Introduction: -

Social media platforms allow users to publish and exchange content in text, image and many other formats. Users of such platforms establish different forms of social communities that affect societies in a much broader sense: from consumption patterns to presidential elections. These platforms are designed to retain their audience with algorithms that will sort and filter the content displayed to each user based on their preferences and attention patterns. These algorithms are so effective in retaining users attention that a growing number of them are now suffering with addiction to their social media. With users spending an ever-increasing amount of time on their favorite platforms, social media have amassed huge databases on user profiling and segmentation and became, arguably, some of the most effective mass communication tools. They monetize their databases serving targeted advertising and charging companies that consume their Application Programming Interfaces (APIs) to interact with users on their platforms. This business model is threatened by the misuse of the platforms with the creation of accounts that do not represent the interaction of a real person or a real organization with the network. These fake accounts are usually set up as means for other malicious activities. Fake accounts on social media have become the main channel for automated software agents known as bots to interact with the networks. The malicious use of bots has already been applied, for example, to manipulate opinions on extremely relevant issues, such as presidential elections in France and the United States. Identifying and clearly flagging bot accounts may help users to assert better judgment on the content spread by these accounts and lessen their negative effects.

Existing system: -

CNN is used in existing model of this problem statement.

Drawbacks: -

- Less amount of accuracy score
- Small level data-set

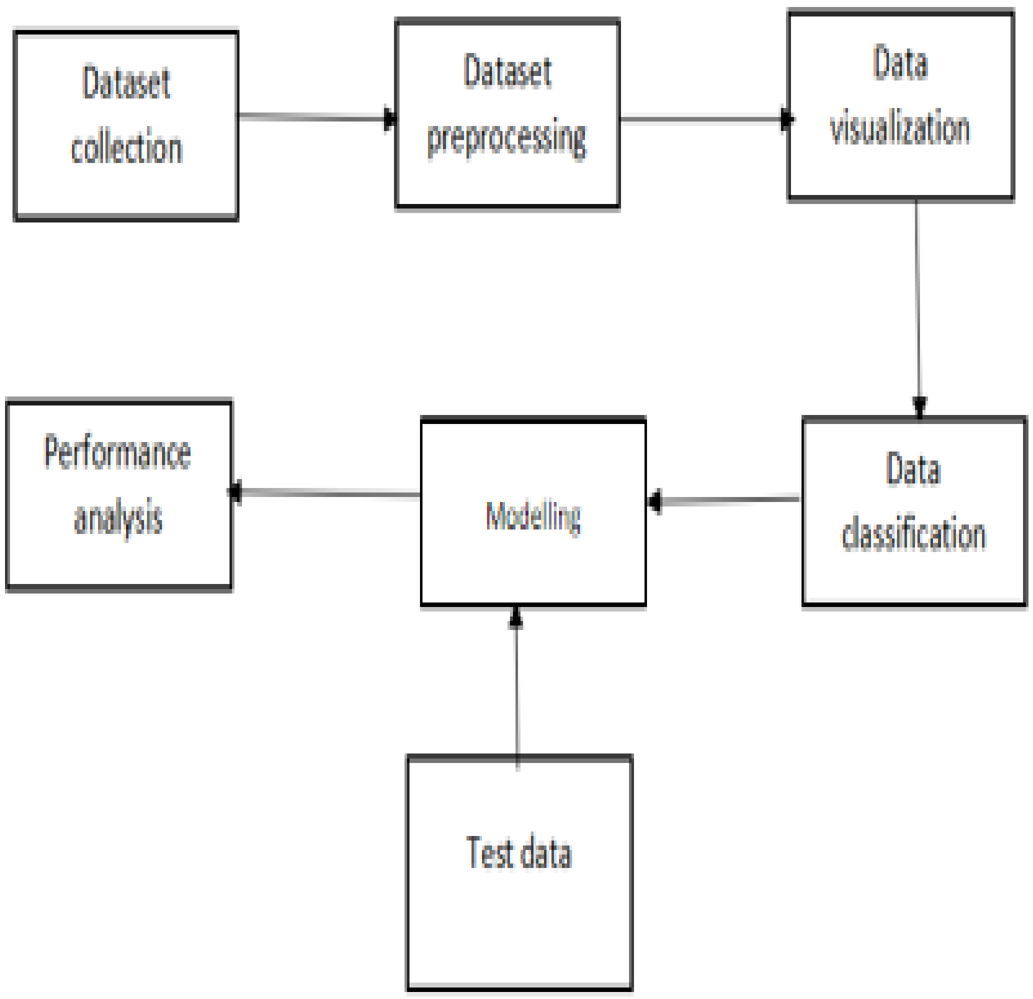
Proposed system: -

In this problem statement we have used supervised machine algorithms like Decision Tree, Random Forest and Multinominal Naive Bayes.

Advantages: -

- Accuracy level is good
- Time consumption is less
- Comparison of different algorithms can be observed

System Architecture: -



Algorithms

Decision Tree Classifier

Decision tree learning is one of the predictive modeling approaches that use a decision tree (as a predictive model) to go from observations about an item i.e., attribute (represented in the branches) to conclusions about the item's target value i.e., bot or not (represented in the leaves).

Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. This algorithm splits a data sample into two or more homogeneous sets based on the most significant differentiator in input variables to make a prediction. With each split, a part of a tree is being generated. As a result, a tree with decision nodes and leaf nodes (which are decisions or classifications) is developed. A tree starts from a root node – the best predictor.

Naïve Bayes Algorithm

The Naïve Bayes algorithm is used in supervised learning tasks. It assumes that the predictive attributes are conditionally independent.

The Bayesian method is a classification method that makes use of the Bayesian theorem. The theorem updates the prior knowledge of an event with the independent probability of each feature that can affect the event.

Random Forest Classifier

The random forest model creates many variations of trees. The best outcome will be used to predict identity deception. This model works well for bot detection, as rules are easily represented in tree format. An example would be where accounts that have an image or name are considered human, whereas the rest are denoted as bots. Each of these outcomes represents a different section in the tree.

Bag of words model

The bag of words model detects if a given account is a bot account or not based on two conditions.

1.It calculates the engagement ratio for every new tuple and if it is very low it classifies such a tuple as a bot account.

2. We create a bag of words that contain bad and obscene words, if the account description, status, id, bio contain words from the bag then such an account is classified as a bot account.

3. We also calculate the rate at which posts are updated on the platform. If the rate of creating posts is inhumanly high then such an account is considered a bot account

System Requirements: -

Hardware:

1.OS – Windows 7, 8 and 10 (32 and 64 bit)
2.RAM – 4GB

Software:

1.Python / Anaconda Navigator
2.In Python language
3. Jupyter Notebook

Literature survey: -

There is an increase in the use of deep learning and machine learning in the industry. The use of artificial intelligence techniques and machine learning can also be used to detect the bots in social media to avoid harm to individuals by them. In, the authors suggested a novel method of extraction of features to identify bot accounts on social media sites. Their method involves the development of ego networks for each account, followed by the extraction of features. Results show that machine learning algorithms give better effectiveness when high-order topological characteristics are provided. In, the authors defined a set of features derived from Twitter profile metadata and used the same to detect Twitter bots. They used the Deep Forest Algorithm for this purpose. Lin and Haung proposed a system to detect spammers on Twitter. They applied the algorithm on the shortlisted features which include the rate of Universe Resource Locator (URL) and rate of interaction. Kudugunta and Ferrara developed a method for Twitter bot detection using deep learning neural networks based on long short-term memory (LSTM). They used both contextual data and metadata for the purpose. In, the authors focused on identifying the Twitter bot accounts using the behavior data of the accounts. For this, they focused on the data gathered between February 2014 to December 2015 and applied machine learning strategies to the data for efficient detection. In, The authors stated that Twitter bot identification is viable via a supervised classification process. They also demonstrated that binary supervised classifiers are unable to correctly detect new or distinct bot accounts from those used to train the classifier. Yang and Menczer developed a strict validation process that would improve the accuracy of the system on unseen data objects. They proposed a bot detection model that scales the entire public Twitter stream into real-time computing and generates accounts far outside of the training data and also found that carefully selecting data objects for training could enhance the model performance. In, A framework for semi-controlled spam detection called S3D was developed by the authors. Four lightweight detectors by S3D are used to detect spam tweets in real-time and update the batch models periodically. They also found that the system can recognize new trends of spamming with confident labeling of clusters and tweets.

Modules: -

DATA COLLECTION

Data collection is a process in which information is gathered from many sources which is later used to develop the machine learning models. The data should be stored in a way that makes sense for problem. In this step the data set is converted into the understandable format which can be fed into machine learning models.
Data used in this paper is a set of cervical cancer data with 15 features. This step is concerned with selecting the subset of all available data that you will be working with. ML problems start with data preferably, lots of data (examples or observations) for which you already know the target answer. Data for which you already know the target answer is called *labelled data*.

DATA PRE-PROCESSING

Organize your selected data by formatting, cleaning and sampling from it.

Three common data pre-processing steps are:

Formatting: The data you have selected may not be in a format that is suitable for you to work with. The data may be in a relational database and you would like it in a flat file, or the data may be in a proprietary file format and you would like it in a relational database or a text file.

Cleaning: Cleaning data is the removal or fixing of missing data. There may be data instances that are incomplete and do not carry the data you believe you need to address the problem. These instances may need to be removed. Additionally, there may be sensitive information in some of the attributes and these attributes may need to be anonymized or removed from the data entirely.

Sampling: There may be far more selected data available than you need to work with. More data can result in much longer running times for algorithms and larger computational and memory requirements. You can take a smaller representative sample of the selected data that may be much faster for exploring and prototyping solutions before considering the whole dataset.

FEATURE EXTRATION

Next thing is to do Feature extraction is an attribute reduction process. Unlike feature selection, which ranks the existing attributes according to their predictive significance, feature extraction actually transforms the attributes. The transformed attributes, or features, are linear combinations of the original attributes. Finally, our models are trained using Classifier algorithm. We use classify module on Natural Language Toolkit library on Python. We use the labelled dataset gathered. The rest of our labelled data will be used to evaluate the models. Some machine learning algorithms were used to classify pre-processed data. The chosen classifiers were Random Forest. These algorithms are very popular in text classification tasks.

EVALUATION MODEL

Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future. Evaluating model performance with the data used for training is not acceptable in data science because it can easily generate overoptimistic and over fitted models. There are two methods of evaluating models in data science, Hold-Out and Cross-Validation. To avoid over fitting, both methods use a test set (not seen by the model) to evaluate model performance.

Performance of each classification model is estimated base on its averaged. The result will be in the visualized form. Representation of classified data in the form of graphs.

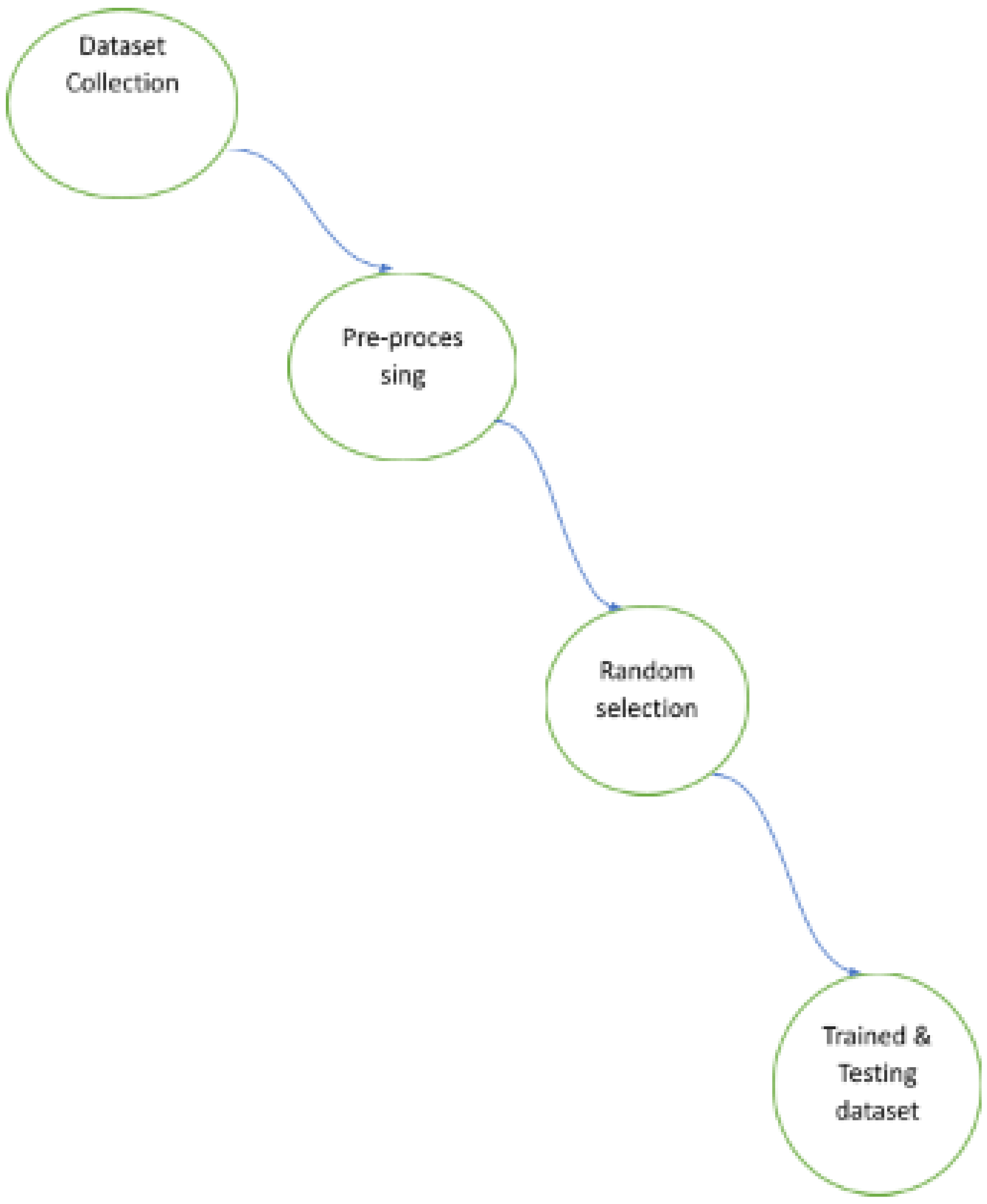
Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

