Minor Project Report

On

"TWITTER BOT ACCOUNT DETECTION"

SUBMITTED IN PARTIAL FULFILLMENT FOR THE AWARD OF THE DEGREE

MASTERS OF COMPUTER APPLICATIONS (MCA)

(2023-2025)

IN

JAGAN INSTITUTE OF MANAGEMENT STUDIES

SECTOR – 5, ROHINI, NEW DELHI

jims
TECHNICAL CAMPUS
SECTOR-5 ROHINI, DELHI

(Affiliated to)

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

SECTOR – 16 C, DWARKA, NEW DELHI

Submitted to:
Mr. Sanjive Saxena
Associate Professor

Submitted By:
DIVYAM PARHWAL
Enrollment No.: 01214004423
MCA 1st Year (Section-A)
1st Semester

DECLARATION

I, Divyam Parhwal, student of MCA section A 1st Shift bearing Enrolment No 01214004423 hereby declare that the Seminar titled "TWITTER BOT ACCOUNT DETECTION" which is submitted by me to Mr. Sanjive Saxena, Jagan Institute of Management Studies, Sector-5 Rohini, Delhi, in partial fulfillment of requirement for the award of the degree of "Master of Computer Applications", has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition. The Author attests that permission has been obtained for the use of any copy righted material appearing in the Minor Project Report other than brief excerpts requiring only proper acknowledgement in scholarly writing and all such use is acknowledged.

Signature:

DIVYAM PARHWAL
01214004423
MCA 1st Year (Section-A)
1st Semester

CERTIFICATE

This is to certify that Divyam Parhwal, student of MCA section A 1st Shift bearing Enrolment No 01214004423 has undertaken Minor Project titled "Twitter Bot Account Detection" completed under my supervision and guidance in partial fulfilment of the requirement for the award of degree of Master of Computer Applications (MCA) as a part of the curriculum in Guru Gobind Singh Indraprastha University, New Delhi-110078.

To the best of my knowledge and belief, the data and information presented by the student in the Minor Project has not been submitted earlier.

Signature of the Faculty Guide:

Name: Mr. Sanjive Saxena

Designation: Assistant

Professor Date: 12/12/23

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Mr. Sanjive Saxena for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents & member of Jagan Institute of Management Studies for their kind co-operation and encouragement which help me in completion of this project. I would like to express my special gratitude and thanks to industry persons for giving me such attention and time. Our thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

Signature:

DIVYAM PARHWAL
01214004423

ABSTRACT

Online social media is dominating the world in several ways. In the world of social media and Internet, there is around 3.8 billion active social media users & 4.5 billion everyday internet users. The main advantage of online social media is that we can connect to people easily and communicate with them in a better way.

In the present generation, online social networks (OSNs) have become increasingly popular, people's social lives have become more associated with these sites. They use OSNs to keep in touch with each other's, share news, organize events, and even run their own e-business.

On the other hand, researchers have started to investigate efficient techniques to detect abnormal activities and fake accounts relying on accounts features, and classification algorithms. However, some of the account's exploited features have negative contribution in the final results or have no impact, also using stand-alone classification algorithms does not always reach satisfied results.

The amount of internet users is increasing by roughly 9% every year and around the half of internet traffic mostly includes of fake accounts and bots.

The project aims to use machine learning algorithms to detect fake followers on Twitter, which can be a useful tool for users who want to ensure the authenticity of their followers and improve the overall quality of their audience.

To achieve this, the project identifies a number of features which distinguish genuine and fake followers, which are then used as attributes for the machine learning algorithms. Some common indicators of fake followers include a high number of followers relative to their activity on the platform, low engagement rates, and a lack of personal information on their profiles. However, the project may also consider other factors in the detection process. It's important to note that machine learning algorithms are not always perfect and may produce false negatives and false positives. Effectiveness of the features employed and the quality of the training data both affect how accurate the algorithm is. Additionally, there are ethical implications to consider when using machine learning to detect fake followers, as it could potentially lead to the wrongful targeting of legitimate users.

TABLE OF CONTENT

S.no	Topic
1	Declaration
2	Certificate
3	Acknowledgement
4	Abstract
5	Introduction
6	Objective, Scope and Purpose of the Project
7	System Design
8	System Requirements Specifications
9	Project Coding
10	Project Implementation and Findings
11	Tools & Technologies
12	Results
13	Conclusion and Future Scope
14	References

INDEX

S.no	Topic	Page No.
1	Chapter 1: Introduction	8
2	Chapter 2: Objective, Scope and Purpose of the Project	9
3	Chapter 3: System Design	11
4	Chapter 4: System Requirements Specifications	20
5	Chapter 5: Project Coding	23
6	Chapter 6: Project Implementation and Findings	45
7	Chapter 7: Results	62
12	Chapter 8: Conclusion and Future Scope	64
13	Chapter 9: References	65

CHAPTER 1: INTRODUCTON

Twitter is a famous online social networking platform that provides a medium for people to communicate, share information, and connect with each other. However, with the increasing use of Twitter, there has been an alarming rise in the number of fake accounts, which poses a serious threat to the security and privacy of genuine users.

These fake accounts can be used to disseminate spam, malware, and propaganda, and can also be employed in cyber-attacks.

To combat this problem, various researchers have explored the development of algorithms and techniques for detecting fake accounts on Twitter. These techniques range from using graph-based features and syntactic and semantic analysis to machine learning approaches.

In this project, we aim to investigate the problem of fake account detection on Twitter using machine learning techniques. Specifically, we will explore the use of graph-based features and machine learning algorithms to build a classifier that can detect fake accounts with high accuracy. We will also compare our approach with other existing techniques and evaluate its performance on a real-world Twitter dataset.

The primary objective of the project is to provide an effective and reliable solution for detecting fake accounts on Twitter, which can help improve the overall security and privacy of Twitter users. We hope that the outcomes of this project will be useful for researchers, social media companies, and law enforcement agencies in identifying and mitigating the threat posed by fake accounts on Twitter.

CHAPTER 2: OBJECTIVE, SCOPE AND PURPOSE

2.1 OBJECTIVE OF THE PROJECT:

Our project's main objective is to build machine learning models that is able to detect bot accounts from the given data to it.

¬ To Collect twitter data, form the Kaggle.

¬ To pre-process and extract useful features from the data.

¬ To train a Machine Learning Model on these extracted features.

¬ To make our own ML model based on bag of words approach.

¬ To train own ML Model on these extracted features.

¬ To implement all the tradition model and own model.

¬ To make a comparison of the ROC curve of all the models both on the testing data as well as the training data

¬ To make a comparison chart of all the models performance on the testing data as well as the training data to know which model performed best on what criteria.

¬ To save the predictions made by own classifier in submission.csv

2.2 PURPOSE OF THE PROJECT:

This project helps in detecting the fake/bot identities in Twitter. Basically, Nowadays Online Social Networks like Twitter are used by billions of users all around the world to build network connections. The ease and accessibility of social networks have created a new era of networking. OSN users share a lot of information in the network like photos, videos, school name, college name, phone numbers, email address, home address, family relations, bank details, career details etc. So, machine learning algorithms are used to detect the fake identities in twitter.

2.3 SCOPE OF THE PROJECT:

The scope of the project is to detect fake/bot accounts in the official dataset of twitter. Traditional Machine learning techniques like Decision Tree, Random Forest and Multinomial Naïve Bayes are used and an own ML model was built using bag of words approach is used to detect bot accounts. In today's online social networks there have been a lot of problems like fake profiles, online impersonation etc. Till date, no one has come up with a feasible solution to these problems. In this project the main idea is to give a framework which automatically detects the fake profiles, so that the social life of people becomes secured and using automatic detection technique, this model makes it easier for the sites to manage the huge number of profiles which can't be done manually.

CHAPTER 3: SYSTEM DESIGN

3.1 DFD(Data Flow Diagram):

Each level of the DFD delves deeper into the specific tasks and processes involved in the overall workflow, providing a more granular understanding of the system's functionality.

3.1.1 Level-0 Data Flow Diagram(DFD):

It provides an overview of the entire process flow, including data collection, preprocessing, feature extraction, model training, evaluation, and deployment. Level-0 DFD is as follows:

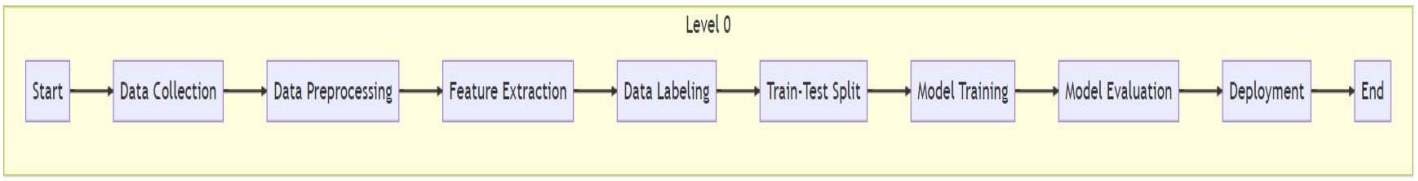


Fig 3.1.1 Level 0 Data Flow Diagram

Explanation of different components of the Level-0 DFD is as follows:

- **Start:** The starting point of the process.
- **Data Collection:** The phase where data is collected either through accessing the Twitter API or by importing datasets from Kaggle.
- **Data Preprocessing:** The process of handling missing values and performing text preprocessing on columns like description and status.
- **Feature Extraction:** The extraction of relevant features from the preprocessed data, including user-based features and text-based features.
- **Data Labelling:** The assignment of labels to the dataset indicating whether a Twitter account is a bot/fake or not.
- **Train-Test Split:** The splitting of the labelled dataset into training and testing datasets.
- **Model Training:** The training of different machine learning models using the training dataset.
- **Model Evaluation:** The evaluation of the trained models using the testing dataset.

- **Deployment:** The deployment of the models for real-time fake account detection.
- **End:** The end point of the process.

3.1.2 Level-1 Data Flow Diagram(DFD):

It break-down the main processes into more detailed steps, such as downloading datasets, handling missing values, text preprocessing, feature extraction, labelling, dataset splitting, model training, performance metrics calculation, model comparison, and integration. Level-1 DFD is as follows:

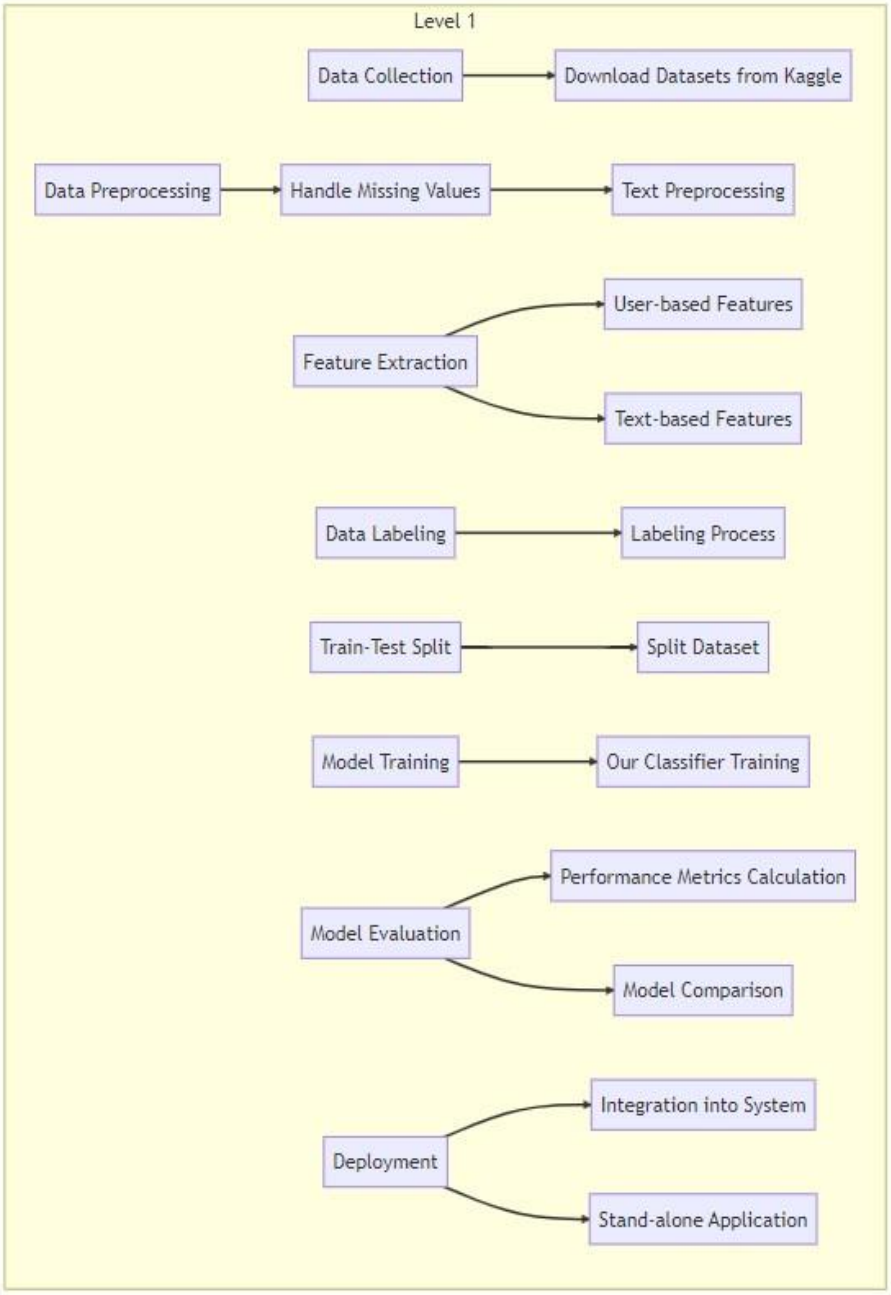


Fig 3.1.2 Level 1 Data Flow Diagram

Explanation of different components of the Level-1 DFD is as follows:

- **Download Datasets from Kaggle:** The specific step of downloading datasets from Kaggle.
- **Handle Missing Values:** The handling of missing values in the dataset.
- **Text Preprocessing:** The preprocessing of textual data to remove noise and standardize the format.
- **User-based Features:** The extraction of features derived from user profile attributes.
- **Text-based Features:** The extraction of features derived from textual data like description and status.
- **Labelling Process:** The process of assigning labels to the dataset.
- **Split Dataset:** The splitting of the dataset into training and testing datasets.
- **Our Classifier Training:** The training of the custom classifier "Our Classifier".
- **Performance Metrics Calculation:** The calculation of performance metrics like accuracy, precision, recall, and F1-score.
- **Model Comparison:** The comparison of the performance metrics of different classifiers.
- **Integration into System:** The integration of the models into a larger system.
- **Stand-alone Application:** The use of the models as standalone applications.

