Experiment 6

Aim: Classification modeling
a. Choose a classifier for a classification problem.
b. Evaluate the performance of the classifier.

To do:-
Perform Classification using the below 4 classifiers

a. **K-Nearest Neighbors (KNN)**
b. **Naive Bayes**
c. **Support Vector Machines (SVMs)**
d. **Decision Tree**

About Dataset:
The dataset contains details such as the Airline Company, Origin and destination of flight, Departure and arrival time of the flight. The task is to predict whether a given flight will be delayed. As the number of flights is increasing everyday and people are starting to travel more and more using flights, the delay in flights is inevitable. So predicting the delay in flights is a necessity.

Theory:

What is classification ?

Classification is a supervised form of learning, where you teach the computer to do something with data that's already labeled by humans. This training set includes a fixed amount of labels or categories for the computer to learn from. By spotting patterns in the training data, the machine can classify new data to pre-determine categories, so that's classification.

What are Classifiers(Classification Algorithms)?

Based on training data, the Classification algorithm is a Supervised Learning technique used to categorize new observations. In classification, a program uses the dataset or observations provided to learn how to categorize new observations into various classes or groups. For instance, 0 or 1, red or blue, yes or no, spam or not spam, etc. Targets, labels, or categories can all be used to describe classes. The Classification algorithm uses labeled input data because it is a supervised learning technique and comprises input and output information. A discrete output function (y) is transferred to an input variable in the classification process (x). In simple words, classification is a type of pattern recognition in which classification algorithms are performed on training data to discover the same pattern in new data sets.

1. KNN

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another. For classification problems, a class label is assigned on the basis of a majority vote—i.e. the label that is most frequently represented around a given data point is used. While this is technically considered “plurality voting”, the term, “majority vote” is more commonly used in literature. The distinction between these terminologies is that “majority voting” technically requires a majority of greater than 50%, which primarily works when there are only two categories.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know whether it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

2. Naive Bayes

The Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. The Naive Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naive Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Bayes' Theorem:

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability. The formula for Bayes' theorem is given as:

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

Where,

- P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.
- P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

Types of Naive Bayes Classifiers

A. **Gaussian Naive Bayes**

This type of Naive Bayes is used when variables are continuous in nature. It assumes that all the variables have a normal distribution. So if you have some variables which do not have this property, you might want to transform them to the features having distribution normal.

B. **Multinomial Naive Bayes**

This is used when the features represent the frequency. Suppose you have a text document and you extract all the unique words and create multiple features where each feature represents the count of the word in the document. In such a case, we have a frequency as a feature. In such a scenario, we use multinomial Naive Bayes. It ignores the non-occurrence of the features. So, if you have frequency 0 then the probability of occurrence of that feature will be 0 hence multinomial naive Bayes ignores that feature. It is known to work well with text classification problems.