Feasibility study: -

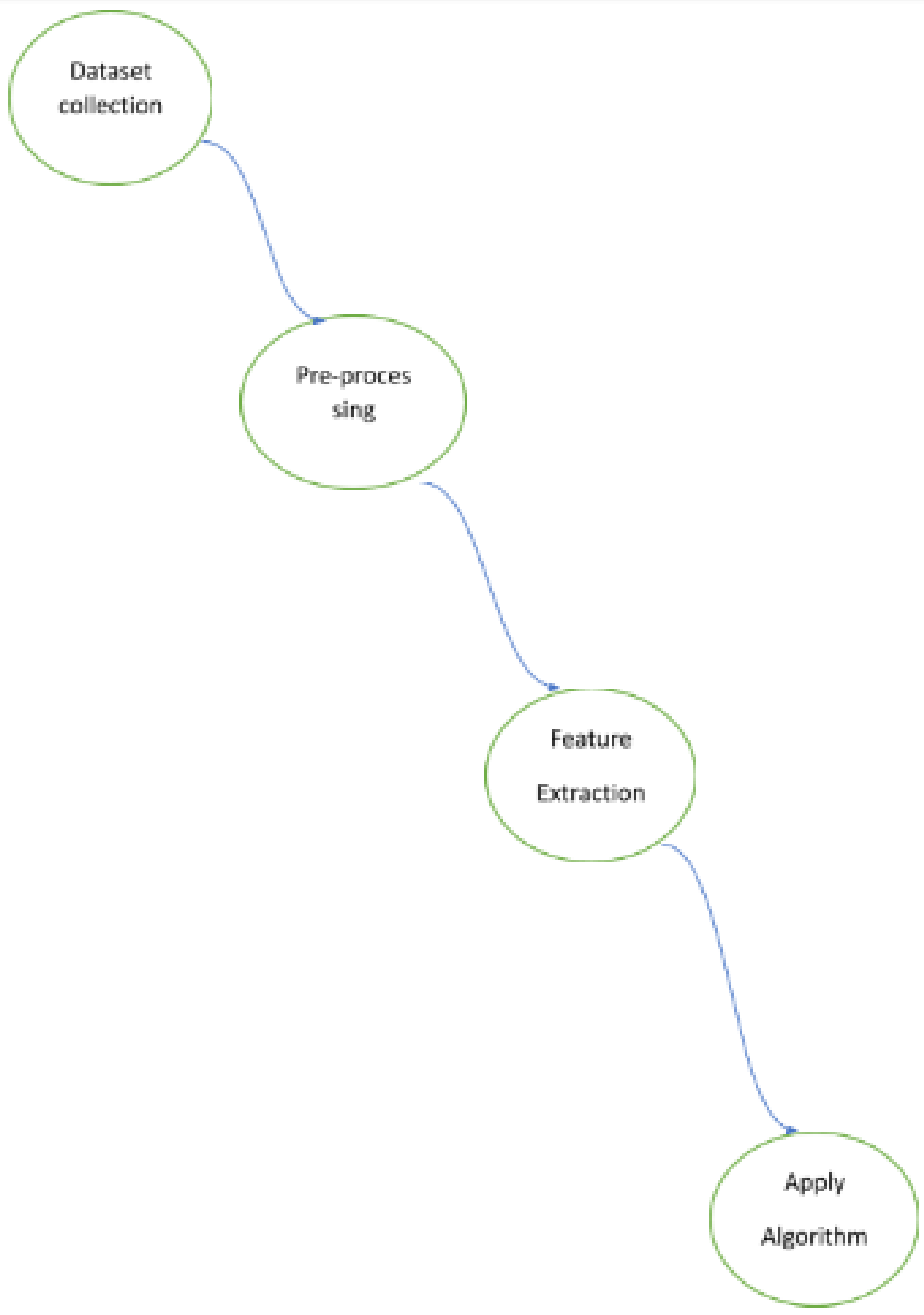
1. First, we take dataset.
2. Filter dataset according to requirements and create a new dataset which has attribute according to analysis to be done
3. Perform Pre-Processing on the dataset
4. Split the data into training and testing
5. Train the model with training data then analyze testing dataset over classification algorithm
6. Finally you will get results as accuracy metrics

Data flow diagrams: -

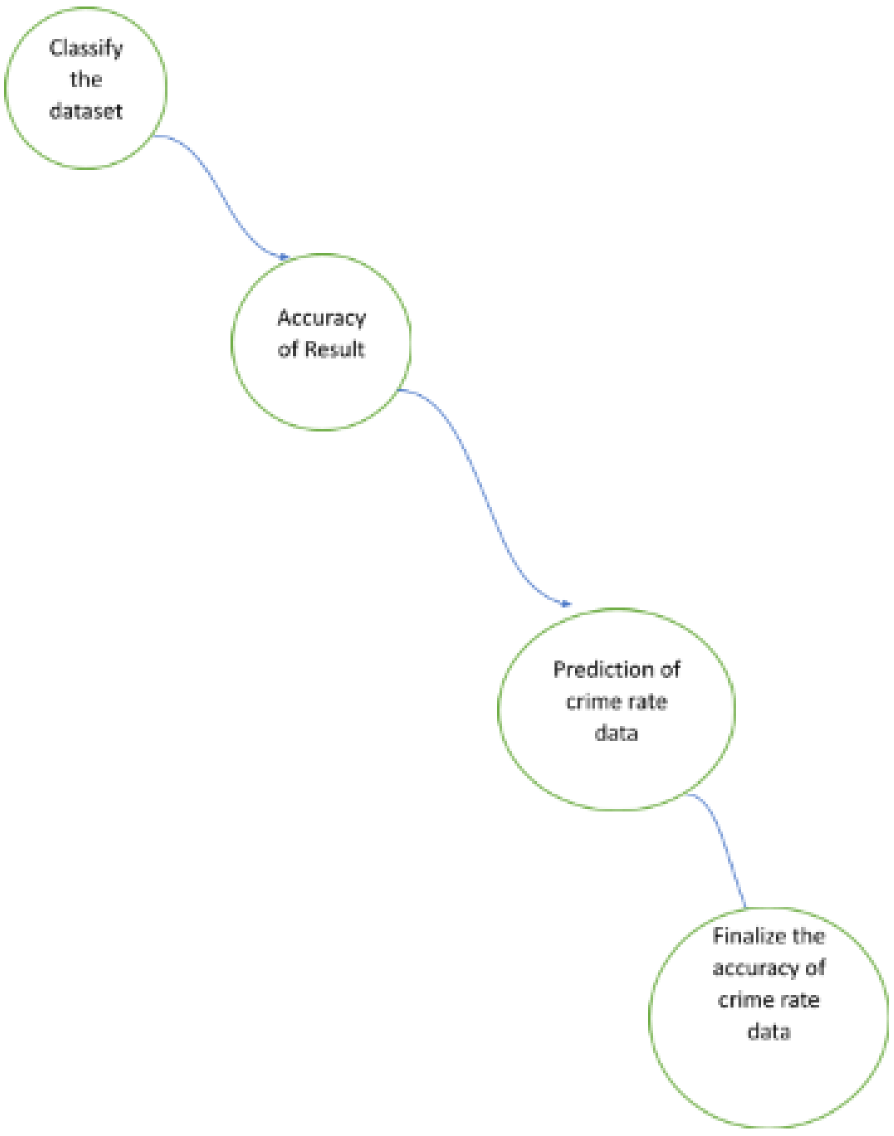
Level 0



Level 1


Dataset collection
Pre-proces sing
Feature Extraction
Apply Algorithm

Level 2


Classify the dataset
Accuracy of Result
Prediction of crime rate data
Finalize the accuracy of crime rate data

Uml Diagrams: -

The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development. UML offers a standard way to visualize a system's architectural blueprints, including elements such as:

- actors
- business processes
- (logical) components
- activities
- programming language statements
- database schemas, and
- Reusable software components.

UML combines best techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies· UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems.

Sequence Diagram:

Sequence Diagrams Represent the objects participating the interaction horizontally and time vertically. A Use Case is a kind of behavioral classifier that represents a declaration of an offered behavior. Each use case specifies some behavior, possibly including variants that the subject can perform in collaboration with one or more actors. Use cases define the offered behavior of the subject without reference to its internal structure. These behaviors, involving interactions between the actor and the subject, may result in changes to the state of the subject and communications with its environment. A use case can include possible variations of its basic behavior, including exceptional behavior and error handling.

Activity Diagrams-:

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

Usecase diagram:

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

UML was created by Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

OMG is continuously putting effort to make a truly industry standard.

UML stands for Unified Modeling Language.

UML is a pictorial language used to make software blue prints

Class diagram

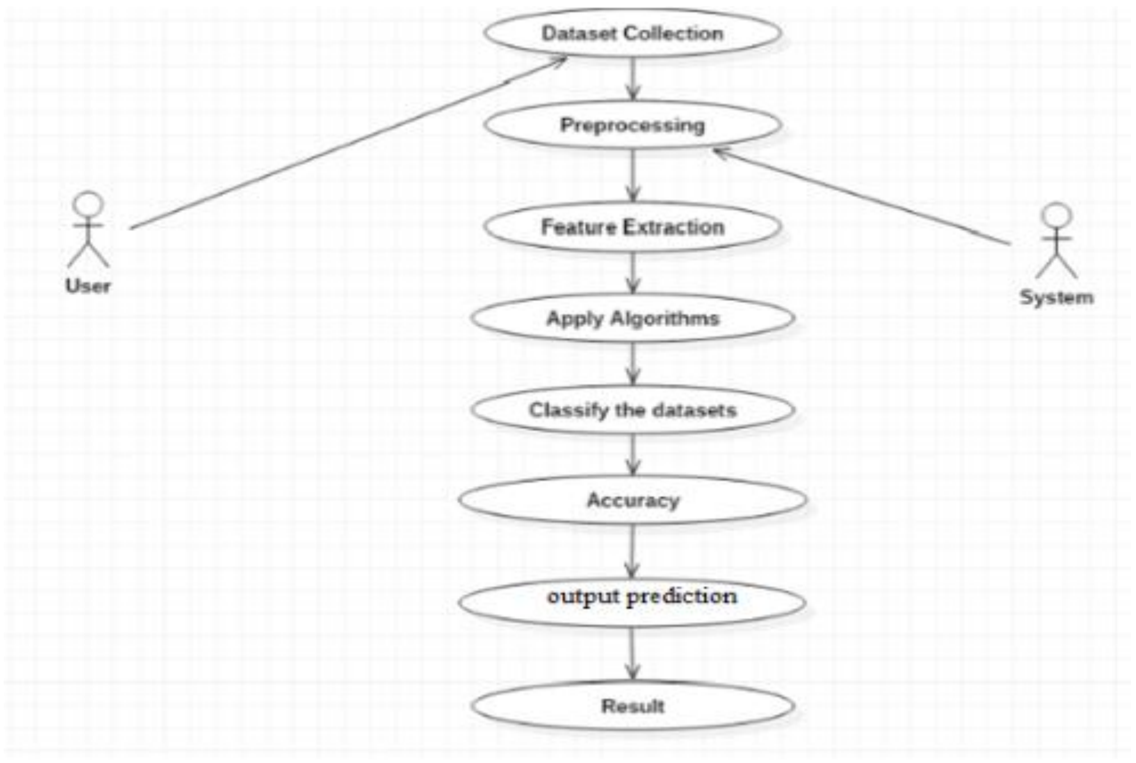
The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

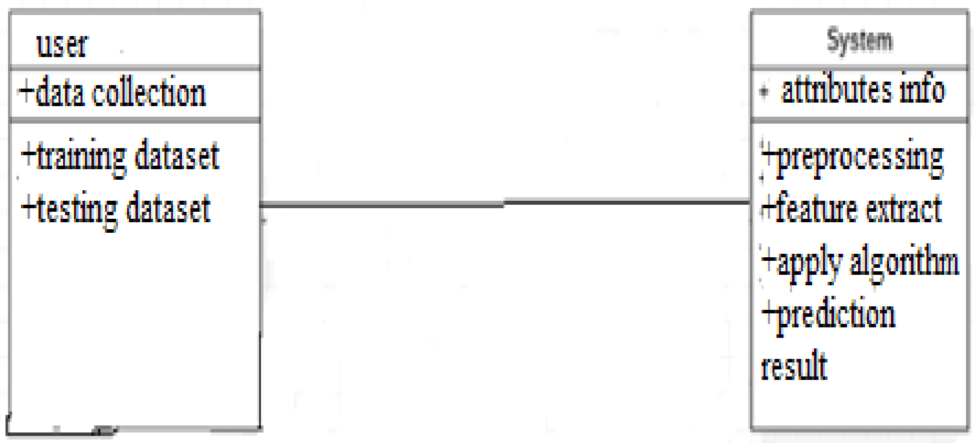
The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.