3.1.3 Level-2 Data Flow Diagram(DFD):

It further expands on the steps involved in text preprocessing, feature extraction, dataset splitting, and model training. It includes specific tasks like removing noise, standardizing formats, extracting user-based features, calculating account age, and applying text analysis techniques. Level-2 DFD is as follows:

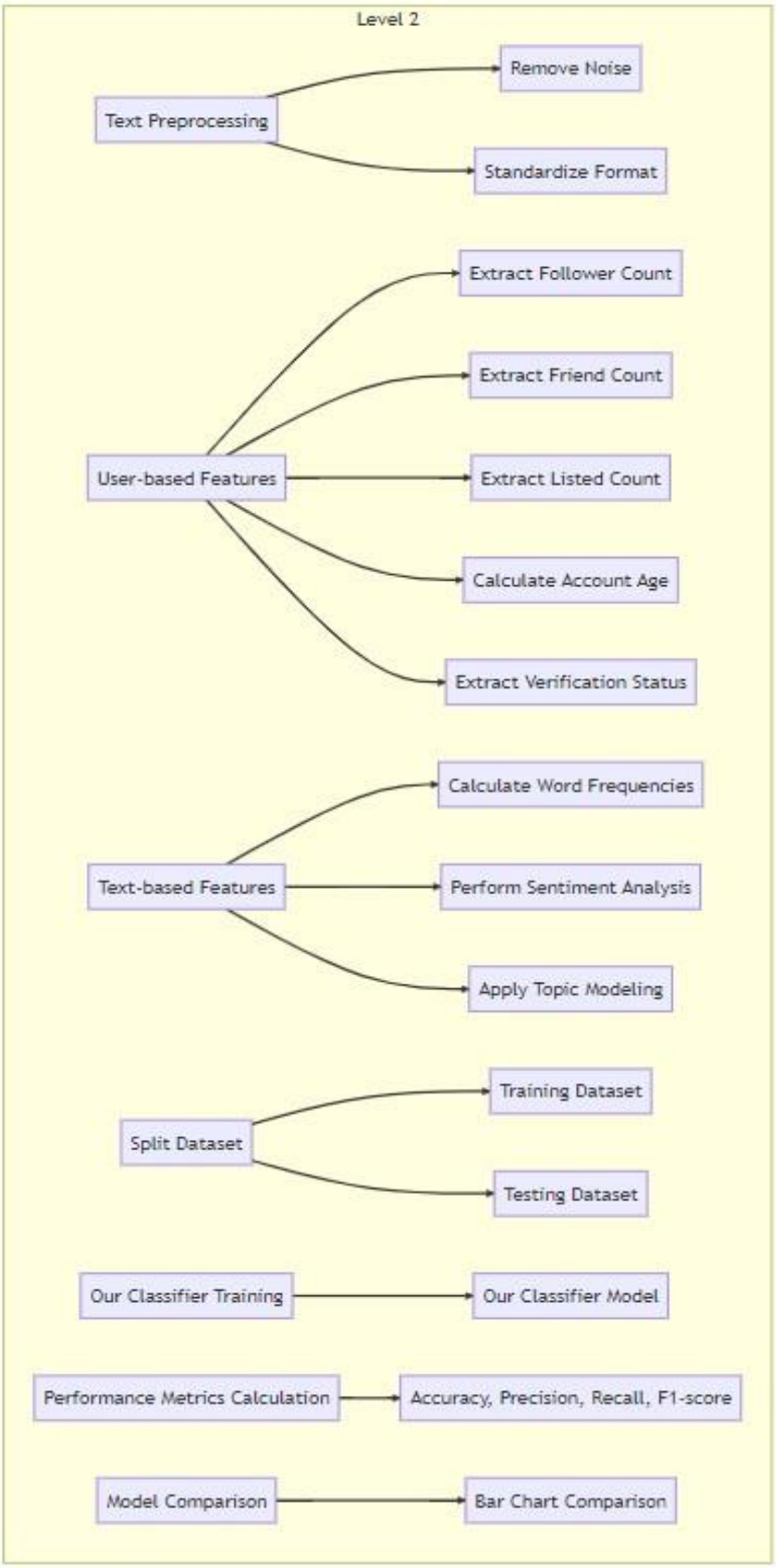


Fig 3.1.3 Level 2 Data Flow Diagram

Explanation of different components of the Level-2 DFD is as follows:

• **Remove Noise:** The step of removing noise from the textual data.

• **Standardize Format:** The step of standardizing the format of the textual data.

• **Extract Follower Count:** The extraction of the follower count feature from user profiles.

• **Extract Friend Count:** The extraction of the friend count feature from user profiles.

• **Extract Listed Count:** The extraction of the listed count feature from user profiles.

• **Calculate Account Age:** The calculation of the account age feature from the created date.

• **Extract Verification Status:** The extraction of the verification status feature from user profiles.

• **Calculate Word Frequencies:** The calculation of word frequencies from textual data.

• **Perform Sentiment Analysis:** The analysis of sentiment expressed in the textual data.

• **Apply Topic Modelling:** The application of topic modeling techniques to identify latent topics.

• **Training Dataset (X_train, y_train):** The training dataset split into feature data (X_train) and label data (y_train).

• **Testing Dataset (X_test, y_test):** The testing dataset split into feature data (X_test) and label data (y_test).

• **Our Classifier Model:** The trained model for the custom classifier "Our Classifier".

• **Accuracy, Precision, Recall, F1-score:** Performance metrics calculated for the trained models.

• **Bar Chart Comparison:** A bar chart comparing the performance metrics of different classifiers.

3.1.4 Level-3 Data Flow Diagram(DFD):

Provides additional details on the processes involved in feature extraction, such as extracting follower count, friend count, listed count, verification status, and performing text analysis tasks like sentiment analysis and topic modelling.

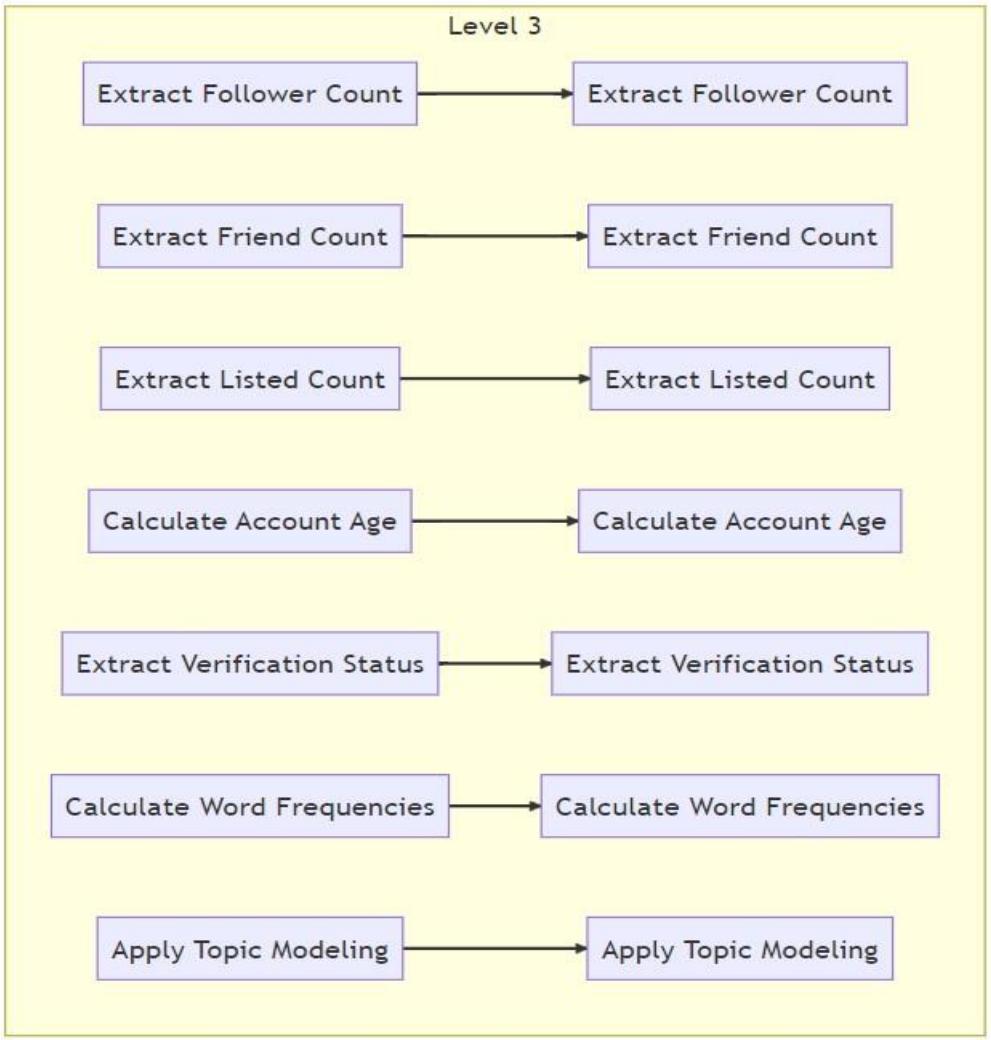


Fig 3.2.4 Level 3 Data Flow Diagram

Explanation of different components of the Level-3 DFD is as follows:

- **Extract Follower Count:** Further details on the extraction of the follower count feature.
- **Extract Friend Count:** Further details on the extraction of the friend count feature.
- **Extract Listed Count:** Further details on the extraction of the listed count feature.
- **Calculate Account Age:** Further details on the calculation of the account age feature.
- **Extract Verification Status:** Further details on the extraction of the verification status feature.
- **Calculate Word Frequencies:** Further details on the calculation of word frequencies.
- **Apply Topic Modelling:** Further details on the application of topic modelling techniques.