C. **Bernoulli Naive Bayes**

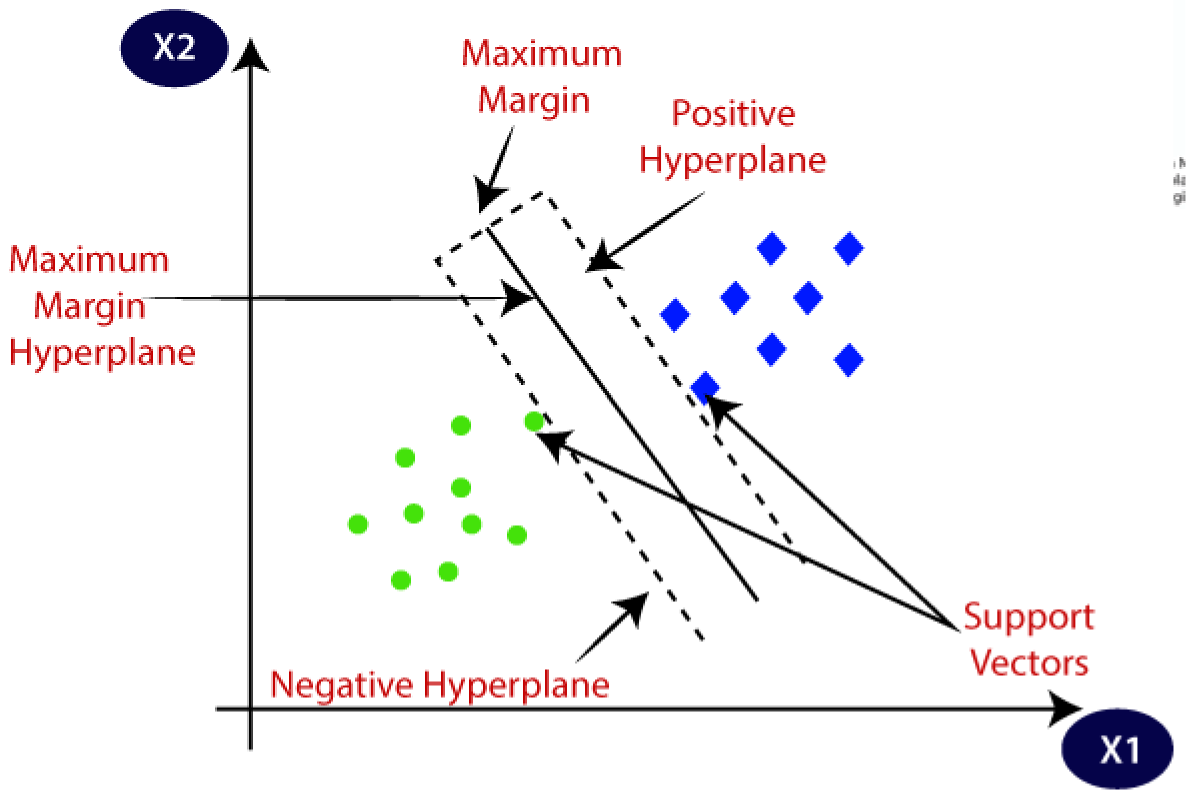
This is used when features are binary. So, instead of using the frequency of the word, if you have discrete features in 1s and 0s that represent the presence or absence of a feature. In that case, the features will be binary and we will use Bernoulli Naive Bayes. Also, this method will penalize the non-occurrence of a feature, unlike multinomial Naive Bayes.

D. **Complement Naïve Bayes Classifier**

Complement Naive Bayes is somewhat a modification of the standard Multinomial Naive Bayes algorithm. Multinomial Naive Bayes is not able to do very well with unstable data. Imbalanced data sets are instances where the number of instances belonging to a particular class is greater than the number of instances belonging to different classes. This implies the spread of the examples is not even. This kind of data can be difficult to analyze as models can easily overfit this data to benefit a class with a larger instance.

3. SVM

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:
Below diagram explains the SVM:

