpose strength (out of 5), Letter of Recommendation strength (out of 5), Undergraduate GPA (out of 10), Research Experience (0=no, 1=yes), Admitted (0=no, 1=yes). Admitted is the target variable.

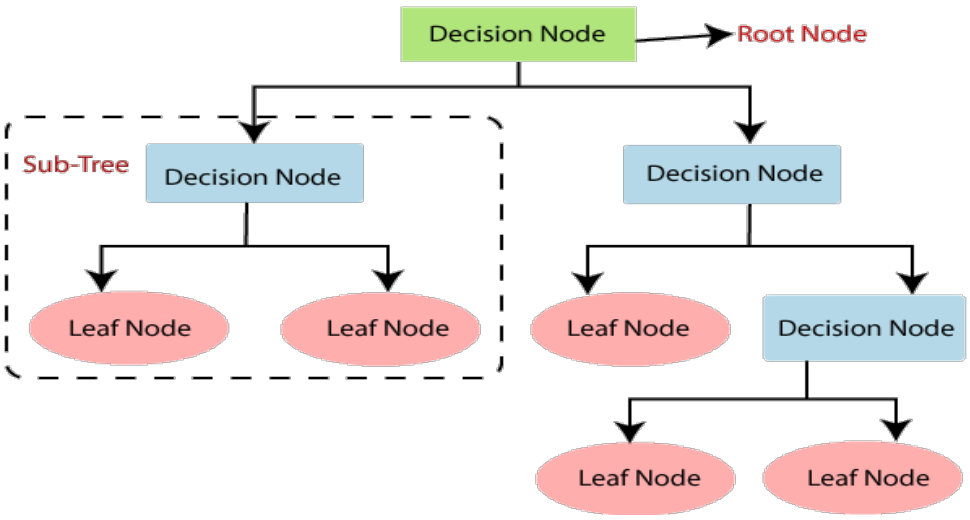
Data Set Available on kaggle (The last column of the dataset needs to be changed to 0 or 1)
Data Set : https://www.kaggle.com/mohansacharya/graduate-admissions

The counsellor of the firm is supposed check whether the student will get an admission or not based on his/her GRE score and Academic Score. So to help the counselor to take appropriate decisions build a **machine learning model classifier using Decision tree** to predict whether a student will get admission or not.

A. Apply Data pre-processing (Label Encoding, Data Transformation….) techniques if necessary.
B. Perform data-preparation (Train-Test Split)
C. Apply Machine Learning Algorithm
D. Evaluate Model.

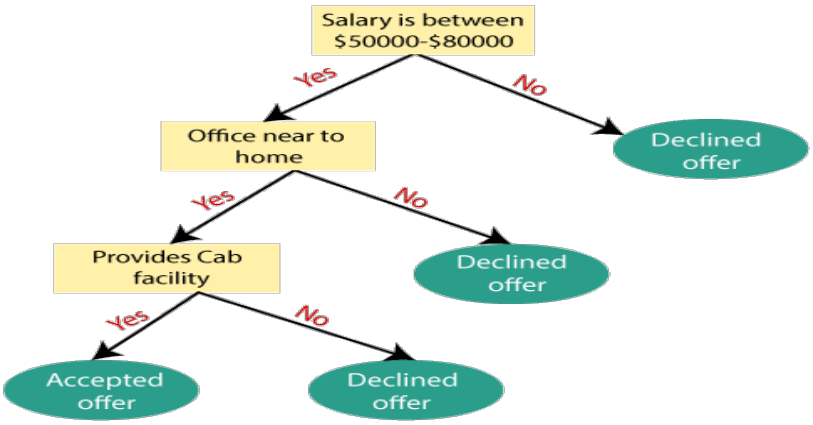
## I] Theory

- 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.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- 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.
- **How does the Decision Tree algorithm Work?**
- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.
- For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:
- **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:




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:

1. 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.**

### Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

### Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

## Python Implementation of Decision Tree

Steps are given below:

1. **Data Pre-processing step**

**Data Normalization**

- Normalization is a rescaling of the data from the original range so that all values are within the new range of 0 and 1.
  A value is normalized as follows:
  y = (x – min) / (max – min)
  Where the minimum and maximum values pertain to the value x being normalized.
  For normalization use the scikit-learn object **MinMaxScaler.**
  Good practice usage with the MinMaxScaler and other scaling techniques is as follows:
  - **Fit the scaler using available training data**. For normalization, this means the training data will be used to estimate the minimum and maximum observable values. This is done by calling the *fit()* function.
  - **Apply the scale to training data**. This means you can use the normalized data to train your model. This is done by calling the *transform()* function.

- **Apply the scale to data going forward**. This means you can prepare new data in the future on which you want to make predictions.