3.1.5 Combined Data Flow Diagram(DFD)

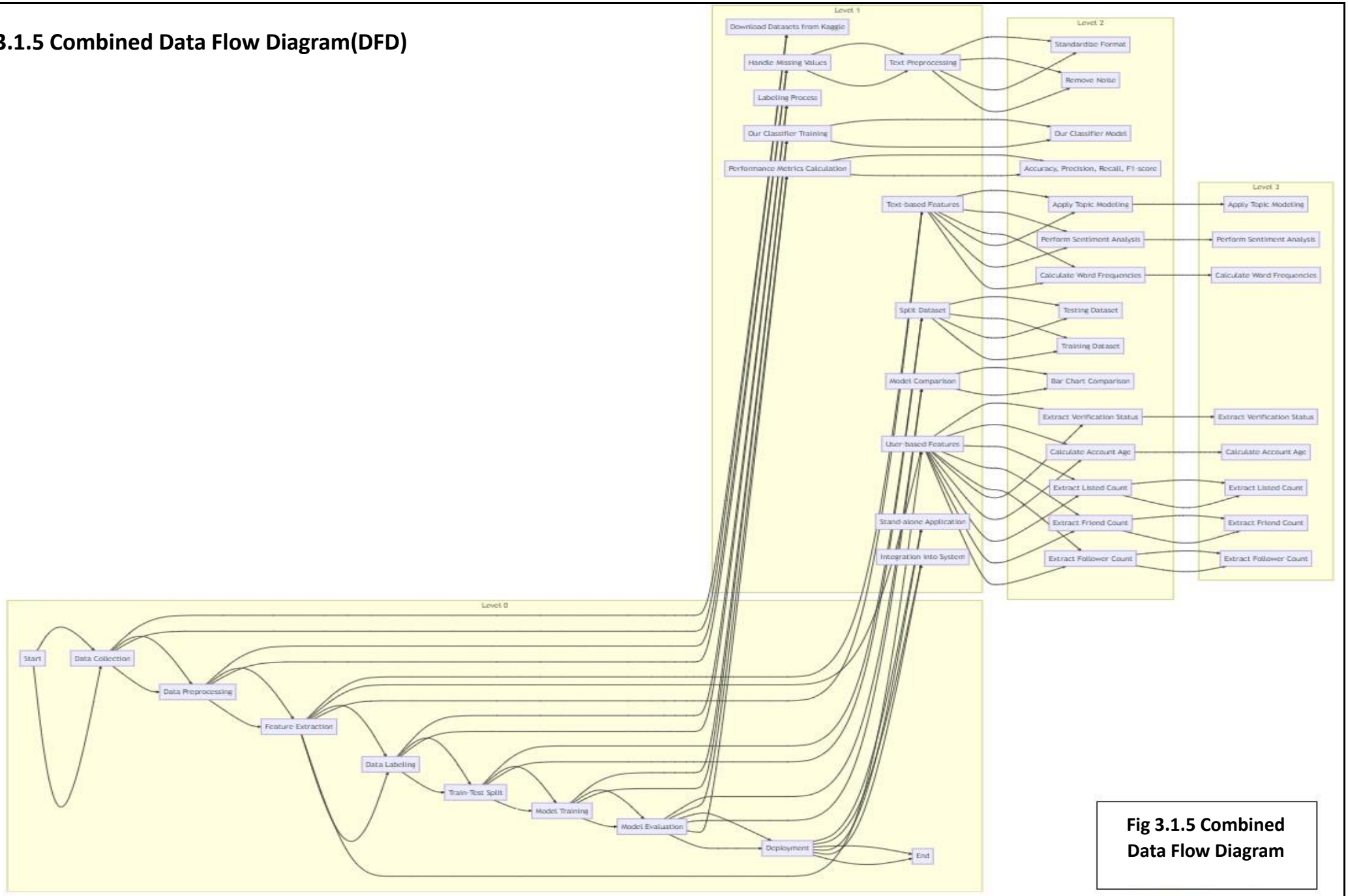


Fig 3.1.5 Combined Data Flow Diagram

3.2 System Diagram

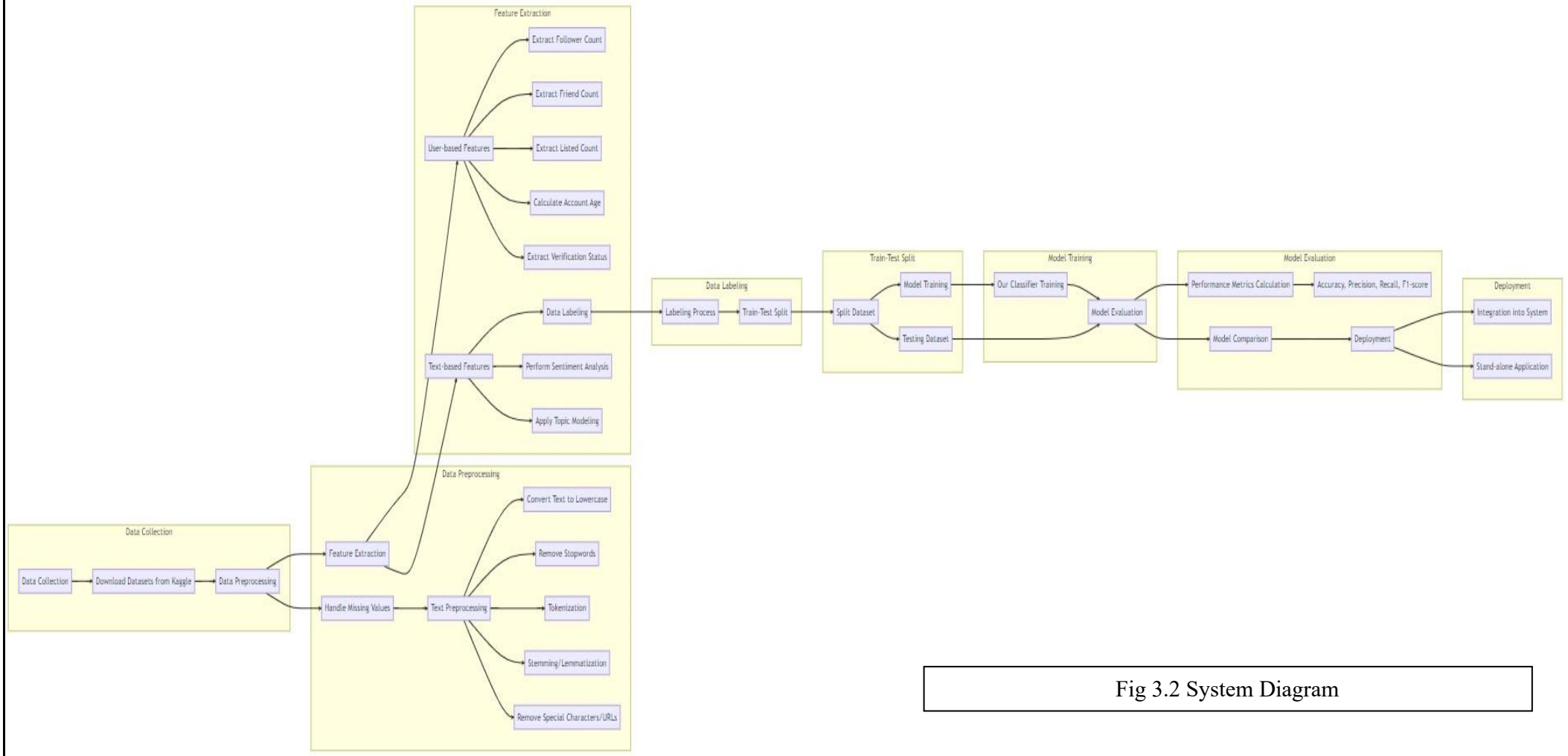


Fig 3.2 System Diagram

The data set was taken from Kaggle platform and using various machine learning algorithms and training data we have trained our classifier model. Test set is used on classifier model for giving prediction according to the given scenario.

3.3 Activity Flow Diagram:

The activity flow diagram outlines the sequential steps involved in Twitter fake account detection. It begins with data collection from Kaggle, followed by preprocessing, feature extraction, and labelling. The dataset is split into training and testing sets for model training and evaluation. Performance metrics are calculated, and models are compared. Finally, the trained models can be integrated into a system or used independently for real-time fake account detection.

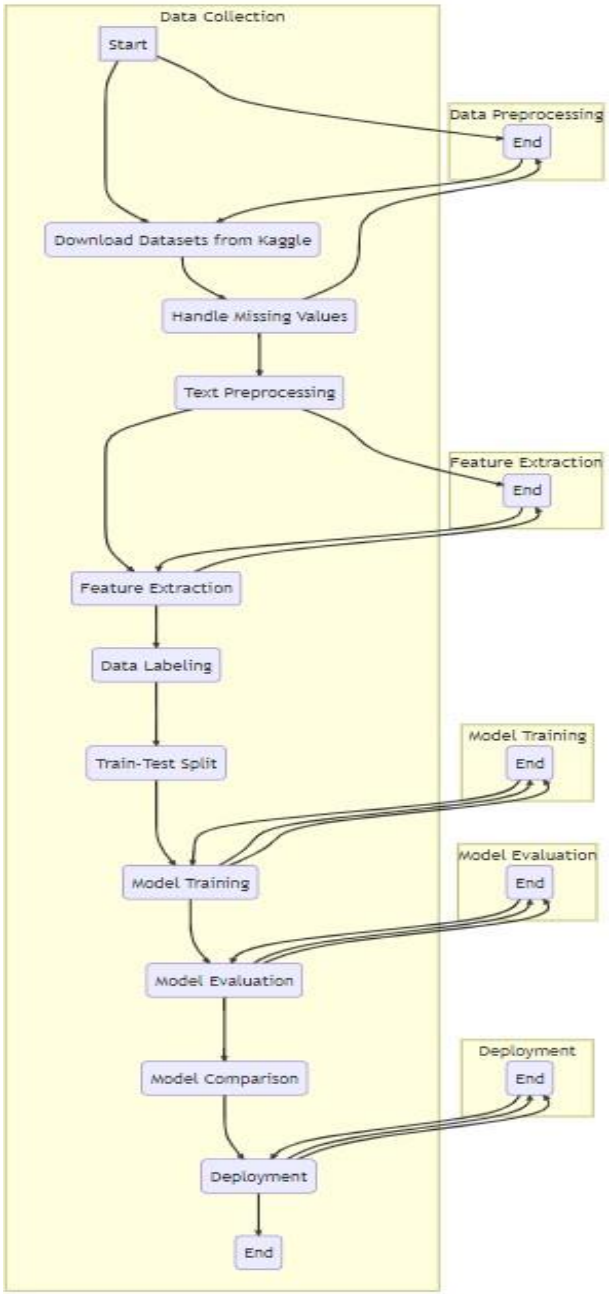


Fig 3.3 Activity Flow Diagram

CHAPTER 4: SYSTEM REQUIREMENTS SEPCIFICATIONS

This project is developed using Python as development tool. Python version 3.6 is used for this purpose. The project dataset is downloaded from Kaggle and kept in the working system. For development environment, 'Jupyter Notebook' is installed. Jupyter Notebook is a Powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts.
It offers a unique combination of the advanced editing, analysis, debugging, and profiling functionalities of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

4.1 System Configuration:

4.1.1 Hardware Requirements:

- Processor: Intel i5 2.4 GegaHertz or similar processor
- RAM: 4 GB or more
- Hard Disk: 1 TB or more
- Solid State Drive (SSD): A SSD will make the project code run much faster

4.1.2 Software Requirements:

- Operating System: Windows 7 or above
- Programming Language: Python
- Software: Anaconda

4.2 Python 3:

Python is one of the most common languages used for machine learning because it is easy to learn, enormous packages it supports like scikit learn that contains the implementation of common machine learning algorithms, built-in library supports, pre-processed models, support for larger networks and huge toolset.

4.3 Benefits of Python:

• Scikit learn- It features various classification, regression and clustering algorithms including support vector machines, random forest, gradient boosting, K-means and DBSCAN, and is designed to interoperate with the python numerical and scientific libraries Numpy and SciPy.
• Data frame object for data manipulation with integrated indexing.
• Tools for reading and writing data between in memory data structures and different file formats.
• Data alignment and integrated handling of missing data.
• Reshaping and pivoting of datasets.
• Label based slicing, fancy indexing and sub setting of large dataset.
• Data structure column insertion and deletion.
• Group by engine allowing split-apply-combine operations on dataset.
• Data set merging and joining.
• Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.

4.4 Anaconda Software:

Anaconda is a Python distribution that includes installations and packages managements tools. It offers a wide selection of packages and commercial support [20]. Additionally, Anaconda is an environment manager that allows users to create different Python environments, each without its own settings. This feature enables developers to work on multiple projects with different dependencies and configurations without conflicts [20]. Anaconda's package management tools provide access to over 7,500 open-source packages for data science, machine learning, and AI. It includes packages such as NumPy, pandas, and scikit-learn, making it an excellent tool for data science projects. Furthermore, Anaconda provides commercial support for its enterprise users, allowing businesses to use Anaconda with the assurance of dedicated support. Overall, Anaconda is a powerful tool for Python developers, especially for those working on data science and machine learning projects. Its package management tools and environment management features provide flexibility and ease of use for working on multiple projects. Anaconda can help with:

• Python can be installed over the multiple platforms
• Different environments can be supported separately
• Distributing with not having correct privileges and

• Support and running with specific packages and libraries

4.5. Kaggle:

Kaggle is a data science competition platform and online community of data scientists and machine learning practitioners under Google LLC. Kaggle enables users to find and publish datasets, explore and build models in a web-based data science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges. Kaggle platform is in our project to get the dataset on the help of which project is implemented.

CHAPTER 5: PROJECT CODING

Source Code:

5.1 Importing the dataset & libraries & overviewing the dataset:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
import warnings
import time
from sklearn import metrics
mpl.rcParams['patch.force_edgecolor'] = True
warnings.filterwarnings("ignore")
%matplotlib inline
file= 'training_data.csv'
training_data = pd.read_csv(file)
training_data.head()
# printing all columns
count = 1
for col in training_data.columns:
  print(f'{count} -> {col}')
  count += 1
training_data.info()
# Statistical details of the data
training_data.describe()
# Returns True if there are any NaN values and Returns False otherwise
training_data.isnull().values.any()
# Total Number of NaN values
training_data.isnull().sum().sum()
bots = training_data[training_data.bot==1]
nonbots = training_data[training_data.bot==0]
```

```
#### Number of Real Accounts vs Number of Fake Accounts
# value counts of humans and bots
human_bots = training_data.bot
human_bots.value_counts()
count_real = human_bots.value_counts()[0]
count_fake = human_bots.value_counts()[1]
plt.figure(figsize=(6, 6))
sns.barplot(x = ['Real Account', 'Fake Account'], y = [count_real, count_fake], palette =
sns.color_palette('magma'))
plt.title("Number of Entries by Account Type", fontname="Times New Roman", fontsize=21,
fontweight="bold")
sns.despine()
```

5.2 Exploratory Data Analysis:-

5.2.1 Data cleaning:

```
# Column wise NaN value count
training_data.isnull().sum()
def get_heatmap(df):
    #This function gives heatmap of all NaN values
    plt.figure(figsize=(29,21))
    sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='plasma')
    sns.set(font_scale=15.0)
    plt.title('Heatmap of all NaN values', fontname="Times New Roman", fontsize=50,
fontweight="bold")
    return plt.show()
get_heatmap(training_data)
sns.set(font_scale=1)
# Returns list of columns with NaN values
training_data.columns[training_data.isna().any()].tolist()
#### ✈ We won't be using 'description',  'lang','profile_background_image_url',
'profile_image_url' in our analysis. So, we can directly convert the NaN to string directly.
#### ✈ But for 'location' -> NaN values can be changed to 'unknown'
```