The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.

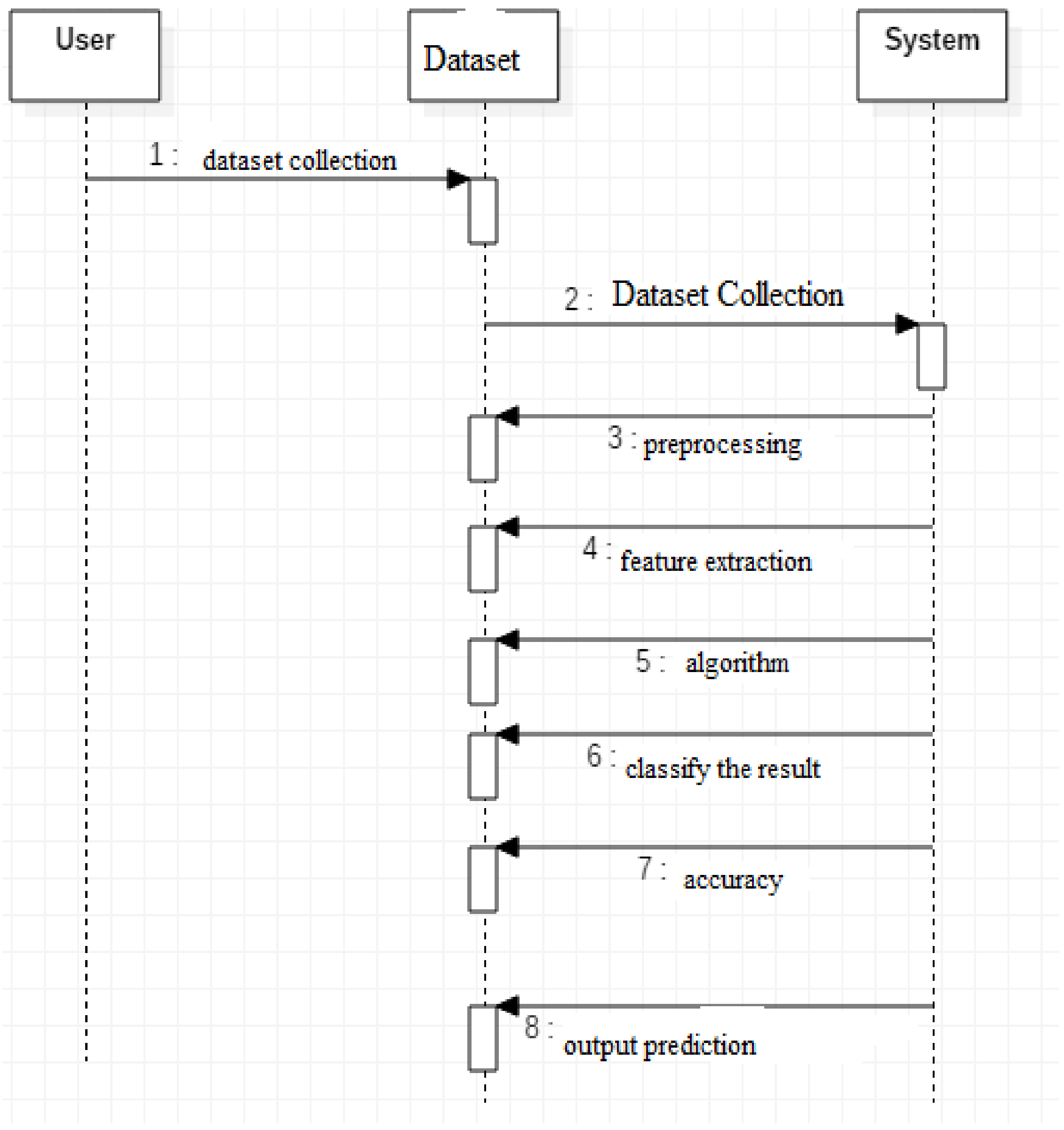
Use Case: -


Dataset Collection
Preprocessing
Feature Extraction
Apply Algorithms
Classify the datasets
Accuracy
output prediction
Result
User
System

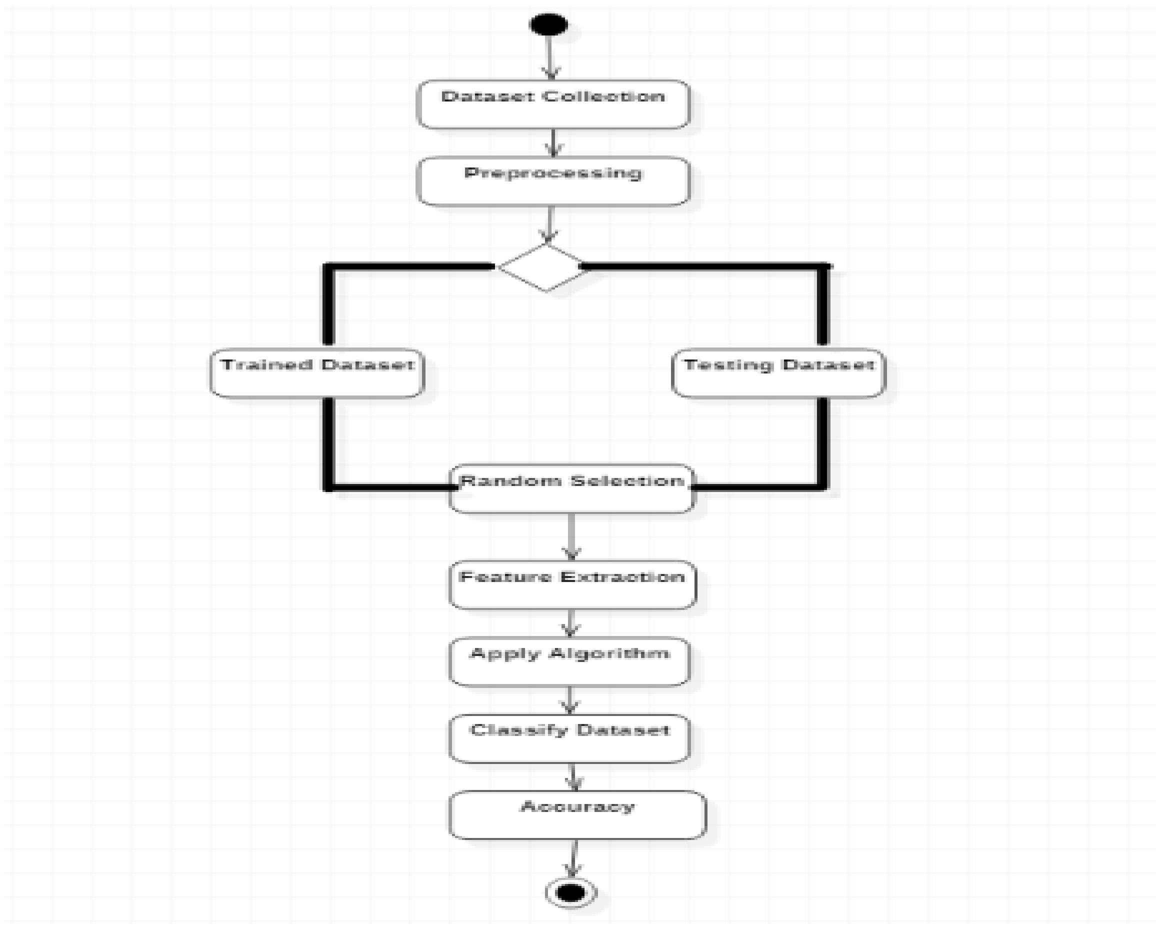
Class


user
+data collection
+training dataset
+testing dataset
System
+ attributes info
+preprocessing
+feature extract
+apply algorithm
+prediction
result

Sequence


User
Dataset
System
1 : dataset collection
2 : Dataset Collection
3 : preprocessing
4 : feature extraction
5 : algorithm
6 : classify the result
7 : accuracy
8 : output prediction

Activity


Dataset Collection
Preprocessing
Trained Dataset
Testing Dataset
Random Selection
Feature Extraction
Apply Algorithm
Classify Dataset
Accuracy

Source code: -

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['patch.force_edgecolor'] = True
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline

filepath = 'C:/Users/kommu/Downloads/TWITTER PROJECT/'
file= filepath+'training_data_2_csv_UTF.csv'

training_data = pd.read_csv(file)
bots = training_data[training_data.bot==1]
nonbots = training_data[training_data.bot==0]

### Exploratory Data Analysis

#### Identifying Missingness in the data

def get_heatmap(df):
    #This function gives heatmap of all NaN values
    plt.figure(figsize=(10,6))
    sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
    plt.tight_layout()
    return plt.show()

get_heatmap(training_data)

bots.friends_count/bots.followers_count

plt.figure(figsize=(10,5))
plt.subplot(2,1,1)
plt.title('Bots Friends vs Followers')
sns.regplot(bots.friends_count, bots.followers_count, color='red',
label='Bots')
plt.xlim(0, 100)
plt.ylim(0, 100)
plt.tight_layout()

plt.subplot(2,1,2)
plt.title('NonBots Friends vs Followers')
sns.regplot(nonbots.friends_count, nonbots.followers_count, color='blue',
label='NonBots')
plt.xlim(0, 100)
plt.ylim(0, 100)

plt.tight_layout()
plt.show()

#### Identifying Imbalance in the data

bots['friends_by_followers'] = bots.friends_count/bots.followers_count
bots[bots.friends_by_followers<1].shape

nonbots['friends_by_followers'] =
nonbots.friends_count/nonbots.followers_count
nonbots[nonbots.friends_by_followers<1].shape

plt.figure(figsize=(10,5))
plt.plot(bots.listed_count, color='red', label='Bots')
plt.plot(nonbots.listed_count, color='blue', label='NonBots')
plt.legend(loc='upper left')
plt.ylim(10000,20000)
print(bots[(bots.listed_count<5)].shape)

bots_listed_count_df = bots[bots.listed_count<16000]
nonbots_listed_count_df = nonbots[nonbots.listed_count<16000]

bots_verified_df =
bots_listed_count_df[bots_listed_count_df.verified==False]
bots_screenname_has_bot_df_ =
bots_verified_df[(bots_verified_df.screen_name.str.contains("bot",
case=False)==True)].shape
```

```
plt.figure(figsize=(12,7))

plt.subplot(2,1,1)
plt.plot(bots_listed_count_df.friends_count, color='red', label='Bots
Friends')
plt.plot(nonbots_listed_count_df.friends_count, color='blue',
label='NonBots Friends')
plt.legend(loc='upper left')

plt.subplot(2,1,2)
plt.plot(bots_listed_count_df.followers_count, color='red', label='Bots
Followers')
plt.plot(nonbots_listed_count_df.followers_count, color='blue',
label='NonBots Followers')
plt.legend(loc='upper left')