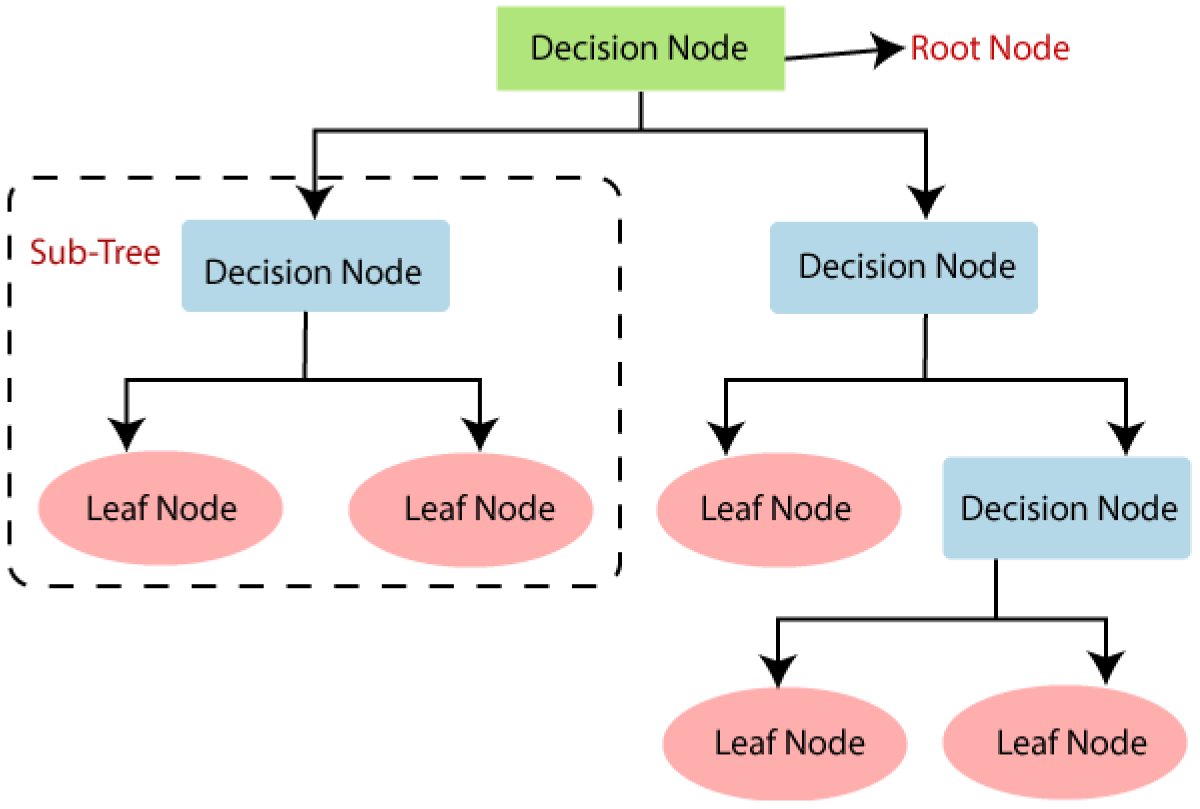


4. Decision Tree

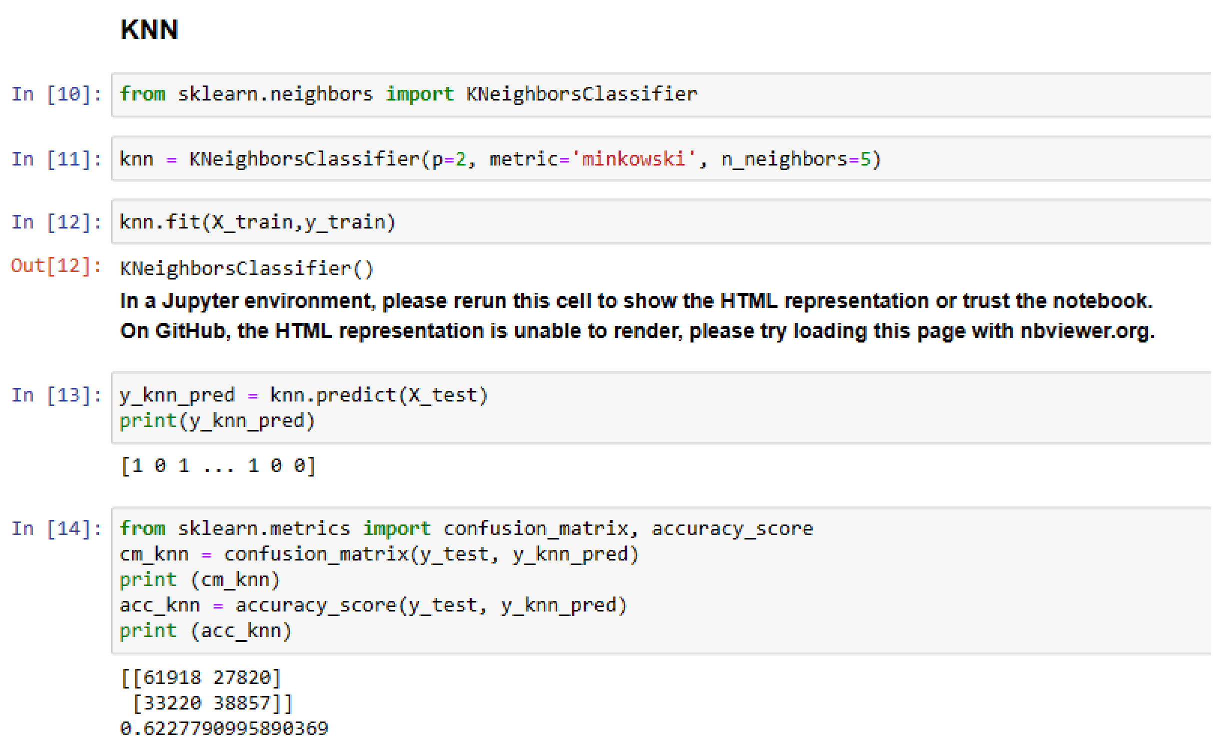
Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision 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. 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 splits the tree into subtrees.