```
# checking the dataframe for values for 'location' as NaN
training_data[pd.isnull(training_data['location'])]
# Most accounts have not disclosed their location. So, unknown is the mode (i.e. the account didn't disclose it for privacy reasons)
training_data['location'].value_counts().keys()[0]
# mode of column 'location' == 'unknown'
training_data['location'].mode()
# we replace NaN values with mode of the column 'location'
training_data.fillna({'location': training_data['location'].mode()}, inplace=True)
```

5.2.2 Finding the general characteristics of real & fake accounts:

```
bots.friends_count/bots.followers_count
plt.figure(figsize=(10,5))
plt.subplot(2,1,1)
plt.title('Fake Accounts Friends vs Followers', fontname="Times New Roman", fontsize=24, fontweight="bold")
sns.regplot(bots.friends_count, bots.followers_count, color='purple', label='Fake Accounts')
plt.xlim(0, 100)
plt.ylim(0, 100)
plt.tight_layout()
plt.subplot(2,1,2)
plt.title('Real Accounts Friends vs Followers', fontname="Times New Roman", fontsize=24, fontweight="bold")
sns.regplot(nonbots.friends_count, nonbots.followers_count, color='blue', label='Real Accounts')
plt.xlim(0, 100)
plt.ylim(0, 100)
plt.tight_layout()
plt.show()
```

5.2.3 Identifying the Imbalance in the Data:

```
#### 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='purple', label='Fake Accounts')
plt.plot(nonbots.listed_count, color='blue', label='Real Accounts')
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='purple', label='Fake Accounts Friends')
plt.plot(nonbots_listed_count_df.friends_count, color='blue', label='Real Accounts Friends')
plt.legend(loc='upper left')
plt.subplot(2,1,2)
plt.plot(bots_listed_count_df.followers_count, color='purple', label='Fake Accounts Followers')
plt.plot(nonbots_listed_count_df.followers_count, color='blue', label='Real Accounts
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
```

5.2.4 Feature Independence using Spearman Correlation:

```
### Feature Independence using Spearman correlation
df.corr(method='spearman')
plt.figure(figsize=(14,12))
sns.heatmap(df.corr(method='spearman'), cmap='RdBu_r', 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.
```

5.2.5 Feature Engineering:

```
#### Performing Feature Engineering
file= open('training_data.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)
```

5.2.6 Feature Extraction:

```
#### 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']
```

5.3 Implementing Different Models:

```
## Implementing Different Models
```

5.3.1 Decisions Trees Classifier:

```
## Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_curve, auc, confusion_matrix, precision_score,
recall_score, f1_score
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)

# Confusion matrix
train_cm = confusion_matrix(y_train, y_pred_train)
test_cm = confusion_matrix(y_test, y_pred_test)
print("Train Confusion Matrix:\n", train_cm,"\n")
print("Test Confusion Matrix:\n", test_cm,"\n")

# Getting performance metrics
train_precision_decisiontree = precision_score(y_train, y_pred_train)
test_precision_decisiontree = precision_score(y_test, y_pred_test)
train_recall_decisiontree = recall_score(y_train, y_pred_train)
test_recall_decisiontree = recall_score(y_test, y_pred_test)
train_f1_score_decisiontree = f1_score(y_train, y_pred_train)
test_f1_score_decisiontree = f1_score(y_test, y_pred_test)

# Print performance metrics
print(f'Train precision: {train_precision_decisiontree:.5f}')
print(f'Test precision: {test_precision_decisiontree:.5f}\n')
print(f'Train recall: {train_recall_decisiontree:.5f}')
print(f'Test recall: {test_recall_decisiontree:.5f}\n')
print(f'Train F1-score: {train_f1_score_decisiontree:.5f}')
print(f'Test F1-score: {test_f1_score_decisiontree:.5f}\n')

# Getting and Printing the train and test accuracy of the classifier
train_accuracy_decisiontree = accuracy_score(y_train, y_pred_train)
test_accuracy_decisiontree = accuracy_score(y_test, y_pred_test)
print("Trainig Accuracy: %.5f" %train_accuracy_decisiontree)
print("Test Accuracy: %.5f" %test_accuracy_decisiontree)
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='black', 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 0.93, so we will try other models.
```

5.3.2 Multinomials Naïve Bayes Classifiers:

```
## 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)
```

```
# Confusion matrix
train_cm = confusion_matrix(y_train, y_pred_train)
test_cm = confusion_matrix(y_test, y_pred_test)
print("Train Confusion Matrix:\n", train_cm,"\n")
print("Test Confusion Matrix:\n", test_cm,"\n")

# Getting performance metrics
train_precision_naive_bayes = precision_score(y_train, y_pred_train)
test_precision_naive_bayes = precision_score(y_test, y_pred_test)
train_recall_naive_bayes = recall_score(y_train, y_pred_train)
test_recall_naive_bayes = recall_score(y_test, y_pred_test)
train_f1_score_naive_bayes = f1_score(y_train, y_pred_train)
test_f1_score_naive_bayes = f1_score(y_test, y_pred_test)

# Print performance metrics
print(f'Train precision: {train_precision_naive_bayes:.5f}')
print(f'Test precision: {test_precision_naive_bayes:.5f}\n')
print(f'Train recall: {train_recall_naive_bayes:.5f}')
print(f'Test recall: {test_recall_naive_bayes:.5f}\n')
print(f'Train F1-score: {train_f1_score_naive_bayes:.5f}')
print(f'Test F1-score: {test_f1_score_naive_bayes:.5f}\n')

# Getting and Printing the train and test accuracy of the classifier
train_accuracy_naive_bayes = accuracy_score(y_train, y_pred_train)
test_accuracy_naive_bayes = accuracy_score(y_test, y_pred_test)
print("Trainig Accuracy: %.5f" %train_accuracy_naive_bayes)
print("Test Accuracy: %.5f" %test_accuracy_naive_bayes)
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 Naive Bayes 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.
```

5.3.3 Random Forest Classifier:

```
# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix

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)

# Confusion matrix
train_cm = confusion_matrix(y_train, y_pred_train)
test_cm = confusion_matrix(y_test, y_pred_test)
print("Train Confusion Matrix:\n", train_cm,"\n")
print("Test Confusion Matrix:\n", test_cm,"\n")

# Print performance metrics
train_precision_random_forest = precision_score(y_train, y_pred_train)
test_precision_random_forest = precision_score(y_test, y_pred_test)
train_recall_random_forest = recall_score(y_train, y_pred_train)
test_recall_random_forest = recall_score(y_test, y_pred_test)
train_f1_score_random_forest = f1_score(y_train, y_pred_train)
test_f1_score_random_forest = f1_score(y_test, y_pred_test)

# Print performance metrics
print(f'Train precision: {train_precision_random_forest:.5f}')
print(f'Test precision: {test_precision_random_forest:.5f}\n')
print(f'Train recall: {train_recall_random_forest:.5f}')
print(f'Test recall: {test_recall_random_forest:.5f}\n')
print(f'Train F1-score: {train_f1_score_random_forest:.5f}')
print(f'Test F1-score: {test_f1_score_random_forest:.5f}\n')

# Getting and Printing the train and test accuracy of the classifier
train_accuracy_random_forest = accuracy_score(y_train, y_pred_train)
test_accuracy_random_forest = accuracy_score(y_test, y_pred_test)
print("Trainig Accuracy: %.5f" %train_accuracy_random_forest)
print("Test Accuracy: %.5f" %test_accuracy_random_forest)
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')
```

5.3.4 Our Proposed Classifier:

```
# Own 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.iloc[:,-1]
        X_test, y_test = test, test.iloc[:, -1]
        return (X_train, y_train, X_test, y_test)
```

```
    def get_heatmap(df):
        # This function gives heatmap of all NaN values
        plt.figure(figsize=(29,21))
        sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='plasma')
        plt.tight_layout()
        return plt.show()

    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|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'
        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_confusion_matrix(df):
    (X_train, y_train, X_test, y_test) = twitter_bot.perform_train_test_split(df)
    y_pred_train, y_true = twitter_bot.get_predicted_and_true_values(X_train, y_train)
    y_pred_test, y_true = twitter_bot.get_predicted_and_true_values(X_test, y_test)
    train_cm = confusion_matrix(y_train, y_pred_train)
    test_cm = confusion_matrix(y_test, y_pred_test)
    print("Train Confusion Matrix:\n", train_cm,"\n")
    print("Test Confusion Matrix:\n", test_cm,"\n")

def get_performance_metrics(df):
    (X_train, y_train, X_test, y_test) = twitter_bot.perform_train_test_split(df)
    y_pred_train, y_true = twitter_bot.get_predicted_and_true_values(X_train, y_train)
    y_pred_test, y_true = twitter_bot.get_predicted_and_true_values(X_test, y_test)

    # Getting performance metrics
    train_precision = precision_score(y_train, y_pred_train)
    test_precision = precision_score(y_test, y_pred_test)
    train_recall = recall_score(y_train, y_pred_train)
```

```
        test_recall = recall_score(y_test, y_pred_test)
        train_f1_score = f1_score(y_train, y_pred_train)
        test_f1_score = f1_score(y_test, y_pred_test)
        return train_precision, test_precision, train_recall, test_recall, train_f1_score, test_f1_score

    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):
        sns.set(font_scale=1.5)
        sns.set_style("whitegrid", {'axes.grid': False})
        (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()
    print("Training the Bot detection classifier. Please wait a few seconds.\n")
    train_df = pd.read_csv('training_data.csv')
    test_df = pd.read_csv('test_data.csv', sep='\t', encoding='ISO-8859-1')
    twitter_bot.get_confusion_matrix(train_df)
    train_precision, test_precision, train_recall, test_recall, train_f1_score,
test_f1_score=twitter_bot.get_performance_metrics(train_df)
    print(f'Train precision: {train_precision:.5f}')
    print(f'Test precision: {test_precision:.5f}\n')
    print(f'Train recall: {train_recall:.5f}')
    print(f'Test recall: {test_recall:.5f}\n')
    print(f'Train F1-score: {train_f1_score:.5f}')
    print(f'Test F1-score: {test_f1_score:.5f}\n')
    train_accuracy=twitter_bot.get_accuracy_score(train_df)[0]
    test_accuracy=twitter_bot.get_accuracy_score(train_df)[1]
    print("Train Accuracy: ", train_accuracy)
    print("Test Accuracy: ", test_accuracy, "\n")

    #predicting test data results
    predicted_df = twitter_bot.bot_prediction_algorithm(test_df)
    # preparing subission file
    predicted_df.to_csv('submission.csv', index=False)
    print("Predicted results are saved to submission.csv. File shape:
{}".format(predicted_df.shape))
    #ssss: " ,twitter_bot.get_bot_predictions(test_df))
    #plotting the ROC curve
    twitter_bot.plot_roc_curve(train_df)
    print("Time duration: {} seconds.".format(time.time()-start))
```

5.4 Comparing all the Models:

5.4.1 ROC Curve of all the models combined:

```
## ROC Comparison after tuning the baseline model
plt.figure(figsize=(21,16))
(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)

#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='Multinomial Naive Bayes AUC: %5f'
%auc(fpr_mnb_train, tpr_mnb_train))
plt.title("Training Set ROC Curve", fontname="Times New Roman", fontsize=29,
fontweight="bold")
plt.xlabel("False Positive Rate (FPR)", fontname="Times New Roman", fontsize=19,
fontweight="bold")
plt.ylabel("True Positive Rate (TPR)", fontname="Times New Roman", fontsize=19,
fontweight="bold")
plt.legend(loc='lower right')
plt.figure(figsize=(21,16))
(X_train, y_train, X_test, y_test) = twitter_bot.perform_train_test_split(df)

#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)

#Test ROC
plt.subplot(2,2,1)
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='Multinomial Naive Bayes AUC: %5f'
%auc(fpr_mnb_test, tpr_mnb_test))
plt.title("Test Set ROC Curve", fontname="Times New Roman", fontsize=29,
fontweight="bold")
plt.xlabel("False Positive Rate (FPR)", fontname="Times New Roman", fontsize=19,
fontweight="bold")
plt.ylabel("True Positive Rate (TPR)", fontname="Times New Roman", fontsize=19,
fontweight="bold")
plt.legend(loc='lower right')
```

5.4.2 Comparison Charts:

```
# COMPARISON CHART OF ALL CLASSIFIERS
# set width of bar
barWidth = 0.2
fig = plt.subplots(figsize =(50,26))

# set height of bar
Our_Classifier = [train_accuracy, train_precision, train_recall, train_f1_score]
Random_Forest = [train_accuracy_random_forest, train_precision_random_forest,
train_recall_random_forest, train_f1_score_random_forest]
Decision_Tree = [train_accuracy_decisiontree, train_precision_decisiontree,
train_recall_decisiontree, train_f1_score_decisiontree]
Naive_Bayes = [train_accuracy_naive_bayes, train_precision_naive_bayes,
train_recall_naive_bayes, train_f1_score_naive_bayes]
```

```
# Set position of bar on X axis
br1 = np.arange(len(Our_Classifier))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]

# Make the plot
plt.bar(br1, Our_Classifier, color ='r', width = barWidth, edgecolor ='grey', label ='Our Classifier')
plt.bar(br2, Random_Forest, color ='g', width = barWidth, edgecolor ='grey', label ='Random Forest')
plt.bar(br3, Decision_Tree, color ='b', width = barWidth, edgecolor ='grey', label ='Decision Tree')
plt.bar(br4, Naive_Bayes, color ='k', width = barWidth, edgecolor ='grey', label ='Mutinomial Naive Bayes')