#bots[bots.listedcount>10000]
condition = (bots.screen_name.str.contains("bot",
case=False)==True)|(bots.description.str.contains("bot",
case=False)==True)|(bots.location.isnull())|(bots.verified==False)

bots['screen_name_binary'] = (bots.screen_name.str.contains("bot",
case=False)==True)
bots['location_binary'] = (bots.location.isnull())
bots['verified_binary'] = (bots.verified==False)
bots.shape

condition = (nonbots.screen_name.str.contains("bot",
case=False)==False)| (nonbots.description.str.contains("bot",
case=False)==False)
|(nonbots.location.isnull()==False)|(nonbots.verified==True)

nonbots['screen_name_binary'] =
(nonbots.screen_name.str.contains("bot", case=False)==False)
nonbots['location_binary'] = (nonbots.location.isnull()==False)
nonbots['verified_binary'] = (nonbots.verified==True)

nonbots.shape

df = pd.concat([bots, nonbots])
df.shape

### Feature Independence using Spearman correlation
```

```
df.corr(method='spearman')

plt.figure(figsize=(8,4))
sns.heatmap(df.corr(method='spearman'), cmap='coolwarm', annot=True)
plt.tight_layout()
plt.show()
```

Result:
- There is no correlation between **id, statuses_count, default_profile, default_profile_image** and target variable.
- There is strong correlation between **verified, listed_count, friends_count, followers_count** and target variable.
- We cannot perform correlation for categorical attributes. So we will take **screen_name, name, description, status** into feature engineering. While use **verified, listed_count** for feature extraction.

#### Performing Feature Engineering

```
#filepath = 'https://raw.githubusercontent.com/jubins/ML-TwitterBotDetection/master/FinalCode/kaggle_data/'
filepath ='C:/Users/kommu/Downloads/TWITTER PROJECT/'
file= open(filepath+'training_data_2_csv_UTF.csv', mode='r', encoding='utf-8', errors='ignore')

training_data = pd.read_csv(file)

bag_of_words_bot = r'bot|b0t|cannabis|tweet me|mishear|follow me|updates every|gorilla|yes_ofc|forget' \

r'expos|kill|clit|bbb|butt|fuck|XXX|sex|truthe|fake|anony|free|virus|funky|RNA|kuck|jargon' \

r'nerd|swag|jack|bang|bonsai|chick|prison|paper|pokem|xx|freak|ffd|dunia|clone|genie|bbb' \

r'ffd|onlyman|emoji|joke|troll|droop|free|every|wow|cheese|yeah|bio|magic|wizard|face'

training_data['screen_name_binary'] = training_data.screen_name.str.contains(bag_of_words_bot, case=False, na=False)
```

```
training_data['name_binary'] =
training_data.name.str.contains(bag_of_words_bot, case=False,
na=False)
training_data['description_binary'] =
training_data.description.str.contains(bag_of_words_bot, case=False,
na=False)
training_data['status_binary'] =
training_data.status.str.contains(bag_of_words_bot, case=False,
na=False)
```

#### Performing Feature Extraction

```
training_data['listed_count_binary'] =
(training_data.listed_count>20000)==False
features = ['screen_name_binary', 'name_binary', 'description_binary',
'status_binary', 'verified', 'followers_count', 'friends_count',
'statuses_count', 'listed_count_binary', 'bot']
```

## Implementing Different Models

#### **Decision Tree Classifier**

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_curve, auc
from sklearn.model_selection import train_test_split

X = training_data[features].iloc[:,:-1]
y = training_data[features].iloc[:,-1]

dt = DecisionTreeClassifier(criterion='entropy', min_samples_leaf=50,
min_samples_split=10)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=101)

dt = dt.fit(X_train, y_train)
y_pred_train = dt.predict(X_train)
y_pred_test = dt.predict(X_test)

print("Trainig Accuracy: %.5f" %accuracy_score(y_train, y_pred_train))
print("Test Accuracy: %.5f" %accuracy_score(y_test, y_pred_test))

sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'axes.grid' : False})

scores_train = dt.predict_proba(X_train)
scores_test = dt.predict_proba(X_test)

y_scores_train = []
y_scores_test = []
for i in range(len(scores_train)):
    y_scores_train.append(scores_train[i][1])

for i in range(len(scores_test)):
    y_scores_test.append(scores_test[i][1])

fpr_dt_train, tpr_dt_train, _ = roc_curve(y_train, y_scores_train,
pos_label=1)
fpr_dt_test, tpr_dt_test, _ = roc_curve(y_test, y_scores_test, pos_label=1)

plt.plot(fpr_dt_train, tpr_dt_train, color='darkblue', label='Train
AUC: %5f' %auc(fpr_dt_train, tpr_dt_train))
plt.plot(fpr_dt_test, tpr_dt_test, color='red', ls='--', label='Test
AUC: %5f' %auc(fpr_dt_test, tpr_dt_test))
plt.title("Decision Tree ROC Curve")
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.legend(loc='lower right')
```

Result: Decision Tree gives very good performance and generalizes well. But it may be overfitting as AUC is 0.937, so we will try other models.

#### **Multinomial Naive Bayes Classifier**

```
from sklearn.naive_bayes import MultinomialNB

X = training_data[features].iloc[:,:-1]
y = training_data[features].iloc[:,-1]

mnb = MultinomialNB(alpha=0.0009)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=101)

mnb = mnb.fit(X_train, y_train)
y_pred_train = mnb.predict(X_train)
```

```
y_pred_test = mnb.predict(X_test)

print("Trainig Accuracy: %.5f" %accuracy_score(y_train, y_pred_train))
print("Test Accuracy: %.5f" %accuracy_score(y_test, y_pred_test))

sns.set_style("whitegrid", {'axes.grid' : False})

scores_train = mnb.predict_proba(X_train)
scores_test = mnb.predict_proba(X_test)

y_scores_train = []
y_scores_test = []
for i in range(len(scores_train)):
    y_scores_train.append(scores_train[i][1])

for i in range(len(scores_test)):
    y_scores_test.append(scores_test[i][1])

fpr_mnb_train, tpr_mnb_train, _ = roc_curve(y_train, y_scores_train,
pos_label=1)
fpr_mnb_test, tpr_mnb_test, _ = roc_curve(y_test, y_scores_test,
pos_label=1)

plt.plot(fpr_mnb_train, tpr_mnb_train, color='darkblue', label='Train
AUC: %5f' %auc(fpr_mnb_train, tpr_mnb_train))
plt.plot(fpr_mnb_test, tpr_mnb_test, color='red', ls='--', label='Test
AUC: %5f' %auc(fpr_mnb_test, tpr_mnb_test))
plt.title("Multinomial NB ROC Curve")
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.legend(loc='lower right')
```

Result: Clearly, Multinomial Niave Bayes peforms poorly and is not a good choice as the Train AUC is just 0.556 and Test is 0.555.

#### **Random Forest Classifier**

```
from sklearn.ensemble import RandomForestClassifier

X = training_data[features].iloc[:,:-1]
y = training_data[features].iloc[:,-1]
```

```
rf = RandomForestClassifier(criterion='entropy', min_samples_leaf=100,
min_samples_split=20)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=101)

rf = rf.fit(X_train, y_train)
y_pred_train = rf.predict(X_train)
y_pred_test = rf.predict(X_test)

print("Trainig Accuracy: %.5f" %accuracy_score(y_train, y_pred_train))
print("Test Accuracy: %.5f" %accuracy_score(y_test, y_pred_test))

sns.set_style("whitegrid", {'axes.grid' : False})

scores_train = rf.predict_proba(X_train)
scores_test = rf.predict_proba(X_test)

y_scores_train = []
y_scores_test = []
for i in range(len(scores_train)):
    y_scores_train.append(scores_train[i][1])

for i in range(len(scores_test)):
    y_scores_test.append(scores_test[i][1])

fpr_rf_train, tpr_rf_train, _ = roc_curve(y_train, y_scores_train,
pos_label=1)
fpr_rf_test, tpr_rf_test, _ = roc_curve(y_test, y_scores_test, pos_label=1)

plt.plot(fpr_rf_train, tpr_rf_train, color='darkblue', label='Train
AUC: %5f' %auc(fpr_rf_train, tpr_rf_train))
plt.plot(fpr_rf_test, tpr_rf_test, color='red', ls='--', label='Test
AUC: %5f' %auc(fpr_rf_test, tpr_rf_test))
plt.title("Random ForestROC Curve")
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.legend(loc='lower right')
```

#### Our Classifier

```
class twitter_bot(object):
    def __init__(self):
        pass

    def perform_train_test_split(df):
        msk = np.random.rand(len(df)) < 0.75
        train, test = df[msk], df[~msk]
        X_train, y_train = train, train.ix[:,-1]
        X_test, y_test = test, test.ix[:, -1]
        return (X_train, y_train, X_test, y_test)

    def bot_prediction_algorithm(df):
        # creating copy of dataframe
        train_df = df.copy()
        # performing feature engineering on id and verfied columns
        # converting id to int
        train_df['id'] = train_df.id.apply(lambda x: int(x))
        #train_df['friends_count'] = train_df.friends_count.apply(lambda x:
int(x))
        train_df['followers_count'] = train_df.followers_count.apply(lambda
x: 0 if x=='None' else int(x))
        train_df['friends_count'] = train_df.friends_count.apply(lambda x: 0
if x=='None' else int(x))
        #We created two bag of words because more bow is stringent on test
data, so on all small dataset we check less
        if train_df.shape[0]>600:
            #bag_of_words_for_bot
            bag_of_words_bot = r'bot|b0t|cannabis|tweet me|mishear|follow
me|updates every|gorilla|yes_ofc|forget' \

r'expos|kill|clit|bbb|butt|fuck|XXX|sex|truthe|fake|anony|free|virus|funky|R
NA|kuck|jargon' \

r'nerd|swag|jack|bang|bonsai|chick|prison|paper|pokem|xx|freak|ffd|dunia|c
lone|genie|bbb' \

r'ffd|onlyman|emoji|joke|troll|droop|free|every|wow|cheese|yeah|bio|magic
|wizard|face'
        else:
            # bag_of_words_for_bot
            bag_of_words_bot = r'bot|b0t|cannabis|mishear|updates every'

        # converting verified into vectors
        train_df['verified'] = train_df.verified.apply(lambda x: 1 if ((x ==
True) or x == 'TRUE') else 0)
```

```
    # check if the name contains bot or screenname contains b0t
    condition = ((train_df.name.str.contains(bag_of_words_bot,
case=False, na=False)) |
            (train_df.description.str.contains(bag_of_words_bot,
case=False, na=False)) |
            (train_df.screen_name.str.contains(bag_of_words_bot,
case=False, na=False)) |
            (train_df.status.str.contains(bag_of_words_bot, case=False,
na=False))
            )  # these all are bots
    predicted_df = train_df[condition]  # these all are bots
    predicted_df.bot = 1
    predicted_df = predicted_df[['id', 'bot']]

    # check if the user is verified
    verified_df = train_df[~condition]
    condition = (verified_df.verified == 1)  # these all are nonbots
    predicted_df1 = verified_df[condition][['id', 'bot']]
    predicted_df1.bot = 0
    predicted_df = pd.concat([predicted_df, predicted_df1])

    # check if description contains buzzfeed
    buzzfeed_df = verified_df[~condition]
    condition = (buzzfeed_df.description.str.contains("buzzfeed",
case=False, na=False))  # these all are nonbots
    predicted_df1 =
buzzfeed_df[buzzfeed_df.description.str.contains("buzzfeed", case=False,
na=False)][['id', 'bot']]
    predicted_df1.bot = 0
    predicted_df = pd.concat([predicted_df, predicted_df1])

    # check if listed_count>16000
    listed_count_df = buzzfeed_df[~condition]
    listed_count_df.listed_count =
listed_count_df.listed_count.apply(lambda x: 0 if x == 'None' else x)
    listed_count_df.listed_count =
listed_count_df.listed_count.apply(lambda x: int(x))
    condition = (listed_count_df.listed_count > 16000)  # these all are
nonbots
    predicted_df1 = listed_count_df[condition][['id', 'bot']]
    predicted_df1.bot = 0
    predicted_df = pd.concat([predicted_df, predicted_df1])
```

```
        #remaining
        predicted_df1 = listed_count_df[~condition][['id', 'bot']]
        predicted_df1.bot = 0 # these all are nonbots
        predicted_df = pd.concat([predicted_df, predicted_df1])
        return predicted_df

    def get_predicted_and_true_values(features, target):
        y_pred, y_true =
twitter_bot.bot_prediction_algorithm(features).bot.tolist(), target.tolist()
        return (y_pred, y_true)

    def get_accuracy_score(df):
        (X_train, y_train, X_test, y_test) =
twitter_bot.perform_train_test_split(df)
        # predictions on training data
        y_pred_train, y_true_train =
twitter_bot.get_predicted_and_true_values(X_train, y_train)
        train_acc = metrics.accuracy_score(y_pred_train, y_true_train)
        #predictions on test data
        y_pred_test, y_true_test =
twitter_bot.get_predicted_and_true_values(X_test, y_test)
        test_acc = metrics.accuracy_score(y_pred_test, y_true_test)
        return (train_acc, test_acc)

    def plot_roc_curve(df):
        (X_train, y_train, X_test, y_test) =
twitter_bot.perform_train_test_split(df)
        # Train ROC
        y_pred_train, y_true =
twitter_bot.get_predicted_and_true_values(X_train, y_train)
        scores = np.linspace(start=0.01, stop=0.9, num=len(y_true))
        fpr_train, tpr_train, threshold = metrics.roc_curve(y_pred_train,
scores, pos_label=0)
        plt.plot(fpr_train, tpr_train, label='Train AUC: %5f' %
metrics.auc(fpr_train, tpr_train), color='darkblue')
        #Test ROC
        y_pred_test, y_true =
twitter_bot.get_predicted_and_true_values(X_test, y_test)
        scores = np.linspace(start=0.01, stop=0.9, num=len(y_true))
        fpr_test, tpr_test, threshold = metrics.roc_curve(y_pred_test, scores,
pos_label=0)
```

```
        plt.plot(fpr_test,tpr_test, label='Test
AUC: %5f' %metrics.auc(fpr_test,tpr_test), ls='--', color='red')
        #Misc
        plt.xlim([-0.1,1])
        plt.title("Reciever Operating Characteristic (ROC)")
        plt.xlabel("False Positive Rate (FPR)")
        plt.ylabel("True Positive Rate (TPR)")
        plt.legend(loc='lower right')
        plt.show()



if __name__ == '__main__':
    start = time.time()
    filepath = 'https://raw.githubusercontent.com/jubins/ML-
TwitterBotDetection/master/FinalProjectAndCode/kaggle_data/'
    train_df = pd.read_csv(filepath + 'training_data_2_csv_UTF.csv')
    test_df = pd.read_csv(filepath + 'test_data_4_students.csv', sep='\t')
    print("Train Accuracy: ", twitter_bot.get_accuracy_score(train_df)[0])
    print("Test Accuracy: ", twitter_bot.get_accuracy_score(train_df)[1])

    #predicting test data results
    predicted_df = twitter_bot.bot_prediction_algorithm(test_df)
    #plotting the ROC curve
    twitter_bot.plot_roc_curve(train_df)
```