The default scale for the *MinMaxScaler* is to rescale variables into the range [0,1], although a preferred scale can be specified via the "*feature_range*" argument and specify a tuple, including the min and the max for all variables.

```
# normalization
from sklearn.preprocessing import MinMaxScaler

scalerX = MinMaxScaler(feature_range=(0, 1))
X_train[X_train.columns] = scalerX.fit_transform(X_train[X_train.columns])
X_test[X_test.columns] = scalerX.transform(X_test[X_test.columns])
```

2. **Fitting a Decision-Tree algorithm to the Training set**

Import the **DecisionTreeClassifier** class from **sklearn.tree** library. Below is the code for it:

```
#Fitting Decision Tree classifier to the training set
From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

In the above code, we have created a classifier object, in which we have passed two main parameters;
**"criterion='entropy':** Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.
**random_state=0":** For generating the random states

3. **Predicting the test result**

create a new prediction vector **y_pred.** Below is the code for it:

```
y_pred= classifier.predict(x_test)
```

4. **Test accuracy of the result(Creation of Confusion matrix)**

Below is the code for it:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

**5.Visualizing the test set result and performance metrics**

Below is the code for it:

```
# cm visualization
import seaborn as sns
import matplotlib.pyplot as plt
```

```
f, ax = plt.subplots(figsize=(5, 5))
sns.heatmap(cm_dtc, annot=True, linewidths=0.5,
linecolor="red", fmt=".0f", ax=ax)
plt.title("Test for Test Dataset:decision tree")
plt.xlabel("predicted y values")
plt.ylabel("real y values")
plt.show()

# %% [code]
from sklearn.metrics import precision_score, recall_score

print("precision_score: ", precision_score(y_test_01,
y_pred_dtc))
print("recall_score: ", recall_score(y_test_01, y_pred_dtc))

from sklearn.metrics import f1_score

print("f1_score: ", f1_score(y_test_01, y_pred_dtc))
```

## IV] Sample Code with comments

```
# Assignment 3: Classification using Decision Tree
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# %% [markdown]
# ## Prediction Models : Classification Algorithm (Supervised Machine
Learning)
#
# 4. Decision Tree Classification

# %% [markdown]
# ### Preparing Data for Classification

# %% [markdown]
# If a candidate's Chance of Admit is greater than 80%, the candidate
will receive the 1 label.
#
# If a candidate's Chance of Admit is less than or equal to 80%, the
candidate will receive the 0 label.

# %% [code]
# reading the dataset
df = pd.read_csv("Admission_Predict.csv")
print("shape of dataset",df.shape)

# %% [code]
# it may be needed in the future.
serialNo = df["Serial No."].values
df.drop(["Serial No."], axis=1, inplace=True)

df = df.rename(columns={'Chance of Admit ': 'Chance of Admit'})

# %% [code]
```

```
X = df.drop(["Chance of Admit"], axis=1)
y = df["Chance of Admit"].values

# %% [code]
# separating train (80%) and test (%20) sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=101)

# %% [code]
# normalization
from sklearn.preprocessing import MinMaxScaler

scalerX = MinMaxScaler(feature_range=(0, 1))
X_train[X_train.columns] =
scalerX.fit_transform(X_train[X_train.columns])
X_test[X_test.columns] = scalerX.transform(X_test[X_test.columns])

# %% [code]
y_train_01 = [1 if each > 0.8 else 0 for each in y_train]
y_test_01 = [1 if each > 0.8 else 0 for each in y_test]

# list to array
y_train_01 = np.array(y_train_01)
y_test_01 = np.array(y_test_01)

# %% [markdown]
# ### 4. Decision Tree Classification

# %% [code]
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train_01)
y_pred_dtc = dtc.predict(X_test)
print("score: ", dtc.score(X_test, y_test_01))

# %% [code]
# confusion matrix
from sklearn.metrics import confusion_matrix

cm_dtc = confusion_matrix(y_test_01, y_pred_dtc)
print("confusion matrix for test data:decision tree\n",cm_dtc)
# print("y_test_01 == 1 :" + str(len(y_test_01[y_test_01==1]))) # 29
cm_dtc

# %% [code]
# cm visualization
import seaborn as sns
import matplotlib.pyplot as plt

f, ax = plt.subplots(figsize=(5, 5))
sns.heatmap(cm_dtc, annot=True, linewidths=0.5, linecolor="red",
fmt=".0f", ax=ax)
plt.title("Test for Test Dataset:decision tree")
plt.xlabel("predicted y values")
plt.ylabel("real y values")
```

```
plt.show()

# %% [code]
from sklearn.metrics import precision_score, recall_score

print("precision_score: ", precision_score(y_test_01, y_pred_dtc))
print("recall_score: ", recall_score(y_test_01, y_pred_dtc))

from sklearn.metrics import f1_score

print("f1_score: ", f1_score(y_test_01, y_pred_dtc))

# %% [markdown]
# Test for Train Dataset:

# %% [code]
cm_dtc_train = confusion_matrix(y_train_01, dtc.predict(X_train))
print("confusion matrix for test data:decision tree\n",cm_dtc_train)
f, ax = plt.subplots(figsize=(5, 5))
sns.heatmap(cm_dtc_train, annot=True, linewidths=0.5, linecolor="red",
fmt=".0f", ax=ax)
plt.xlabel("predicted y values")
plt.ylabel("real y values")
plt.title("Test for Train Dataset:decision tree")
plt.show()
```

## V] Output of Code:

**Note: Run the code and attach your output of the code here.**