# Adding x-axis and y-axis labels, xticks, yticks and title
plt.xlabel('\nEvaluation Metrics', fontname="Times New Roman", fontsize=88, fontweight="bold")
plt.ylabel('Score\n', fontname="Times New Roman", fontsize=88, fontweight="bold")
plt.title('Training Performance Metrics Comparison of All the Models\n', fontname="Times New Roman", fontsize=111, fontweight="bold")
plt.yticks(fontsize=44, fontweight="bold")
plt.xticks([r + barWidth for r in range(len(Our_Classifier))], ['Accuracy', 'Precision', 'Recall', 'f1_score'], fontname="Times New Roman", fontsize=46, fontweight="bold")

# Adding value labels above each bar
for i, v in enumerate(Our_Classifier):
    plt.text(r1[i], v + 0.01, f'{v:.3f}', color='black', ha='center', fontname="Times New Roman", fontsize=40, fontweight="bold")
for i, v in enumerate(Random_Forest):
    plt.text(r2[i], v + 0.01, f'{v:.3f}', color='black', ha='center', fontname="Times New Roman", fontsize=40, fontweight="bold")
for i, v in enumerate(Decision_Tree):
```

```
    plt.text(r3[i], v + 0.01, f'{v:.3f}', color='black', ha='center', fontname="Times New Roman",
fontsize=40, fontweight="bold")
for i, v in enumerate(Naive_Bayes):
    plt.text(r4[i], v + 0.01, f'{v:.3f}', color='black', ha='center', fontname="Times New Roman",
fontsize=40, fontweight="bold")

# Adding legend
plt.legend(loc='upper right', fontsize=36)

# Adjust the spacing between the legend and bars
plt.subplots_adjust(top=2.1)

# Display the chart

plt.show()
# set width of bar
barWidth = 0.2
fig = plt.subplots(figsize =(50,26))
# set height of bar
Our_Classifier = [test_accuracy, test_precision, test_recall, test_f1_score]
Random_Forest = [test_accuracy_random_forest, test_precision_random_forest,
test_recall_random_forest, test_f1_score_random_forest]
Decision_Tree = [test_accuracy_decisiontree, test_precision_decisiontree,
test_recall_decisiontree, test_f1_score_decisiontree]
Naive_Bayes = [test_accuracy_naive_bayes, test_precision_naive_bayes,
test_recall_naive_bayes, test_f1_score_naive_bayes]
# Set position of bar on X axis
br1 = np.arange(len(Our_Classifier))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]
# Make the plot
plt.bar(br1, Our_Classifier, color ='r', width = barWidth, edgecolor ='grey', label ='Our
Classifier')
```

```
plt.bar(br2, Random_Forest, color ='g', width = barWidth, edgecolor ='grey', label ='Random Forest')
plt.bar(br3, Decision_Tree, color ='b', width = barWidth, edgecolor ='grey', label ='Decision Tree')
plt.bar(br4, Naive_Bayes, color ='k', width = barWidth, edgecolor ='grey', label ='Mutinomial Naive Bayes')
# Adding x-axis and y-axis labels, xticks, yticks and title
plt.xlabel('\nEvaluation Metrics', fontname="Times New Roman", fontsize=88, fontweight="bold")
plt.ylabel('Score\n', fontname="Times New Roman", fontsize=88, fontweight="bold")
plt.title('Testing Performance Metrics Comparison of All the Models\n', fontname="Times New Roman", fontsize=111, fontweight="bold")
plt.yticks(fontsize=44, fontweight="bold")
plt.xticks([r + barWidth for r in range(len(Our_Classifier))], ['Accuracy', 'Precision', 'Recall', 'f1_score'], fontname="Times New Roman", fontsize=46, fontweight="bold")
# Adding value labels above each bar
for i, v in enumerate(Our_Classifier):
    plt.text(r1[i], v + 0.01, f'{v:.3f}', color='black', ha='center', fontname="Times New Roman", fontsize=40, fontweight="bold")
for i, v in enumerate(Random_Forest):
    plt.text(r2[i], v + 0.01, f'{v:.3f}', color='black', ha='center', fontname="Times New Roman", fontsize=40, fontweight="bold")
for i, v in enumerate(Decision_Tree):
    plt.text(r3[i], v + 0.01, f'{v:.3f}', color='black', ha='center', fontname="Times New Roman", fontsize=40, fontweight="bold")
for i, v in enumerate(Naive_Bayes):
    plt.text(r4[i], v + 0.01, f'{v:.3f}', color='black', ha='center', fontname="Times New Roman", fontsize=40, fontweight="bold")
# Adding legend
plt.legend(loc='upper right', fontsize=36)
# Adjust the spacing between the legend and bars
plt.subplots_adjust(top=2.1)
# Display the chart
plt.show()
```

CHAPTER 6: PROJECT IMPLEMENTATION AND FINDINGS

6.1 Importing the dataset & libraries & overviewing the dataset:

The project starts by importing of all the libraries which functions will be used for cleaning of the data, data analysis & visualisation and making our own classifier. Then dataset used for the project is been loaded after those dimensions and datatype of the dataset is been known with the help of pandas library and also statistical information of the dataset is known by writing code for it this was all done to have a complete overview of the dataset before doing the data cleaning. Below figure 1 is showing the datatype of the dataset and figure 2 shows statistical information of the dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2797 entries, 0 to 2796
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     2797 non-null   float64
 1   id_str                 2797 non-null   object
 2   screen_name            2797 non-null   object
 3   location               1777 non-null   object
 4   description            2394 non-null   object
 5   url                    1455 non-null   object
 6   followers_count        2797 non-null   int64
 7   friends_count          2797 non-null   int64
 8   listed_count           2797 non-null   int64
 9   created_at             2797 non-null   object
 10  favourites_count       2797 non-null   int64
 11  verified               2797 non-null   bool
 12  statuses_count         2797 non-null   int64
 13  lang                   2797 non-null   object
 14  status                 2508 non-null   object
 15  default_profile        2797 non-null   bool
 16  default_profile_image  2797 non-null   bool
 17  has_extended_profile   2698 non-null   object
 18  name                   2797 non-null   object
 19  bot                    2797 non-null   int64
dtypes: bool(3), float64(1), int64(6), object(10)
memory usage: 379.8+ KB
```

Figure1: Datatypes of the dataset

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)
count	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.0	96453.000000
mean	11.932678	10.855029	0.734899	10.810640	187.509232	10.347325	0.0	1003.235956
std	9.551546	10.696847	0.195473	6.913571	107.383428	4.192123	0.0	116.969906
min	-21.822222	-27.716667	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	4.688889	2.311111	0.600000	5.828200	116.000000	8.339800	0.0	1011.900000
50%	12.000000	12.000000	0.780000	9.965900	180.000000	10.046400	0.0	1016.450000
75%	18.838889	18.838889	0.890000	14.135800	290.000000	14.812000	0.0	1021.090000
max	39.905556	39.344444	1.000000	63.852600	359.000000	16.100000	0.0	1046.380000

Figure2: Statistical details of the dataframe

6.2 Exploratory Data Analysis:-

6.2.1 Data cleaning:

Data cleaning is the method of removing all the inaccuracies and inconsistencies from the dataset. So, in this project the data cleaning process of data starts by firstly checking for missing values present within the data. This was done because if during analysis any missing value is found then visualisation of it will not be possible and also because if we trained your models with the inconsistent data then it will have poor performance and less accuracy. For checking the null values, a heatmap was made which shows the null values of the data different attributes with different color. Then isnull().sum() function is used to know the null values if any present in the dataset. The picture of all null values present in dataset as per column name is given in the below figure1 and heatmap showing null values in different attributes of training data is given in the below figure2.

```
id                          0
id_str                      0
screen_name                 0
location                 1020
description               403
url                      1342
followers_count             0
friends_count               0
listed_count                0
created_at                  0
favourites_count            0
verified                    0
statuses_count              0
lang                        0
status                    289
default_profile             0
default_profile_image       0
has_extended_profile       99
name                        0
bot                         0
dtype: int64
```

figure1

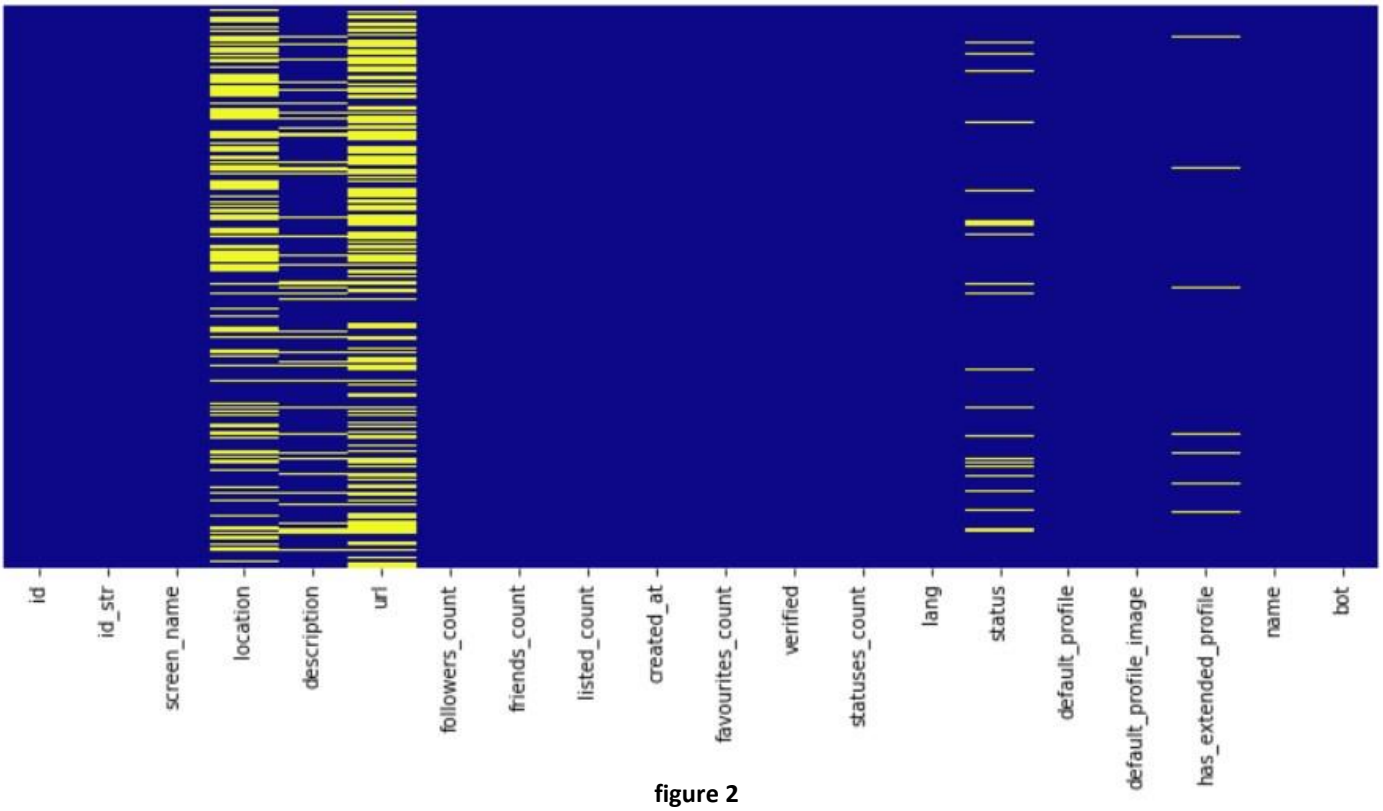
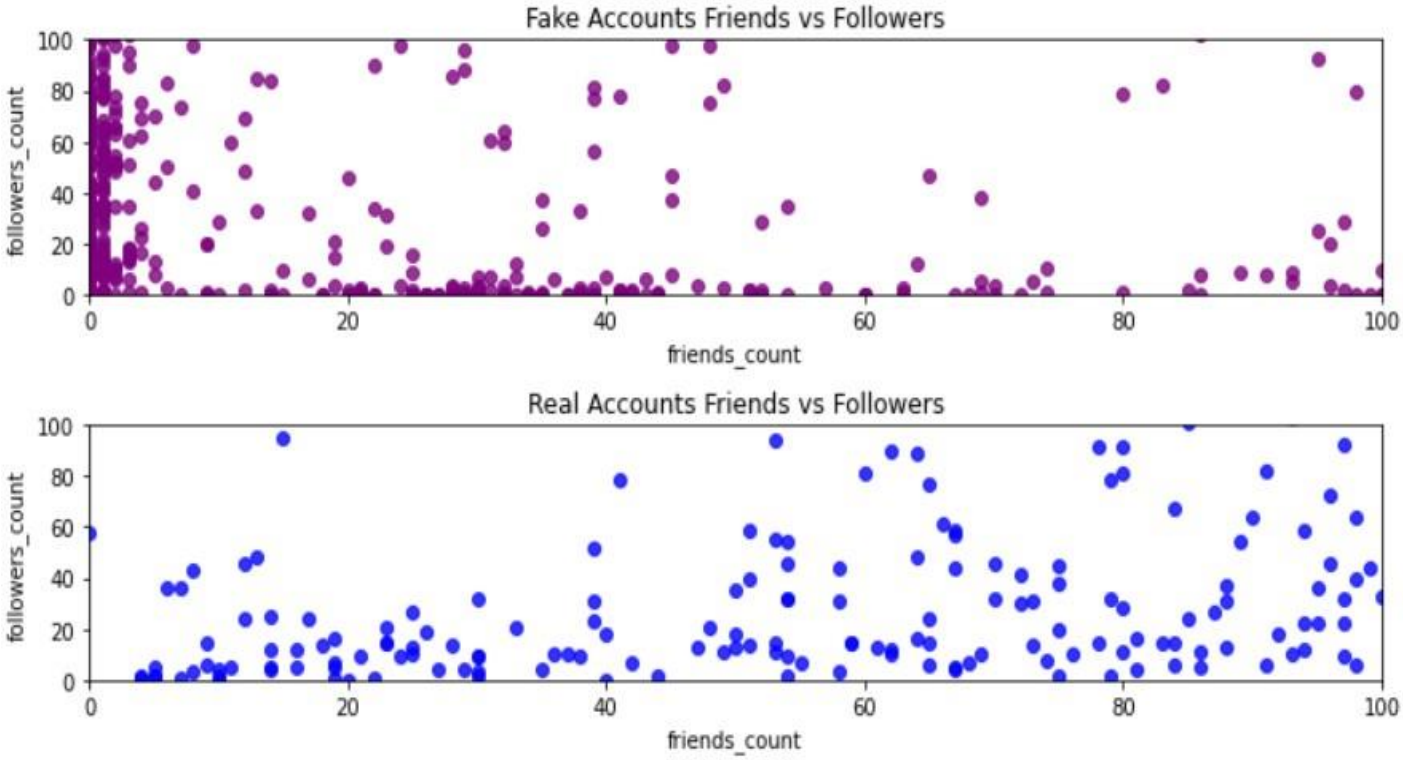