Below diagram explains the general structure of a decision tree:



Output-



KNN

```
In [10]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [11]: knn = KNeighborsClassifier(p=2, metric='minkowski', n_neighbors=5)
```

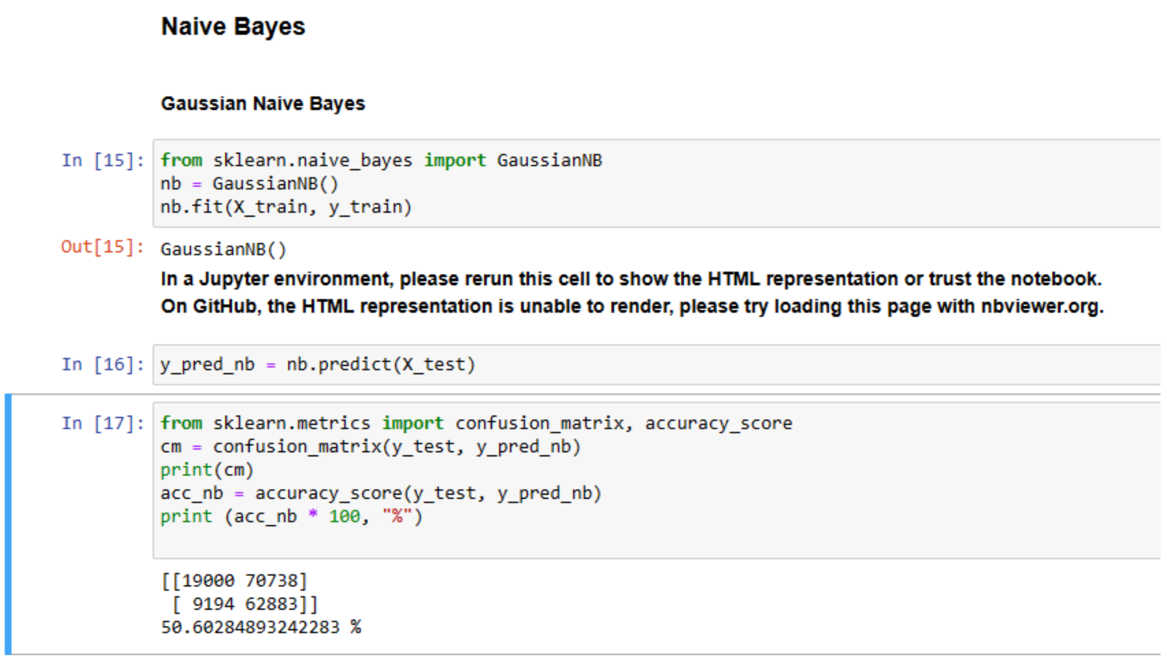
```
In [12]: knn.fit(X_train,y_train)
```

```
Out[12]: KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [13]: y_knn_pred = knn.predict(X_test)
         print(y_knn_pred)

[1 0 1 ... 1 0 0]
```

```
In [14]: from sklearn.metrics import confusion_matrix, accuracy_score
         cm_knn = confusion_matrix(y_test, y_knn_pred)
         print (cm_knn)
         acc_knn = accuracy_score(y_test, y_knn_pred)
         print (acc_knn)

[[61918 27820]
 [33220 38857]]
0.6227790995890369
```