### **ROC Comparison after tuning the baseline model**

```
plt.figure(figsize=(14,10))
(X_train, y_train, X_test, y_test) =
twitter_bot.perform_train_test_split(df)
```
#Train ROC
```
y_pred_train, y_true =
twitter_bot.get_predicted_and_true_values(X_train, y_train)
scores = np.linspace(start=0, stop=1, num=len(y_true))
fpr_botc_train, tpr_botc_train, threshold =
metrics.roc_curve(y_pred_train, scores, pos_label=0)
```

#Test ROC
```
y_pred_test, y_true = twitter_bot.get_predicted_and_true_values(X_test,
y_test)
scores = np.linspace(start=0, stop=1, num=len(y_true))
```

```
fpr_botc_test, tpr_botc_test, threshold = metrics.roc_curve(y_pred_test,
scores, pos_label=0)

#Train ROC
plt.subplot(2,2,1)
plt.plot(fpr_botc_train, tpr_botc_train, label='Our Classifier
AUC: %5f' % metrics.auc(fpr_botc_train,tpr_botc_train),
color='darkblue')
plt.plot(fpr_rf_train, tpr_rf_train, label='Random Forest
AUC: %5f' %auc(fpr_rf_train, tpr_rf_train))
plt.plot(fpr_dt_train, tpr_dt_train, label='Decision Tree
AUC: %5f' %auc(fpr_dt_train, tpr_dt_train))
plt.plot(fpr_mnb_train, tpr_mnb_train, label='MultinomialNB
AUC: %5f' %auc(fpr_mnb_train, tpr_mnb_train))
plt.title("Training Set ROC Curve")
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.legend(loc='lower right')

#Test ROC
plt.subplot(2,2,2)
plt.plot(fpr_botc_test,tpr_botc_test, label='Our Classifier
AUC: %5f' %metrics.auc(fpr_botc_test,tpr_botc_test), color='darkblue')
plt.plot(fpr_rf_test, tpr_rf_test, label='Random Forest
AUC: %5f' %auc(fpr_rf_test, tpr_rf_test))
plt.plot(fpr_dt_test, tpr_dt_test, label='Decision Tree
AUC: %5f' %auc(fpr_dt_test, tpr_dt_test))
plt.plot(fpr_mnb_test, tpr_mnb_test, label='MultinomialNB
AUC: %5f' %auc(fpr_mnb_test, tpr_mnb_test))
plt.title("Test Set ROC Curve")
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.legend(loc='lower right')
plt.tight_layout()
```

### Thank you.

MACHINE LEARNING

Machine Learning is a system that can learn from example through self-improvement and without being explicitly coded by programmer. The breakthrough comes with the idea that a machine can singularly learn from the data (i.e., example) to produce accurate results.

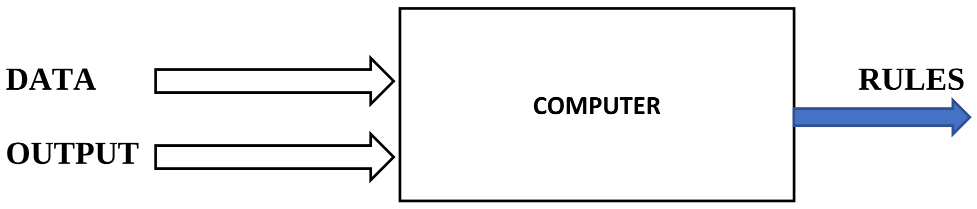
Machine learning combines data with statistical tools to predict an output. This output is then used by corporate to makes actionable insights. Machine learning is closely related to data mining and Bayesian predictive modeling. The machine receives data as input, use an algorithm to formulate answers.

A typical machine learning tasks are to provide a recommendation. For those who have a Netflix account, all recommendations of movies or series are based on the user's historical data. Tech companies are using unsupervised learning to improve the user experience with personalizing recommendation.

Machine learning is also used for a variety of task like fraud detection, predictive maintenance, portfolio optimization, automatize task and so on.

Machine Learning vs. Traditional Programming

Traditional programming differs significantly from machine learning. In traditional programming, a programmer code all the rules in consultation with an expert in the industry for which software is being developed. Each rule is based on a logical foundation; the machine will execute an output following the logical statement. When the system grows complex, more rules need to be written. It can quickly become unsustainable to maintain.

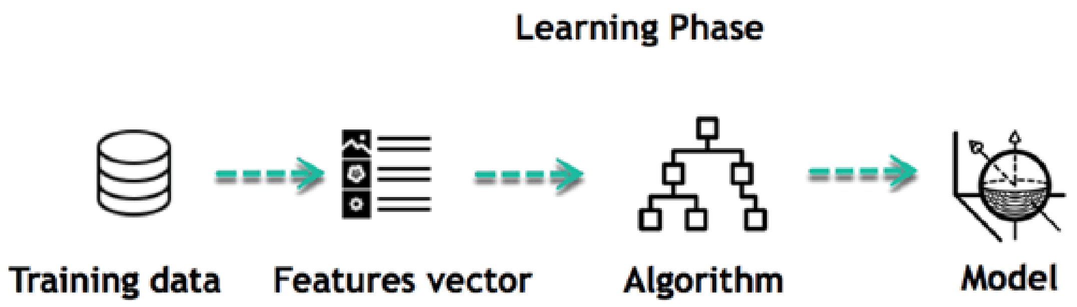


How does Machine learning work?

Machine learning is the brain where all the learning takes place. The way the machine learns is similar to the human being. Humans learn from experience. The more we know, the more easily we can predict. By analogy, when we face an unknown situation, the likelihood of success is lower than the known situation. Machines are trained the same. To make an accurate prediction, the machine sees an example. When we give the machine a similar example, it can figure out the outcome. However, like a human, if it’s feed a previously unseen example, the machine has difficulties to predict.

The core objective of machine learning is the **learning** and **inference**. First of all, the machine learns through the discovery of patterns. This discovery is made thanks to the **data**. One crucial part of the data scientist is to choose carefully which data to provide to the machine. The list of attributes used to solve a problem is called a **feature vector.** You can think of a feature vector as a subset of data that is used to tackle a problem.

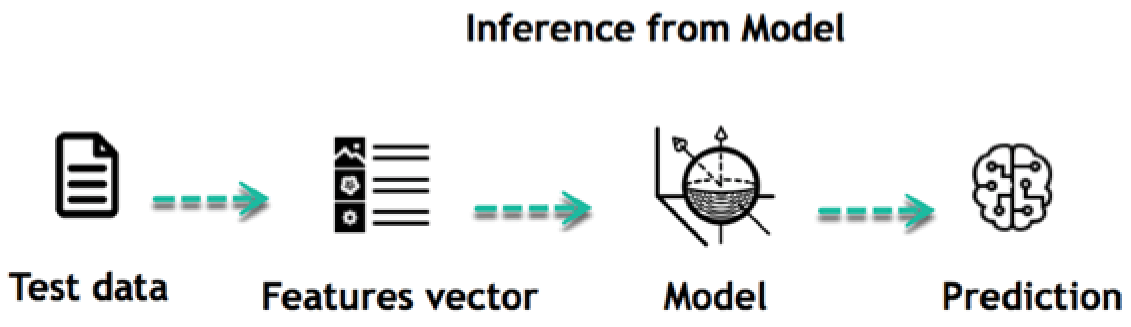
The machine uses some fancy algorithms to simplify the reality and transform this discovery into a **model**. Therefore, the learning stage is used to describe the data and summarize it into a model.



For instance, the machine is trying to understand the relationship between the wage of an individual and the likelihood to go to a fancy restaurant. It turns out the machine finds a positive relationship between wage and going to a high-end restaurant: This is the model

Inferring

When the model is built, it is possible to test how powerful it is on never-seen-before data. The new data are transformed into a features vector, go through the model and give a prediction. This is all the beautiful part of machine learning. There is no need to update the rules or train again the model. You can use the model previously trained to make inference on new data.



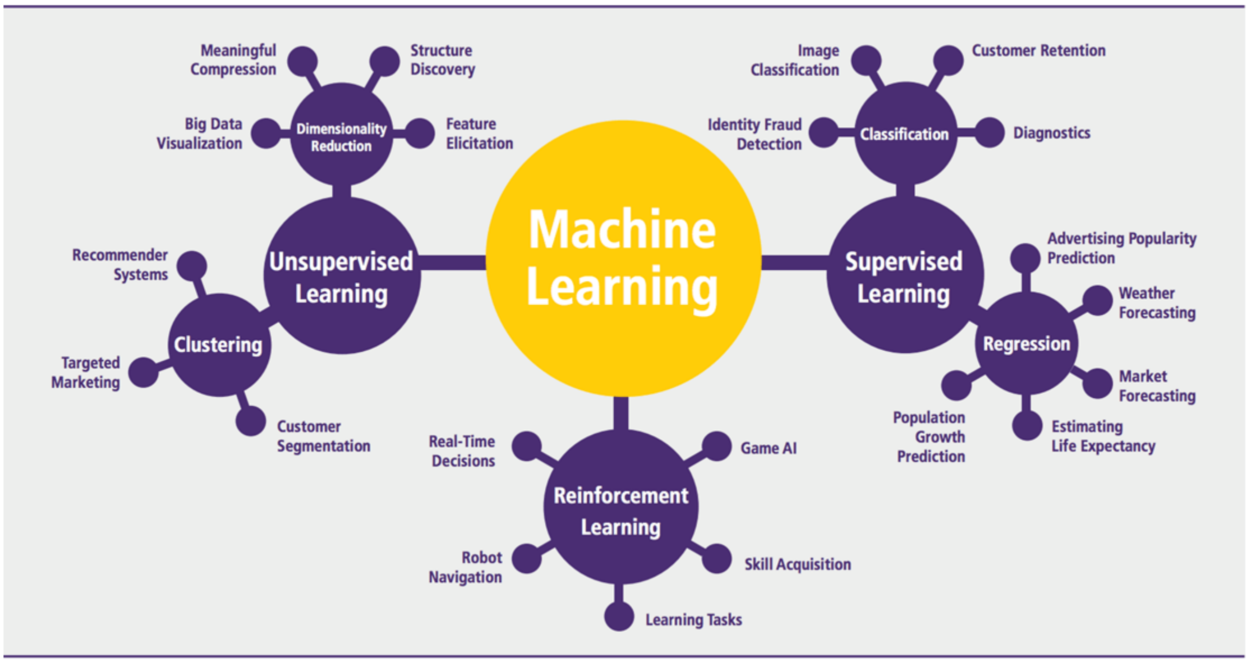
The life of Machine Learning programs is straightforward and can be summarized in the following points:

1. Define a question
2. Collect data
3. Visualize data
4. Train algorithm

5. Test the Algorithm
6. Collect feedback
7. Refine the algorithm
8. Loop 4-7 until the results are satisfying
9. Use the model to make a prediction

Once the algorithm gets good at drawing the right conclusions, it applies that knowledge to new sets of data.