figure 2

As it is clearly visible in the previous figure that the feature — 'description', 'profile_image_url', 'lang', 'location' and 'profile_background_image_url' has null values. We won't be using 'profile_background_image_url' , 'description', 'lang', 'profile_image_url' in our analysis so, we can directly convert the NaN to string directly. But for 'location' -> NaN values is changed to 'unknown'.

6.2.2 Finding the general characteristics of real & fake accounts:

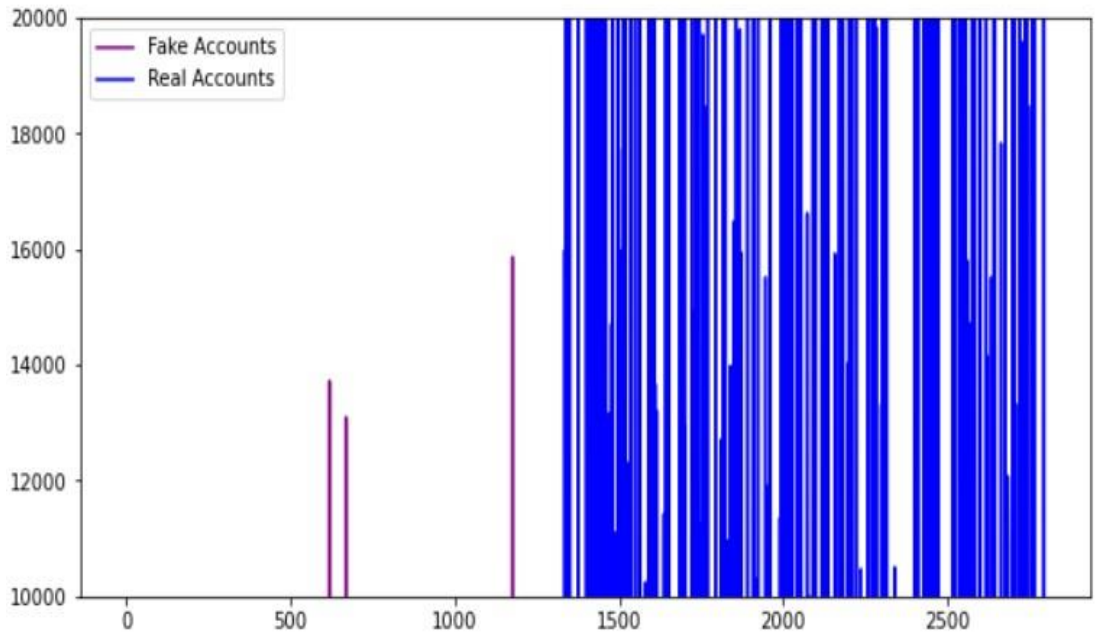
Moving on the project I found out the real account friends vs followers and fake/bot accounts friends vs followers by making the regplot that shows the same to depict the general trend on the characteristics of real and fake accounts that generally what is the ratio of friends to followers that real account have or fake account have, generally real or fake accounts have more followers than following or more following than followers etc. The regplots that was made is given below:



Above are two regplots for 'Bots friends vs Followers' and 'Non-Bots Friends vs followers', They depict the general trend on the characteristics of fake accounts or bot is that they possess more followers count compared to friends. On the other hand, Non-Bots or real accounts have generally an equal number of both friends and followers with some slight variance.

6.2.3 Identifying the Imbalance in the Data:

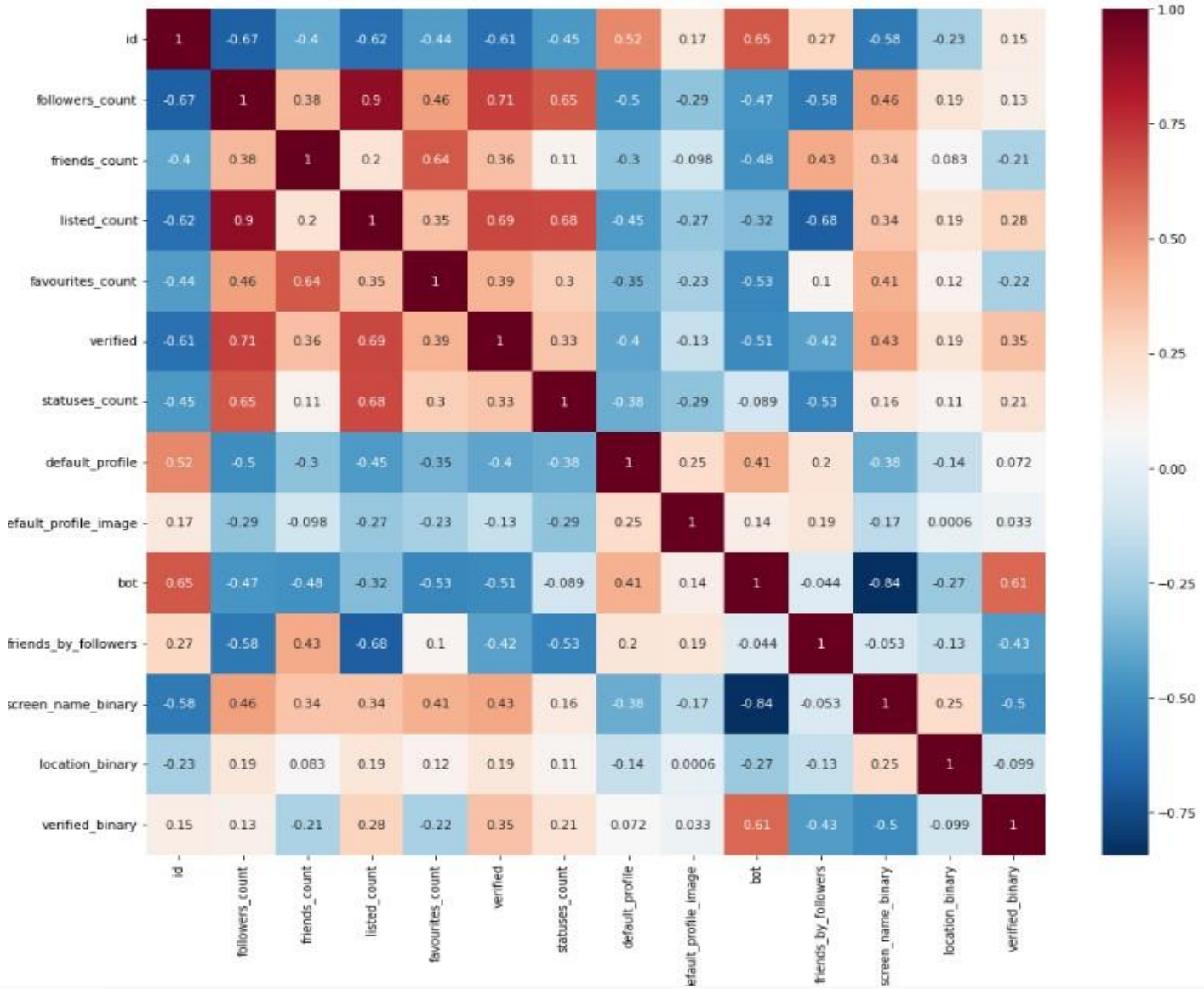
After this the next step is to find out the imbalances in the data. In this I found out that whenever the listed_count is between 10000 to 20000 there is approximately 5% of bot/fake account and approximately 95% of real accounts. This is also shown in the below figure:



6.2.4 Feature Independence using Spearman Correlation:

Then I identified the feature independence of data using Spearman correlation. Spearman's correlation coefficient is a statistical measure of the strength of a monotonic relationship between paired data. In a sample, it is denoted by 'rs' and is by design constrained as -1<=rs<=1 and its interpretation are similar to that of Pearsons, e.g. the closer is to the stronger the monotonic relationship. Correlation is an effect size and so we can verbally describe the strength of the correlation using the following guide for the absolute value of rs :

- 0 to 0.19 - "very weak"
- 0.2 to 0.39 - "weak"
- 0.4 to 0.59 - "moderate"
- 0.6 to 0.79 - "strong"
- 0.8 to 1 - "very strong"



	id	followers_count	friends_count	listed_count	favourites_count	verified	statuses_count	default_profile	default_profile_image	bot
id	1.000000	-0.672925	-0.402346	-0.615005	-0.439430	-0.611899	-0.451945	0.522990	0.166601	0.652131
followers_count	-0.672925	1.000000	0.375522	0.896126	0.457363	0.709732	0.649117	-0.496899	-0.293838	-0.468430
friends_count	-0.402346	0.375522	1.000000	0.204403	0.641529	0.356452	0.111118	-0.296358	-0.097607	-0.483105
listed_count	-0.615005	0.896126	0.204403	1.000000	0.349059	0.694340	0.684976	-0.447376	-0.269035	-0.318445
favourites_count	-0.439430	0.457363	0.641529	0.349059	1.000000	0.394227	0.295108	-0.348043	-0.226956	-0.526228
verified	-0.611899	0.709732	0.356452	0.694340	0.394227	1.000000	0.333278	-0.404650	-0.132298	-0.508555
statuses_count	-0.451945	0.649117	0.111118	0.684976	0.295108	0.333278	1.000000	-0.375918	-0.289999	-0.089018
default_profile	0.522990	-0.496899	-0.296358	-0.447376	-0.348043	-0.404650	-0.375918	1.000000	0.246979	0.407748
default_profile_image	0.166601	-0.293838	-0.097607	-0.269035	-0.226956	-0.132298	-0.289999	0.246979	1.000000	0.139669
bot	0.652131	-0.468430	-0.483105	-0.318445	-0.526228	-0.508555	-0.089018	0.407748	0.139669	1.000000
friends_by_followers	0.270435	-0.577157	0.427638	-0.681034	0.104797	-0.419815	-0.533971	0.197929	0.190986	-0.044056

The findings from Spearman Correlation results are:

• There is no correlation between id, statuses_count, default_profile, default_profile_image and the target variable.

• There is a strong correlation between verified, listed_count, friends_count, followers_count and the 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.