Naive Bayes

Gaussian Naive Bayes

```
In [15]: from sklearn.naive_bayes import GaussianNB
         nb = GaussianNB()
         nb.fit(X_train, y_train)
```

```
Out[15]: GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [16]: y_pred_nb = nb.predict(X_test)
```

```
In [17]: from sklearn.metrics import confusion_matrix, accuracy_score
         cm = confusion_matrix(y_test, y_pred_nb)
         print(cm)
         acc_nb = accuracy_score(y_test, y_pred_nb)
         print (acc_nb * 100, "%")

[[19000 70738]
 [ 9194 62883]]
50.60284893242283 %
```

Multinomial Naive Bayes

```
In [18]: from sklearn.naive_bayes import MultinomialNB
         mb = MultinomialNB()
         mb.fit(X_train, y_train)

Out[18]: MultinomialNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [19]: y_pred_mb = mb.predict(X_test)

In [20]: from sklearn.metrics import confusion_matrix, accuracy_score
         cm = confusion_matrix(y_test, y_pred_mb)
         print(cm)
         acc_mb = accuracy_score(y_test, y_pred_mb)
         print (acc_mb * 100, "%")

         [[70639 19099]
          [40859 31218]]
         62.94657479220097 %
```

Bernoulli Naive Bayes

```
In [24]: from sklearn.naive_bayes import BernoulliNB
         bb = BernoulliNB()
         bb.fit(X_train, y_train)

Out[24]: BernoulliNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [25]: y_pred_bb = bb.predict(X_test)

In [26]: from sklearn.metrics import confusion_matrix, accuracy_score
         cm = confusion_matrix(y_test, y_pred_bb)
         print(cm)
         acc_bb = accuracy_score(y_test, y_pred_bb)
         print (acc_bb * 100, "%")

         [[71437 18301]
          [41770 30307]]
         62.8767419584093 %
```

Complement Naive Bayes

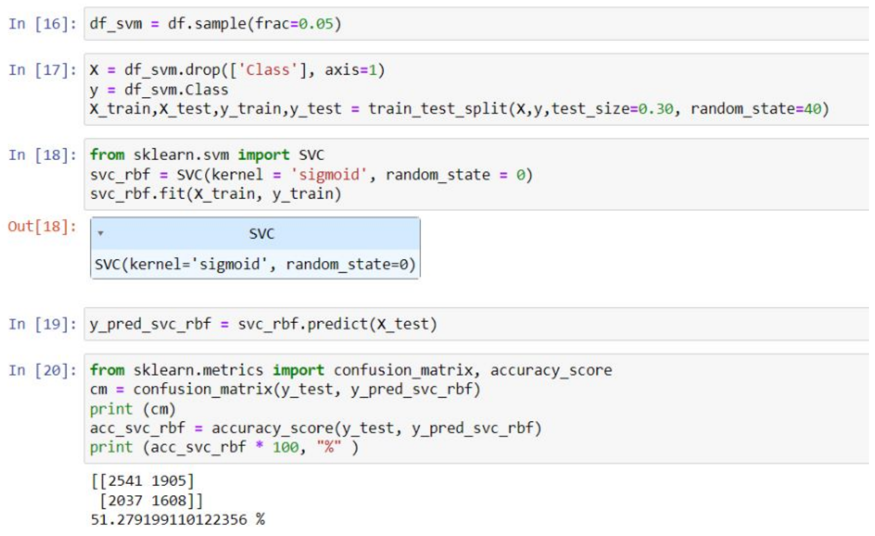
```
In [21]: from sklearn.naive_bayes import ComplementNB
         cb = ComplementNB()
         cb.fit(X_train, y_train)

Out[21]: ComplementNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

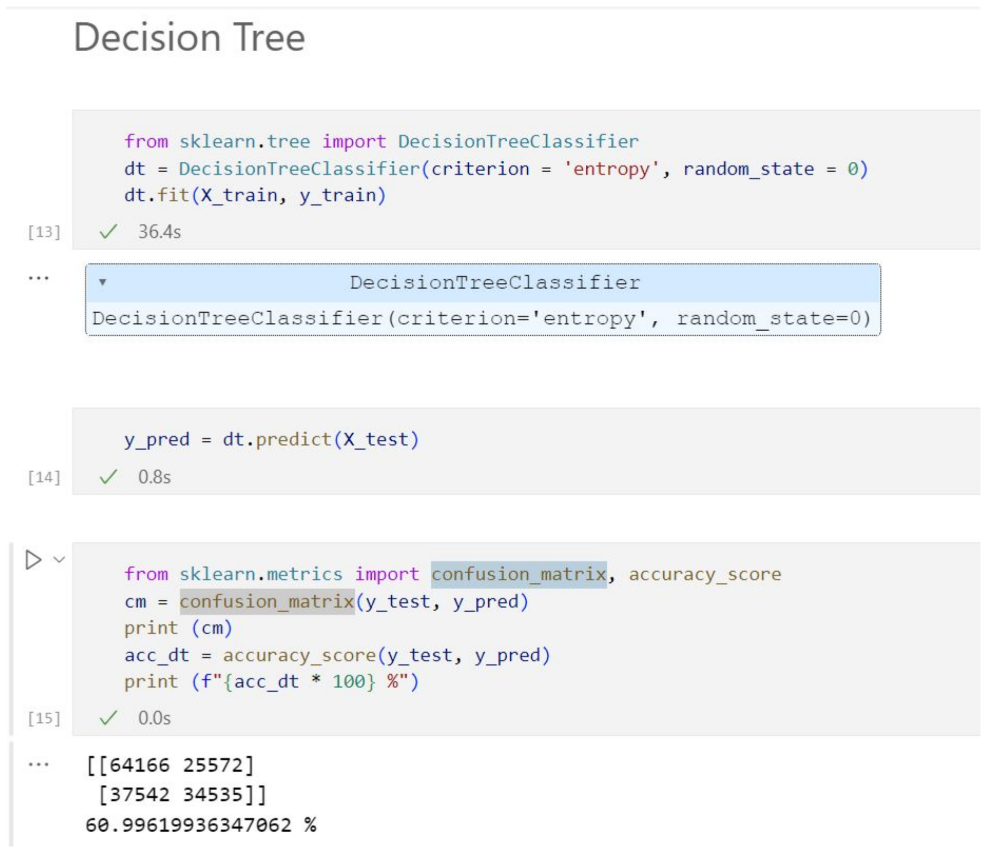
```
In [22]: y_pred_cb = cb.predict(X_test)
```