Machine learning Algorithms and where they are used?



Machine learning can be grouped into two broad learning tasks: Supervised and Unsupervised. There are many other algorithms

Supervised learning

An algorithm uses training data and feedback from humans to learn the relationship of given inputs to a given output. For instance, a practitioner can use marketing expense and weather forecast as input data to predict the sales of cans.

You can use supervised learning when the output data is known. The algorithm will predict new data.

There are two categories of supervised learning:

Algorithm Name	**Description**	**Type**
Linear regression	Finds a way to correlate each feature to the output to help predict future values.	Regression
Logistic regression	Extension of linear regression that's used for classification tasks. The output variable 3is binary (e.g., only black or white) rather than continuous (e.g., an infinite list of potential colors)	Classification
Decision tree	Highly interpretable classification or regression model that splits data-feature values into branches at decision nodes (e.g., if a feature is a color, each possible color becomes a new branch) until a final decision output is made	Regression Classification
Naive Bayes	The Bayesian method is a classification method that makes use of the Bayesian theorem. The theorem updates the prior knowledge of an event with the independent probability of each feature that can affect the event.	Regression Classification
Support vector machine	Support Vector Machine, or SVM, is typically used for the classification task. SVM algorithm finds a hyperplane that optimally divided the classes. It is best used with a non-linear solver.	Regression (not very common) Classification
Random forest	The algorithm is built upon a decision tree to improve the accuracy drastically. Random forest generates many times simple decision trees and uses the 'majority vote' method to decide on which label to return. For the classification task, the final prediction will be the one with the most vote; while for the regression task, the average prediction of all the trees is the final prediction.	Regression Classification
AdaBoost	Classification or regression technique that uses a multitude of models to come up with a decision but weighs them based on their accuracy in predicting the outcome	Regression Classification

Gradient-boosting trees	Gradient-boosting trees is a state-of-the-art classification/regression technique. It is focusing on the error committed by the previous trees and tries to correct it.	Regression Classification

- Classification task
- Regression task

Classification

Imagine you want to predict the gender of a customer for a commercial. You will start gathering data on the height, weight, job, salary, purchasing basket, etc. from your customer database. You know the gender of each of your customer, it can only be male or female. The objective of the classifier will be to assign a probability of being a male or a female (i.e., the label) based on the information (i.e., features you have collected). When the model learned how to recognize male or female, you can use new data to make a prediction. For instance, you just got new information from an unknown customer, and you want to know if it is a male or female. If the classifier predicts male = 70%, it means the algorithm is sure at 70% that this customer is a male, and 30% it is a female.

The label can be of two or more classes. The above example has only two classes, but if a classifier needs to predict object, it has dozens of classes (e.g., glass, table, shoes, etc. each object represents a class)

Regression

When the output is a continuous value, the task is a regression. For instance, a financial analyst may need to forecast the value of a stock based on a range of feature like equity, previous stock performances, macroeconomics index. The system will be trained to estimate the price of the stocks with the lowest possible error.

Unsupervised learning

In unsupervised learning, an algorithm explores input data without being given an explicit output variable (e.g., explores customer demographic data to identify patterns)

You can use it when you do not know how to classify the data, and you want the algorithm to find patterns and classify the data for you

Algorithm	**Description**	**Type**
K-means clustering	Puts data into some groups (k) that each contains data with similar characteristics (as determined by the model, not in advance by humans)	Clustering
Gaussian mixture model	A generalization of k-means clustering that provides more flexibility in the size and shape of groups (clusters	Clustering
Hierarchical clustering	Splits clusters along a hierarchical tree to form a classification system. Can be used for Cluster loyalty-card customer	Clustering
Recommender system	Help to define the relevant data for making a recommendation.	Clustering
PCA/T-SNE	Mostly used to decrease the dimensionality of the data. The algorithms reduce the number of features to 3 or 4 vectors with the highest variances.	Dimension Reduction

Application of Machine learning

Augmentation:

- Machine learning, which assists humans with their day-to-day tasks, personally or commercially without having complete control of the output. Such machine learning is used in different ways such as Virtual Assistant, Data analysis, software solutions. The primary user is to reduce errors due to human bias.

Automation:

- Machine learning, which works entirely autonomously in any field without the need for any human intervention. For example, robots performing the essential process steps in manufacturing plants.

Finance Industry

- Machine learning is growing in popularity in the finance industry. Banks are mainly using ML to find patterns inside the data but also to prevent fraud.

Government organization

- The government makes use of ML to manage public safety and utilities. Take the example of China with the massive face recognition. The government uses Artificial intelligence to prevent jaywalker.

Healthcare industry

- Healthcare was one of the first industry to use machine learning with image detection.

Marketing

- Broad use of AI is done in marketing thanks to abundant access to data. Before the age of mass data, researchers develop advanced mathematical tools like Bayesian analysis to estimate the value of a customer. With the boom of data, marketing department relies on AI to optimize the customer relationship and marketing campaign.

Example of application of Machine Learning in Supply Chain

Machine learning gives terrific results for visual pattern recognition, opening up many potential applications in physical inspection and maintenance across the entire supply chain network.

Unsupervised learning can quickly search for comparable patterns in the diverse dataset. In turn, the machine can perform quality inspection throughout the logistics hub, shipment with damage and wear.

For instance, IBM's Watson platform can determine shipping container damage. Watson combines visual and systems-based data to track, report and make recommendations in real-time.

In past year stock manager relies extensively on the primary method to evaluate and forecast the inventory. When combining big data and machine learning, better forecasting techniques have been implemented (an improvement of 20 to 30 % over traditional forecasting tools). In term of sales, it means an increase of 2 to 3 % due to the potential reduction in inventory costs.

Example of Machine Learning Google Car

For example, everybody knows the Google car. The car is full of lasers on the roof which are telling it where it is regarding the surrounding area. It has radar in the front, which is informing the car of the speed and motion of all the cars around it. It uses all of that data to figure out not only how to drive the car but also to figure out and predict what potential drivers around the car are going to do. What's impressive is that the car is processing almost a gigabyte a second of data.

PYTHON OVERVIEW

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a general-purpose, versatile and popular programming language. It's great as a first language because it is concise and easy to read, and it is also a good language to have in any programmer's stack as it can be used for everything from web development to software development and scientific applications.

It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time.

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, Unix shell, and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

Python's features include:

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.

- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- IT supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python Modules

NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

● A powerful N-dimensional array object

● Sophisticated (broadcasting) functions

● Tools for integrating C/C++ and Fortran code

● Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and timeseries data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open-source data analysis/manipulation tool available in any language.

Pandas is well suited for many different kinds of data:

● Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet

● Ordered and unordered (not necessarily fixed-frequency) time-series data.

● Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels

● Any other form of observational/statistical data sets. The data need not be labeled at all to be placed into a pandas data structure.

The two primary data structures of pandas, Series (1-dimensional) and Data Frame (2- dimensional) handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, Data Frame provides everything that R's data. The frame provides and much more. pandas are built on top of NumPy and are intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Matplotlib

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram, etc. For simple plotting, the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or a set of functions familiar to MATLAB users.

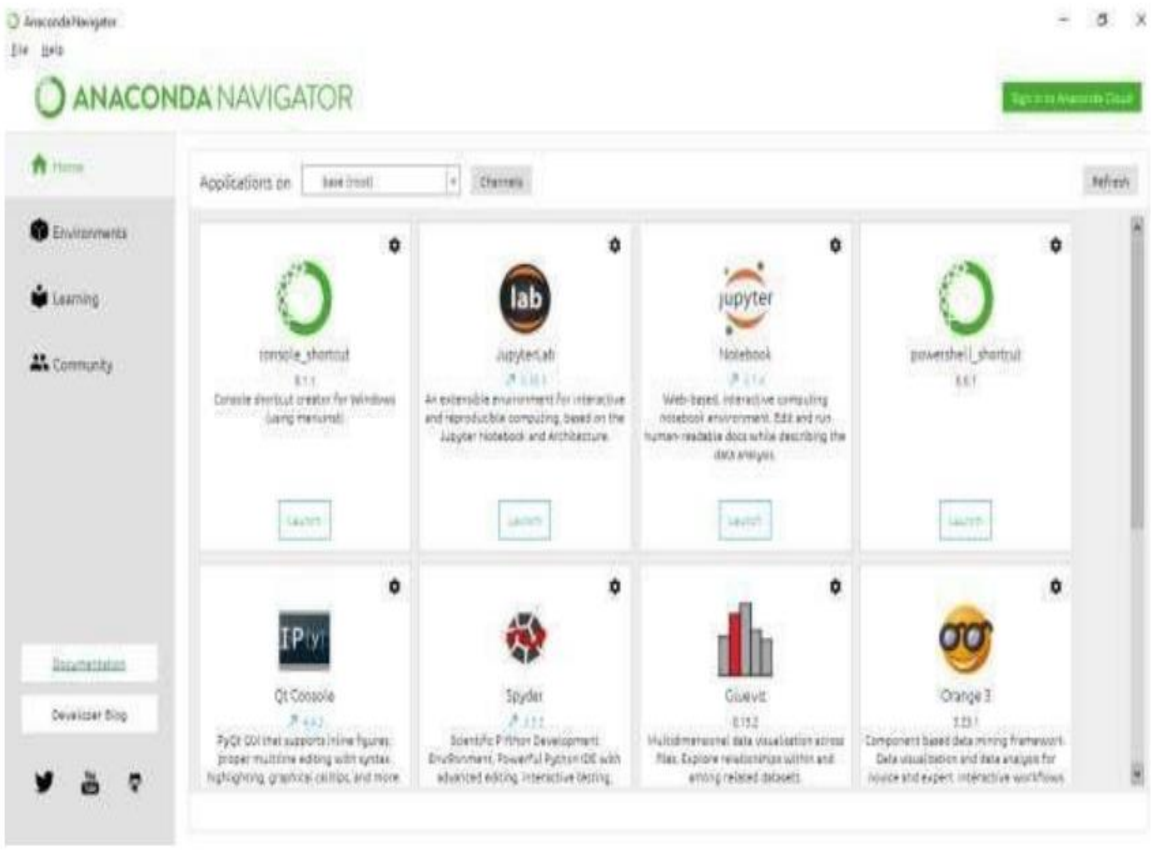
Seaborn

Seaborn is a library for making statistical graphics in **Python.** It is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data.

Its plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them

ANACONDA NAVIGATOR

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows you to launch applications and easily manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It is available for Windows, mac OS and Linux.



Console shortcut creator for Windows (using menuinst)

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing,

Multidimensional data visualization across files. Explore relationships within and among related datasets.

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows

Figure 1.1 Anaconda Navigator window on Windows operating system

Why use Navigator?

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages, and use multiple environments to separate these different versions.

The command line program conda is both a package manager and an environment manager, to help data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages and update them, all inside Navigator.

WHAT APPLICATIONS CAN I ACCESS USING NAVIGATOR?

The following applications are available by default in Navigator:

- Jupyter Lab
- Jupyter Notebook
- QT Console
- Spyder
- VS Code
- Glue viz
- Orange 3 App
- Rodeo
- RStudio

Advanced conda users can also build your own Navigator applications
How can I run code with Navigator?

The simplest way is with Spyder. From the Navigator Home tab, click Spyder, and write and execute your code.

You can also use Jupyter Notebooks the same way. Jupyter Notebooks are an increasingly popular system that combine your code, descriptive text, output, images and interactive interfaces into a single notebook file that is edited, viewed and used in a web browser.

Testing: -

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software Testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs.

Software Testing can also be stated as the process of validating and verifying that a software program/application/product:

- Meets the business and technical requirements that guided its design and Development.
- Works as expected and can be implemented with the same characteristics.