6.2.5 Feature Engineering:

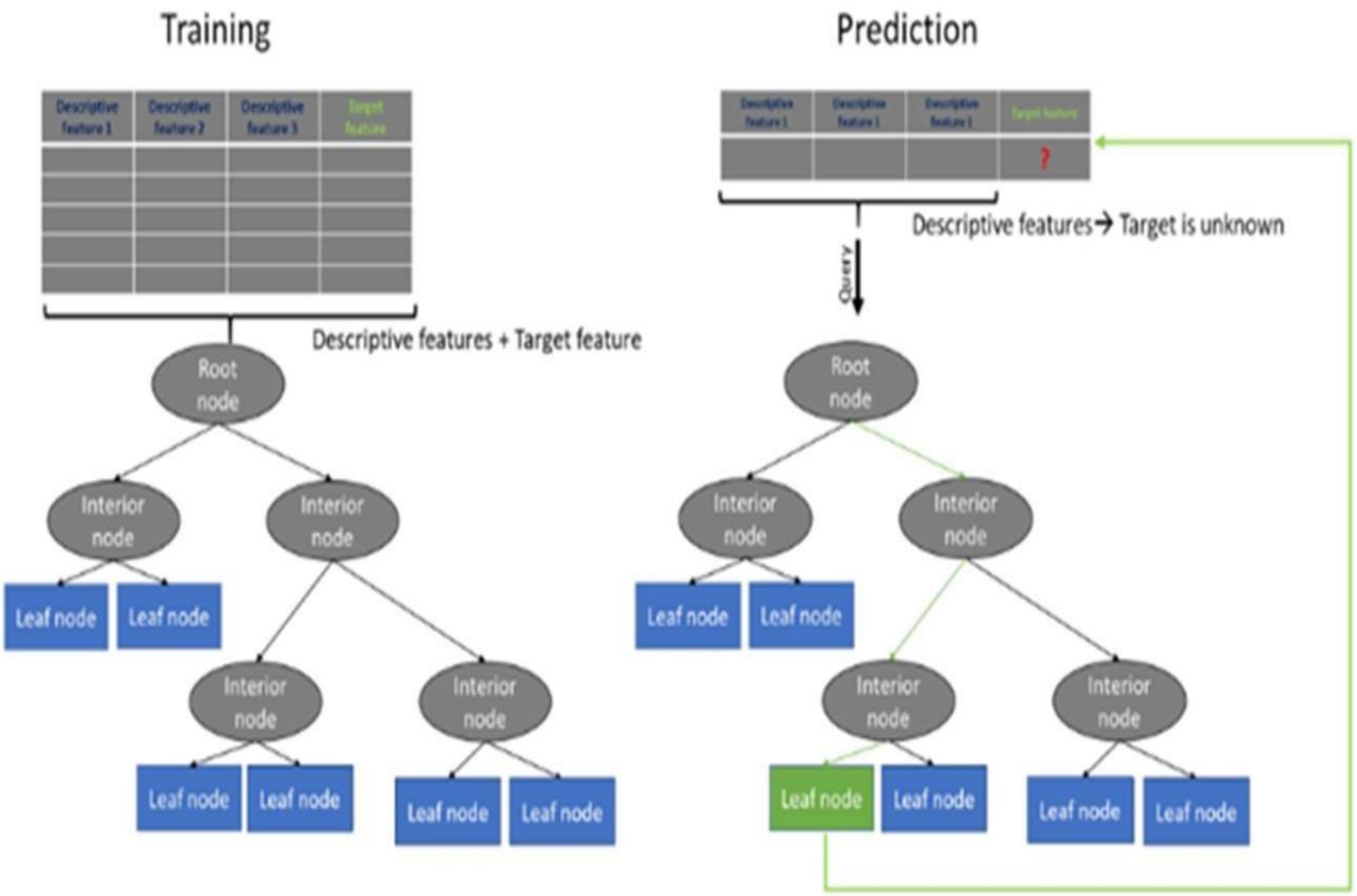
The next step is to do the feature engineering for this we will take name, screen_name, status, description while use the listed_count, verified for feature extraction [21]. For doing feature engineering a bag of world model is also created which identifies whether an account on twitter is bot or not. In feature engineering process we have to convert name, screen_name, status, description into a binary using our own vectorizer algorithm [21]. Then some feature extraction is done where we have taken listed_count_binary 1 and made it false where ever the original listed_count1 > 20000s and we created a new feature which consists of values of, 'name_binary', 'screen_name_binary', 'description_binary', 'status_binary', 'verified', 'friends_count', 'followers_count', 'listed_count_binary', 'statuses_count' and 'bot' [21].

6.3 Implementing Different Models:

We implement different machine learning model and hence find their accuracy on test and training set. And we all find out testing and training precision, confusion matrix, F1-score and recall of the different models. We will also plot the ROC curve which is a graphically plot created by plotting the true positive rate against the false positive rate at the various threshold which depicts the performance of all the models used for the study [22].

6.3.1 Decisions Trees Classifier:

The decisions trees is a well-liked and effective method of prediction and categorization. Similar to a flowchart, it is organised with each leaf or terminal node containing the name of a class, each internal node indicating a test on an attribute, and each branch designating the test's outcome.



Construction of Decision Tree:

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Decision Tree Representation:

Decision trees classify the instances by sorting them down the tree from root to some leaf node, which provides the classification of the instance. An instance is classified by starting from the root node of tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute. This process is then repeated for the subtree rooted at the new node.

Decision Trees Classifier Accuracy, F1-score, precision, recall and ROC curve:

Decisions Trees Classifier Train Confusion Matrix:

[[973 80]

[141 763]]

Decisions Trees Classifier Test Confusion Matrix:

[[389 34]

[68 349]]

Train precision of the Decisions Trees classifier: **0.90510**

Test precision of the Decisions Trees classifier: **0.91123**

Train recall of the Decisions Trees classifier: **0.84403**

Test recall of the Decisions Trees classifier: **0.83693**

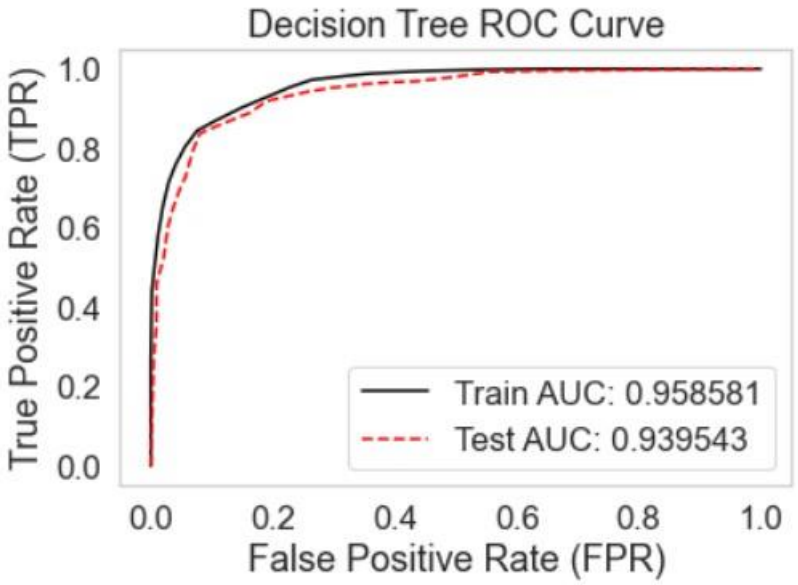
Train F1-score of the Decisions Trees classifier: **0.87350**

Test F1-score of the Decisions Trees classifier: **0.87250**

Training Accuracy of the Decisions Trees classifier: **0.88707**

Test Accuracy of the Decisions Trees classifier: **0.87857**

Decisions Trees classifier ROC curve is shown in the below figure:



Decision Tree Classifier ROC curve

6.3.2 Multinomials Naïve Bayes Classifiers:

Naive Bayes is a powerful machine learning algorithm that has proven to be both simple and effective, despite the recent advances in the field. This approach has been successfully applied in numerous applications and is especially helpful for natural language processing (NLP) jobs. A class of probabilistic models known as naive bayes uses Bayes' theorem and probability theory to predict the category of a given sample, such as a customer review or a news item. Naive Bayes determines which category has the highest probability for a given sample by computing the probabilities for each category. By using Bayes' theorem, which calculates the likelihood of a feature based on knowledge of potential confounding factors, this algorithm determines these probabilities.

Bayes' Theorem:

Now we need to transform the probability we want to calculate into Something that can be calculated using word frequencies. For this, we will use some basic properties of probabilities, and Bayes' Theorem. Bayes' Theorem is useful when working with conditional probabilities because it provides us with a way to reverse them.

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

Multinomials Naïve Bayes Classifiers Accuracy, F1-score, precision, recall and ROC curve:

Multinomials Naïve Bayes Classifiers Train Confusion Matrix:
[[458 595]
[32 872]]
Multinomials Naïve Bayes Classifiers Test Confusion Matrix:
[[181 242]
[12 405]]

Multinomials Naïve Bayes classifiers Train precision: **0.59441**
Multinomials Naïve Bayes classifiers Test precision: **0.62597**

Multinomials Naïve Bayes classifiers Train recall: **0.96460**

Multinomials Naïve Bayes classifiers Test recall: **0.97122**

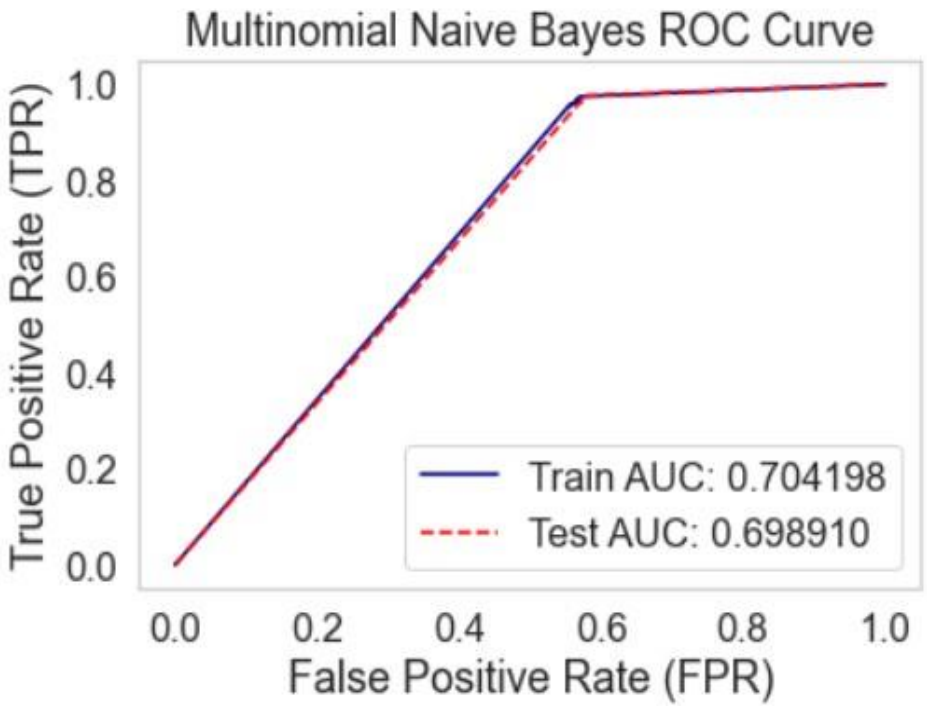
Multinomials Naïve Bayes classifiers Train F1-score: **0.73555**

Multinomials Naïve Bayes classifiers Test F1-score: **0.76128**

Multinomials Naïve Bayes classifiers Training Accuracy: **0.67961**

Multinomials Naïve Bayes classifiers Test Accuracy: **0.69762**

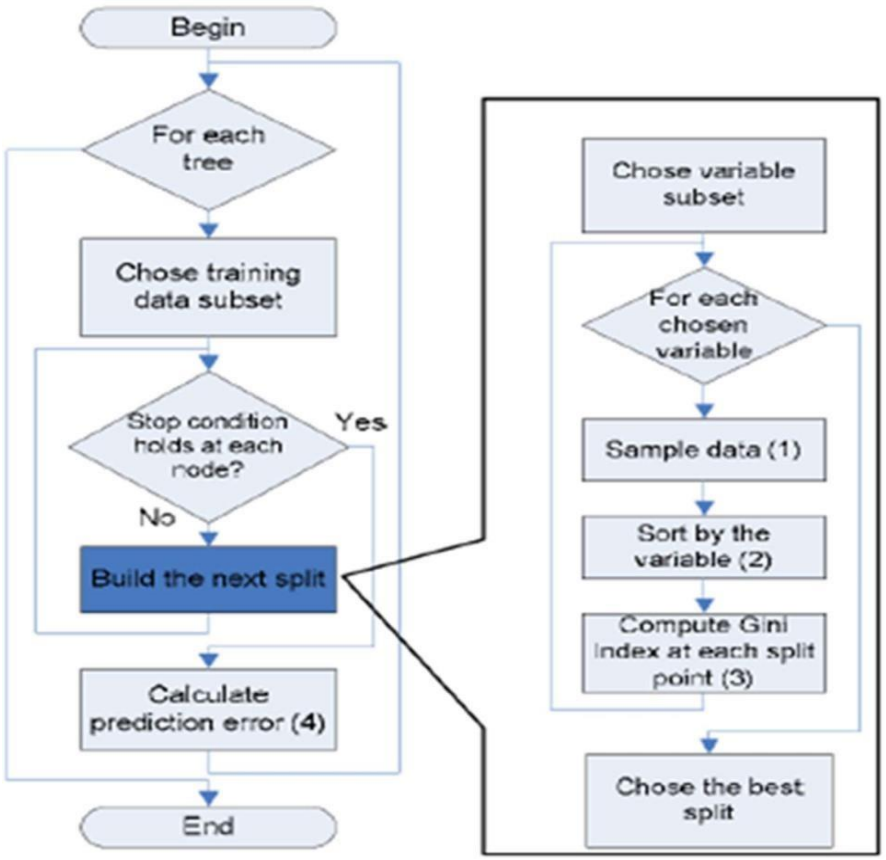
Multinomials Naïve Bayes classifiers ROC curve is shown in the below figure:

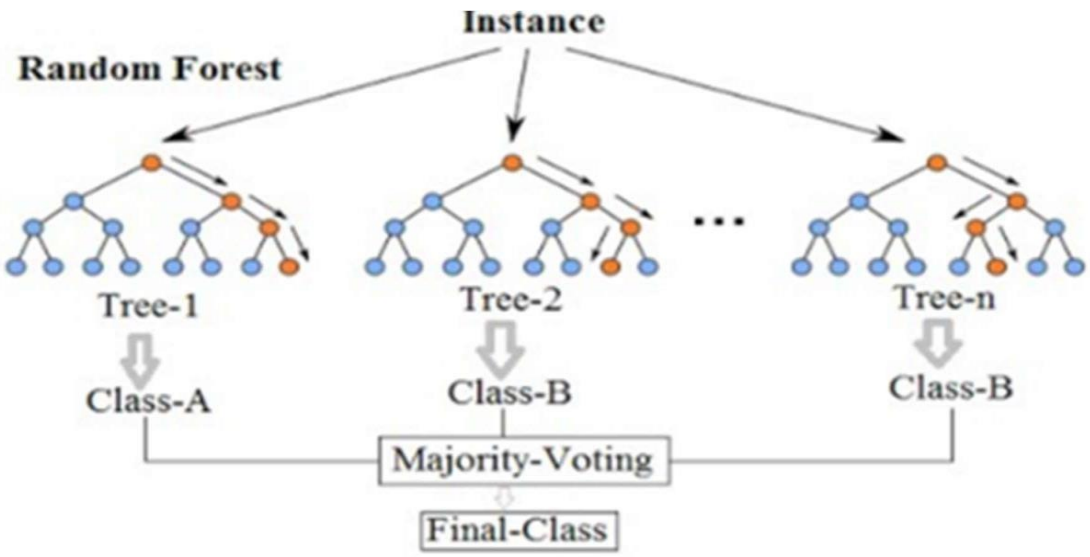


Multinomial Naïve Bayes Classifier ROC curve

6.3.3 Random Forest Classifier:

Random Forest is an extremely flexible and user-friendly machine learning method that frequently produces good results with only a small amount of hyperparameter modification. Its simplicity and suitability for both classifications and regressions problems make it one of the most extensively used algorithms.