```
In [23]: from sklearn.metrics import confusion_matrix, accuracy_score
         cm = confusion_matrix(y_test, y_pred_cb)
         print(cm)
         acc_cb = accuracy_score(y_test, y_pred_cb)
         print (acc_cb * 100, "%")

[[63059 26679]
 [34973 37104]]
61.89970027500541 %
```

SVM



```
In [16]: df_svm = df.sample(frac=0.05)

In [17]: X = df_svm.drop(['Class'], axis=1)
         y = df_svm.Class
         X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30, random_state=40)

In [18]: from sklearn.svm import SVC
         svc_rbf = SVC(kernel = 'sigmoid', random_state = 0)
         svc_rbf.fit(X_train, y_train)

Out[18]: SVC
         SVC(kernel='sigmoid', random_state=0)

In [19]: y_pred_svc_rbf = svc_rbf.predict(X_test)

In [20]: from sklearn.metrics import confusion_matrix, accuracy_score
         cm = confusion_matrix(y_test, y_pred_svc_rbf)
         print (cm)
         acc_svc_rbf = accuracy_score(y_test, y_pred_svc_rbf)
         print (acc_svc_rbf * 100, "%" )

[[2541 1905]
 [2037 1608]]
51.279199110122356 %
```



Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
dt.fit(X_train, y_train)
```
[13] ✓ 36.4s

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
y_pred = dt.predict(X_test)
```
[14] ✓ 0.8s

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print (cm)
acc_dt = accuracy_score(y_test, y_pred)
print (f"{acc_dt * 100} %")
```
[15] ✓ 0.0s

```
[[64166 25572]
 [37542 34535]]
60.99619936347062 %
```

Conclusion-

In this experiment we have successfully applied various classification algorithms. In the process we came to know about the accuracy of the model and the prediction it gave based on various inputs.