TESTING METHODS

Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Functions: Identified functions must be exercised.
- Output: Identified classes of software outputs must be exercised.
- Systems/Procedures: system should work properly

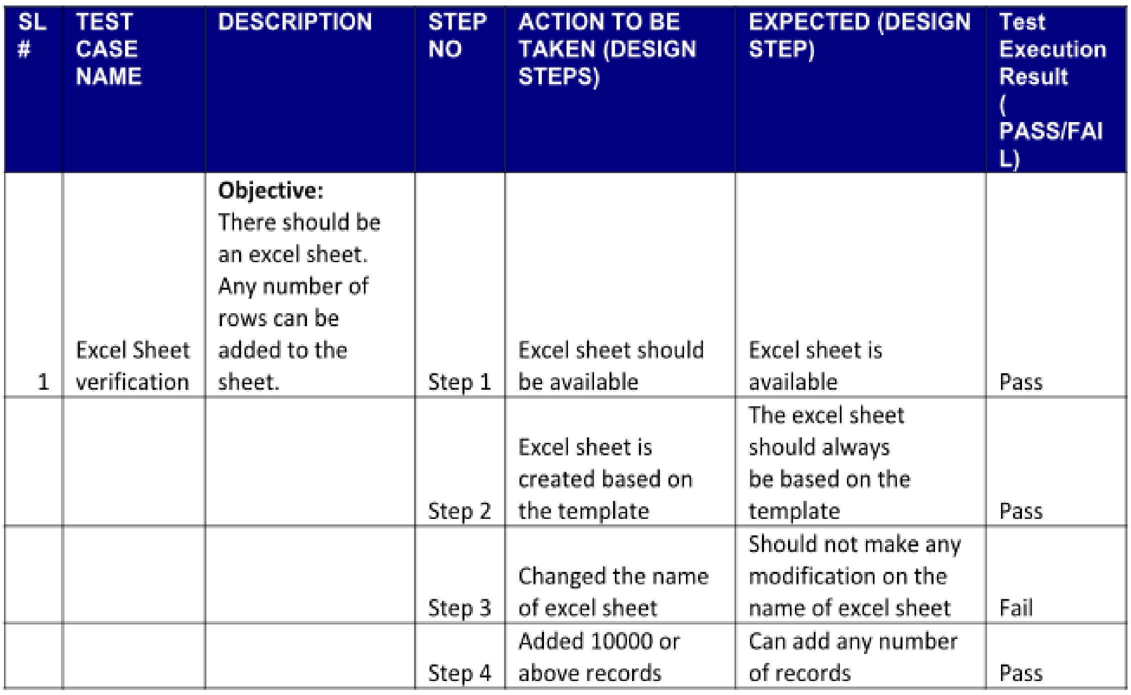
Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

Test Case for Excel Sheet Verification:

Here in machine learning we are dealing with dataset which is in excel sheet format so if any test case we need means we need to check excel file. Later on, classification will work on the respective columns of dataset

Test Case 1:



SL #	TEST CASE NAME	DESCRIPTION	STEP NO	ACTION TO BE TAKEN (DESIGN STEPS)	EXPECTED (DESIGN STEP)	Test Execution Result (PASS/FAIL)
1	Excel Sheet verification	**Objective:** There should be an excel sheet. Any number of rows can be added to the sheet.	Step 1	Excel sheet should be available	Excel sheet is available	Pass
			Step 2	Excel sheet is created based on the template	The excel sheet should always be based on the template	Pass
			Step 3	Changed the name of excel sheet	Should not make any modification on the name of excel sheet	Fail
			Step 4	Added 10000 or above records	Can add any number of records	Pass

ROC curve –

A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

• True Positive Rate – Recall

• False Positive Rate

Model Performances

Decision Tree Classifier

ROC curve for Decision Trees

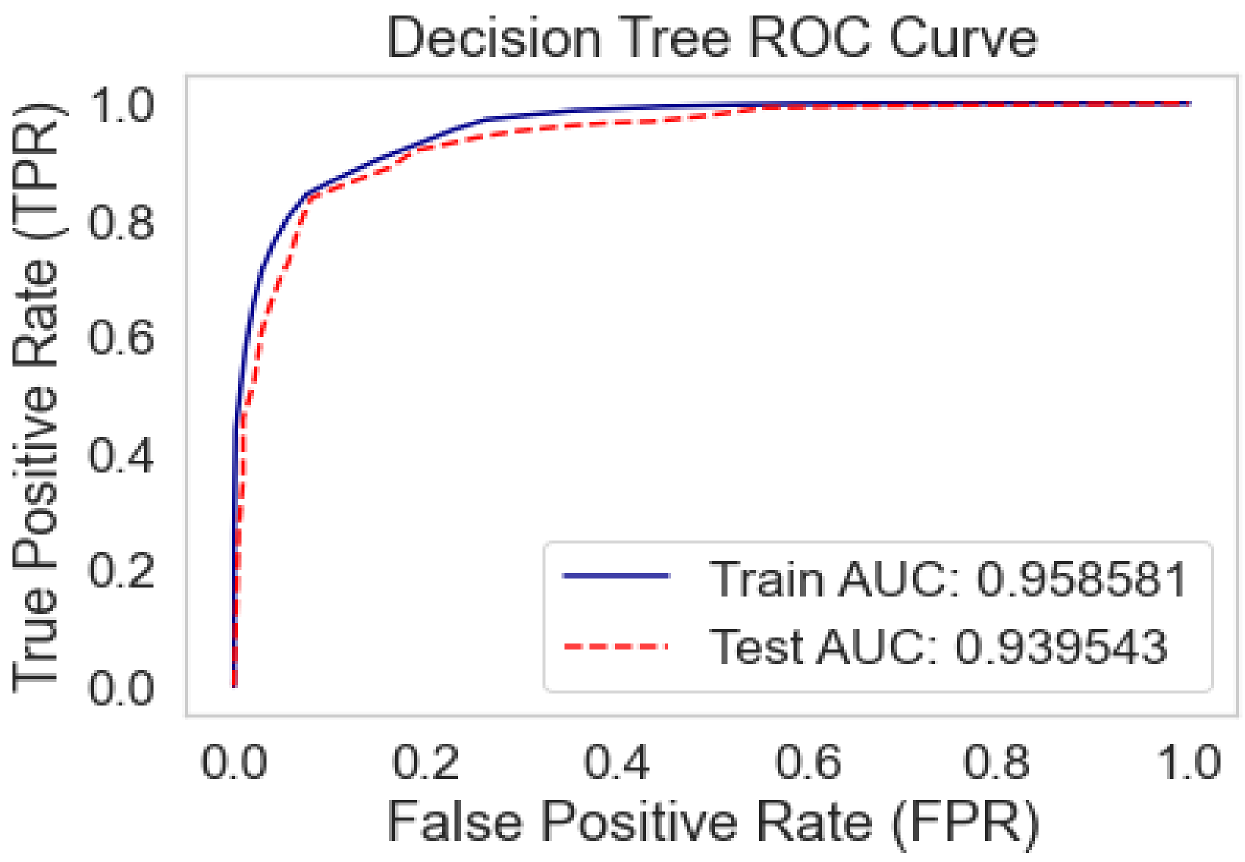


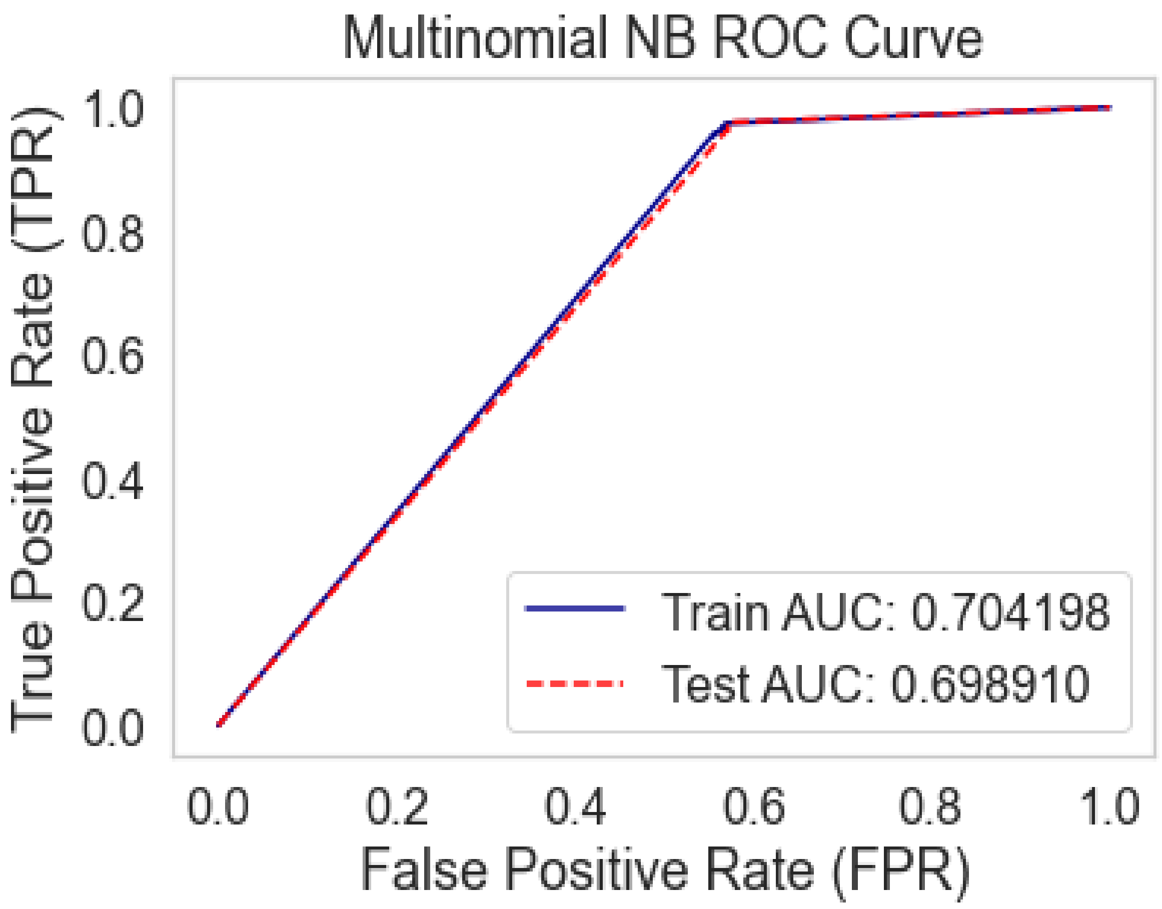
Figure: ROC curve for Decision Trees

From the above classification report and ROC, the following information can be concluded:

Accuracy of a decision tree is 0.88707

Result: The decision Tree gives very good performance and generalizes well. But it may be overfitting as AUC is 0.937, so we will try other models

Multinomial Naïve Bayes Classifier



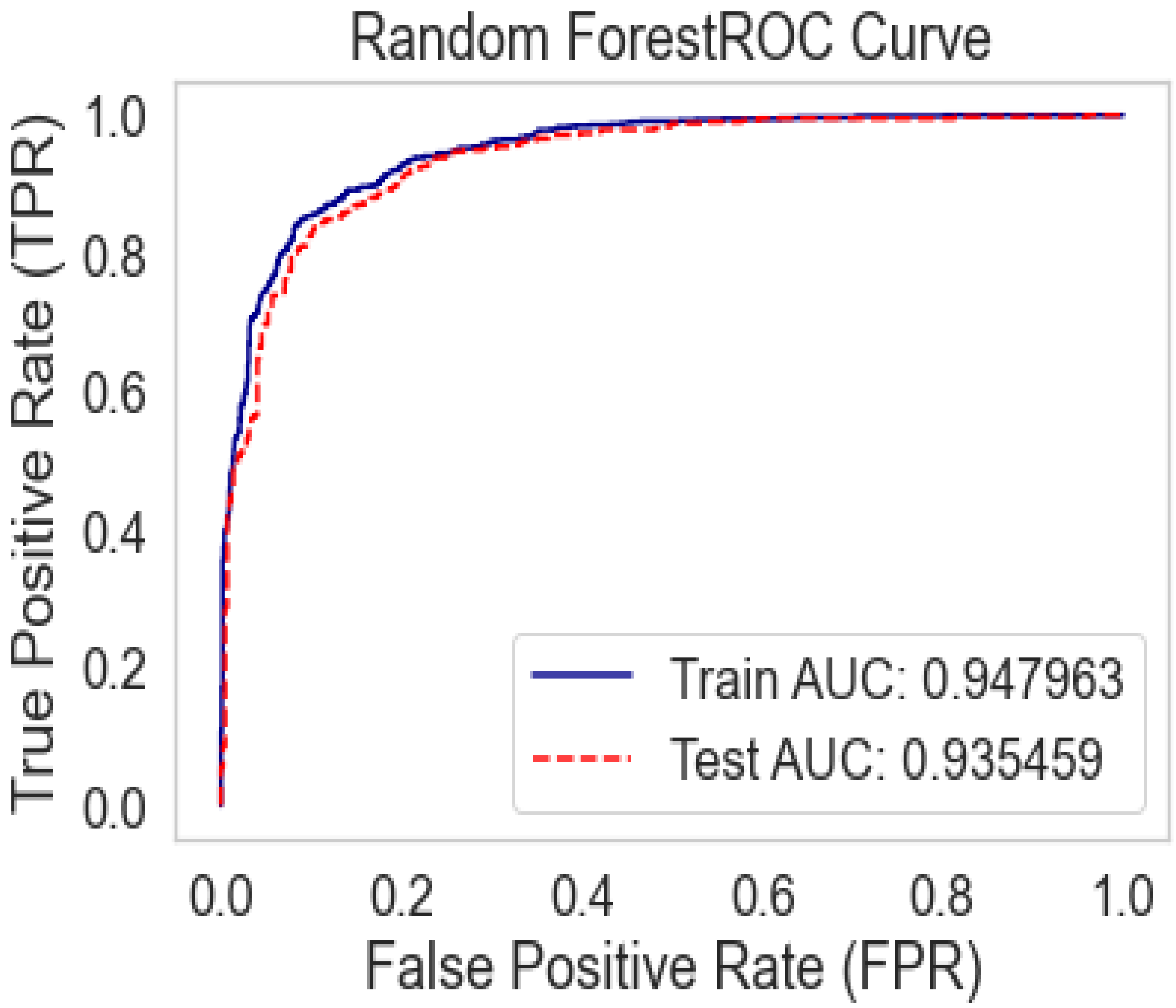
ROC Curve for Multinomial Naïve Bayes

From the above classification report and ROC, the following information can be concluded:

Accuracy of a Multinomial Naïve Bayes is: 68.44

Result: Clearly, Multinomial Naïve Bayes performs poorly and is not a good choice as the Train AUC is just 0.556 and Test is 0.555.

Random Forest Classifier

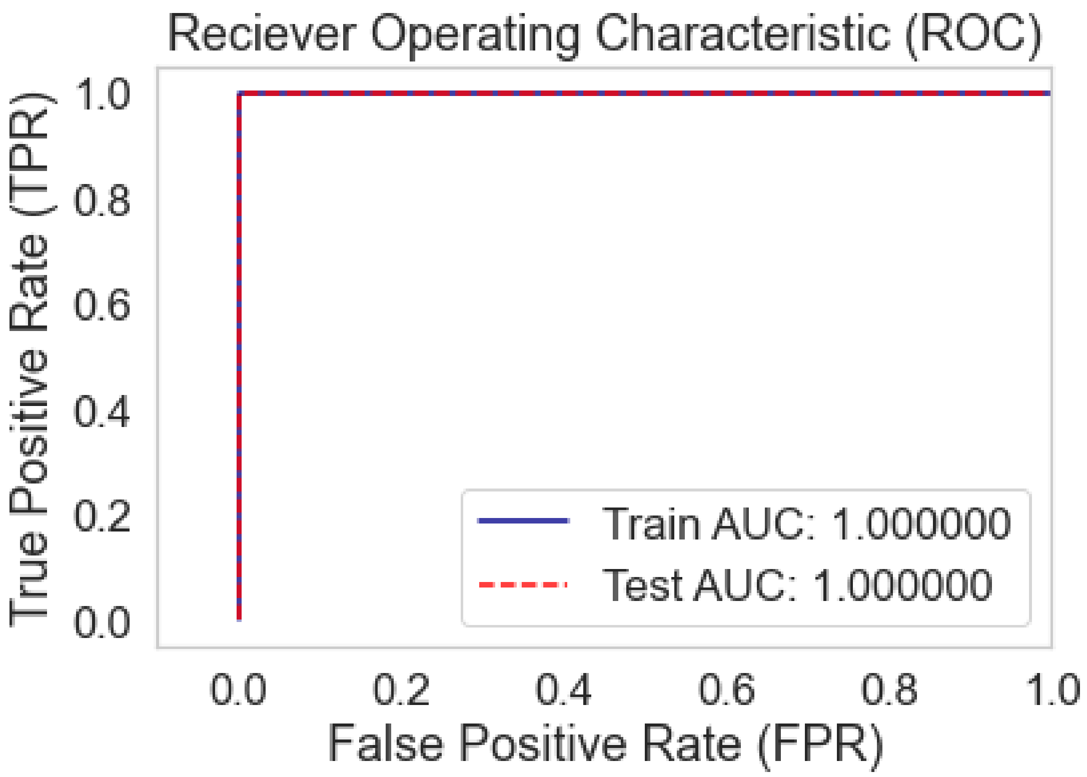


ROC Curve for Random Forest

From the above classification report and ROC, the following information can be concluded:

Accuracy of random forest is 87.174

Bag of words Model

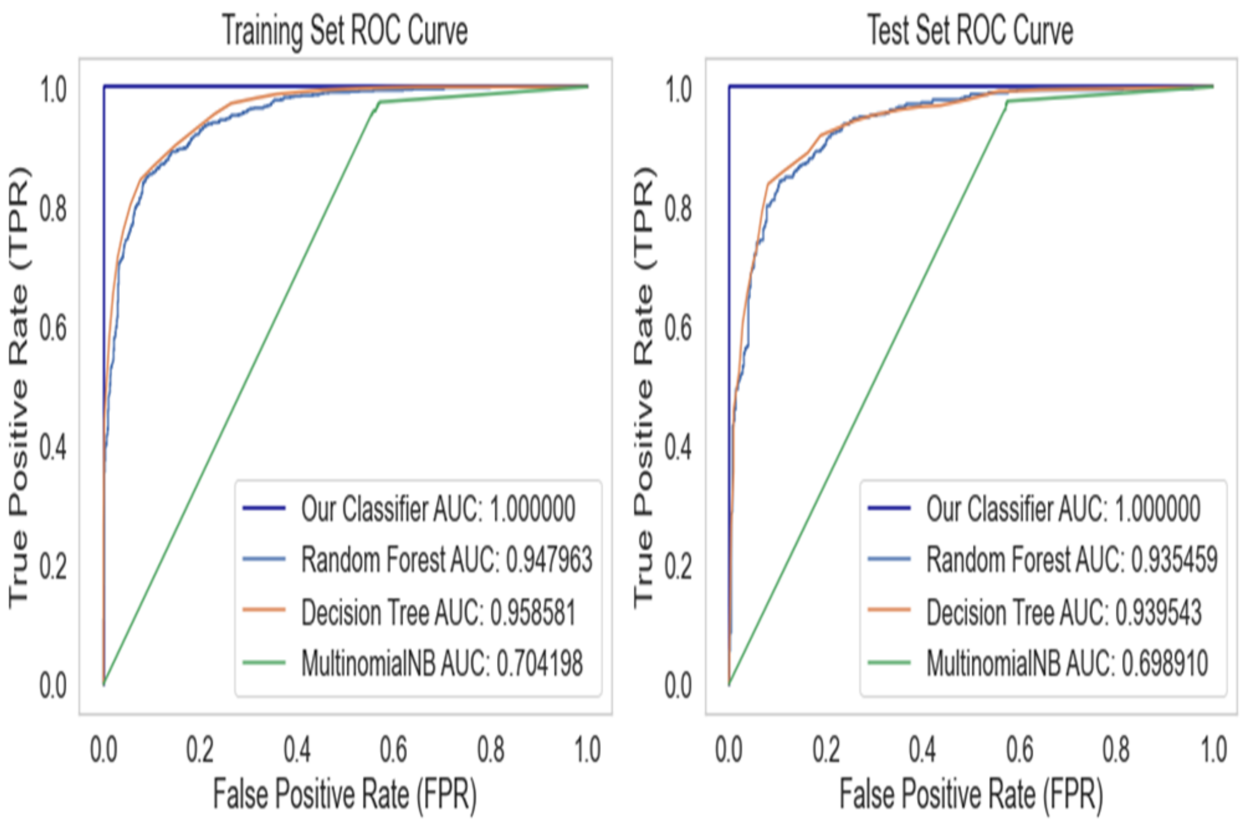


ROC Curve for Bag of words Model

The accuracy of the Bag of words Model is 96.82
Since the ROC curve is perfect, there is no overfitting or underfitting of the training and test data.

Comparison of Models

A thorough comparison of algorithms based on the metrics mentioned above gives a comprehensive insight into the performance and efficiency of each of them. Their performances as measured by ROC curves can be summarized as follows:



Graphical summarization of the performances of all the algorithms used

Tables: Comparison of Results

Model	**Accuracy score** (Training accuracy)	**Accuracy score** (Test accuracy)	**ROC** (Train)	**ROC** (Test)
Decision Tree	0.8870	0.8785	0.9585	0.9396
Naïve Bayes	0.6796	0.6976	0.7041	0.6989
Random Forest	0.8758	0.8619	0.9469	0.9326
Bag of words Model	0.9739	0.9775	1.0000	1.0000

From the above table, we observe that the results predicted by the bag of words algorithm are the most efficient, evident from the high accuracy, and ROC curve score.

Result: -

In this section, we describe the experimental results for our architecture. For training and testing, we split the dataset into two datasets, namely training and testing. The training split contains 80% records of those available in the original, while the test split contains 20% records of the original dataset. The train and test split were carried out in a stratified manner, meaning the ratio of positive samples to negative class samples in the original dataset, train, and test split is kept identical. The dataset training split consists of 29,950 records, of which 9,940 records are from bot accounts, and 20,010 data objects are from human-operated accounts. The split used to evaluate the system consists of 7,488 data items, of which 2,485 records are of bot account-controlled profiles, whereas 5,003 records are of profiles operated by humans. The experimental study is divided into three different stages, the first stage involves evaluating the performance of algorithms by applying two distinct feature encoding techniques. The second stage includes computing the performance of algorithms by employing different feature selection methods, while in the last stage, we evaluate the performance of multiple ensembling techniques. Finally, we showcase a comparison of performance between different algorithms and the best ensembling technique selected from the third stage.

Conclusion: -

In this paper, we designed a framework to detect bots on Twitter using the metadata of the Twitter profile. We carried out a comprehensive comparative study to select optimal feature encoding, feature selection, and ensembling techniques. The experimental analysis shows that the performance of the system increases on using Weight of Evidence(WoE) encoding, extra-tree classifier for feature selection, and by employing blending using Random Forest algorithm as compared to other strategies. The designed system gives an AUC of 93% and as our system works on metadata of Twitter Profile thus the speed of detection of bots on Twitter is greater than the system analyzing the accounts using behavior. However, our system relies on static analysis of Twitter Profile the efficiency of the system is less as compared to the systems that analyze the behavior of Twitter profile

Future Work

The proposed work can be extended to other social media platforms like Facebook, Instagram, YouTube, and many more. The algorithm can be run on any platform that is vulnerable to such attacks.

References: -

1.K.-C. Yang, O. Varol, C. A. Davis, E. Ferrara, A. Flammini, and F. Menczer, “Arming the public with artificial intelligence to counter social bots,” Human Behavior and Emerging Technologies, vol. 1, no. 1, pp. 48–61, 2019 Available: 10.1002/hbe2.115; https: //dx.doi.org/10.1002/hbe2.115

[1] A. Bashar, “Survey on evolving deep learning neural network architectures,” Journal of Artificial Intelligence, vol. 1, no. 02, pp. 73–82, 2019.

[2] M. Nguyen, M. Aktas, and E. Akbas, “Bot Detection on Social Networks Using Persistent Homology,” Mathematical and Computational Applications, vol. 25, no. 3, pp. 58–58, 2020. [Online]. Available: 10.3390/mca25030058;https://dx.doi.org/10.3390/mca25030058

[3] K. E. Daouadi, R. Z. Rebaï, and I. Amous, “Bot Detection on Online Social Networks Using Deep Forest,” in Computer Science On-line Conference. Springer, 2019, pp. 307–315.

[4] P. C. Lin and P. M. Huang, “A study of effective features for detecting long surviving Twitter spam accounts,” 15th International Conference on Advanced Communications Technology (ICACT), pp. 841–846, 2013.

[5] S. Kudugunta and E. Ferrara, “Deep neural networks for bot detection,” Information Sciences, vol. 467, pp. 312–322, 2018. [Online]. Available: 10.1016/j.ins.2018.08.019;https://dx.doi.org/10.1016/j.ins.2018.08.019

[6] D. Stukal, S. Sanovich, R. Bonneau, and J. A. Tucker, “Detecting Bots on Russian Political Twitter,” Big Data, vol. 5, no. 4, pp. 310–324, 2017. [Online]. Available: 10.1089/big.2017.0038;https: //dx.doi.org/10.1089/big.2017.0038

[7] J. Rodríguez-Ruiz, J. I. Mata-Sánchez, R. Monroy, O. Loyola-González, and A. López-Cuevas, “A one-class classification approach for bot detection on Twitter,” Computers & Security, vol. 91, pp. 101 715–101 715, 2020. [Online]. Available: 10.1016/j.cose.2020.101715;https: //dx.doi.org/10.1016/j.cose.2020.101715