RANDOMLY FORESTS CLASSIFIERS WORKINGS:

The Randoms Forests Classifiers is ones of the mostly widely used machines learnings algorithms for classification applications. It is an ensemble learning technique that combines the forecasts from numerous builtin decisions trees. A random subset of the features and a piece of the training data are used in the creation of each decision tree, which helps to reduce overfitting and improve model accuracy. The final prediction made by the random forest classifier is based on the majority votes of all the apredictions made by the decisions trees in the forests.

Randoms Forests Classifiers Accuracy, precision, F1-score, recall and ROC curve:

Randoms Forests Classifiers Train Confusion Matrix:

[[965 88]

[148 756]]

Randoms Forests Classifiers Test Confusion Matrix:

[[382 41]

[74 343]]

Train recall of the Randoms Forests classifiers: **0.83628**

Test recall of the Randoms Forests classifiers: **0.82254**

Train precision of the Randoms Forests classifiers: **0.89573**

Test precision of the Randoms Forests classifiers: **0.89323**

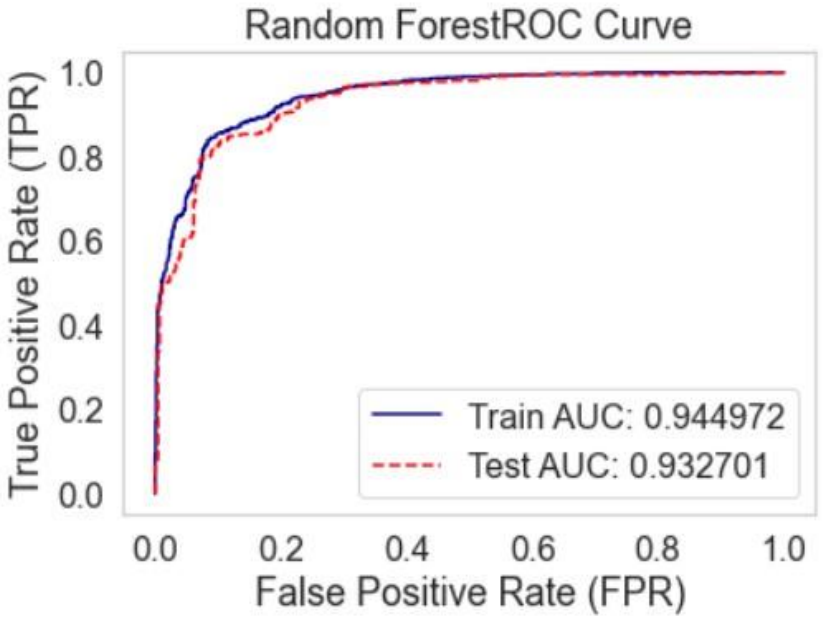
Train F1-score of the Randoms Forests classifiers: **0.86499**

Test F1-score of the Randoms Forests classifiers: **0.85643**

Training Accuracy of the Randoms Forests classifiers: **0.87941**

Test Accuracy of the Randoms Forests classifiers: **0.86310**

Randoms Forests classifiers ROC curve is shown in the below figure:



Random Forest Classifier ROC curve

Our Proposed Classifier Algorithm steps are as follows:

6.3.4 Our Proposed Classifier:

Step 1: Creating copy of dataframe in train_set

Step 2: Converting id to int

Step 3: Replacing Null values with 0 in Friends_count column

Step 4: Replacing Null values with 0 in Followers_count column

Step 5: Preparing bag of words for bot accounts

Step 6: Converting verified account into vectors (True->1 False->0)

Step 7: If the name, description, screen_name, status columns contains bot, then Store the data set in predicted_set_1

7.1: Assign bot column as 1 (BOT)

7.2: Store the rest data set in verified_set for next step

Step 8: For all verified account, Store the data set in predicted_set_2

8.1: Assign bot column as 0 (NON_BOT)

8.2: Store the rest data set in followers_following_set for the next step

8.3: predicted_set_1=:concatenate(predicted_set_1,predicted_set_2)

Step 9: If followers_count is less than 50 and statuses_count greater than 1000 , then then Store the data set in predicted_set_2

9.1: Assign bot column as 1 (BOT)

9.2: Store the rest data set in followers_retweet_set for next step

9.3: predicted_set_1=:concatenate(predicted_set_1,predicted_set_2)

Step 10: If followers_count is less than 150 and statuses_count greater than 10000 then then Store the data set in predicted_set_2

10.1: Assign bot column as 1 (BOT)

10.2: predicted_set_1=:concatenate(predicted_set_1,predicted_set_2)

Step 11: Store the rest data set in predicted_set_2 and assign bot column as 0 (NON_BOT)

11.1: predicted_set_1=:concatenate(predicted_set_1,predicted_set_2)

Step 12: Return predicted_set_1

Our Proposed Algorithm based Classifier Accuracy, F1-score, precision, recall and ROC curve:

Our Own Train Confusion Matrix:

[[1083 0]

[66 905]]

Our Own Test Confusion Matrix:

[[393 0]

[31 319]]

Our Own Train precision: **1.00000**

Our Own Test precision: **1.00000**

Our Own Train recall: **0.92020**

Our Own Test recall: **0.94562**

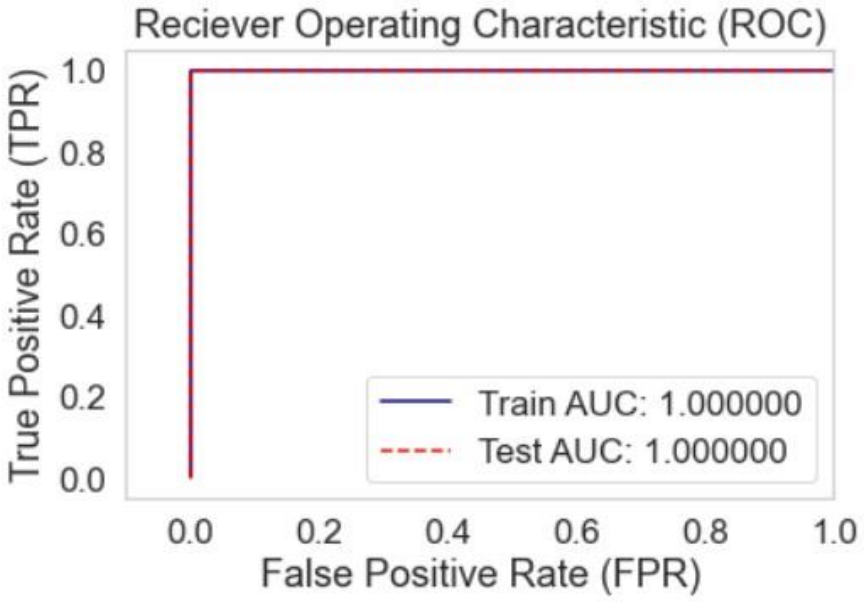
Our Own Train F1-score: **0.95844**

Our Own Test F1-score: **0.97205**

Our Own Train Accuracy: **0.9638267491670633**

Our Own Test Accuracy: **0.9775596072931276**

Our Own Classifier ROC curve is shown in the below figure:

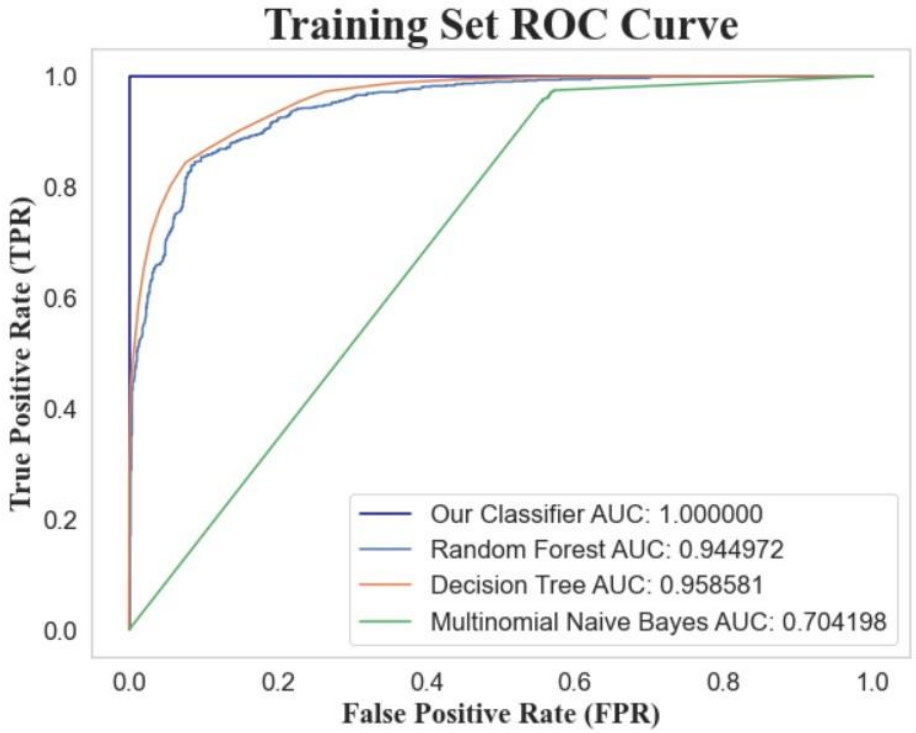


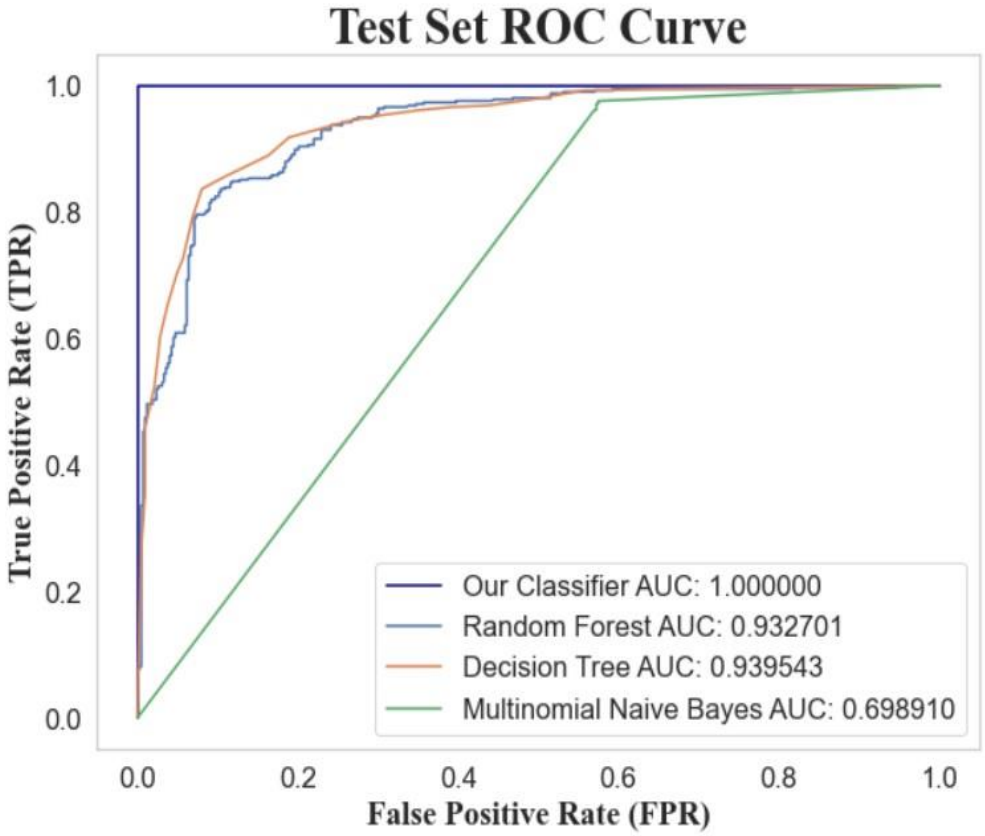
Our Own Classifier ROC curve

5.4 Comparing all the Models :

6.4.1 ROC Curve of all the models combined:

Here we made a combined the ROC curve of all the models to find out which model AUC is best among all. Below figures shows the combined Training set ROC Curve and Test ROC Curve of all the models.

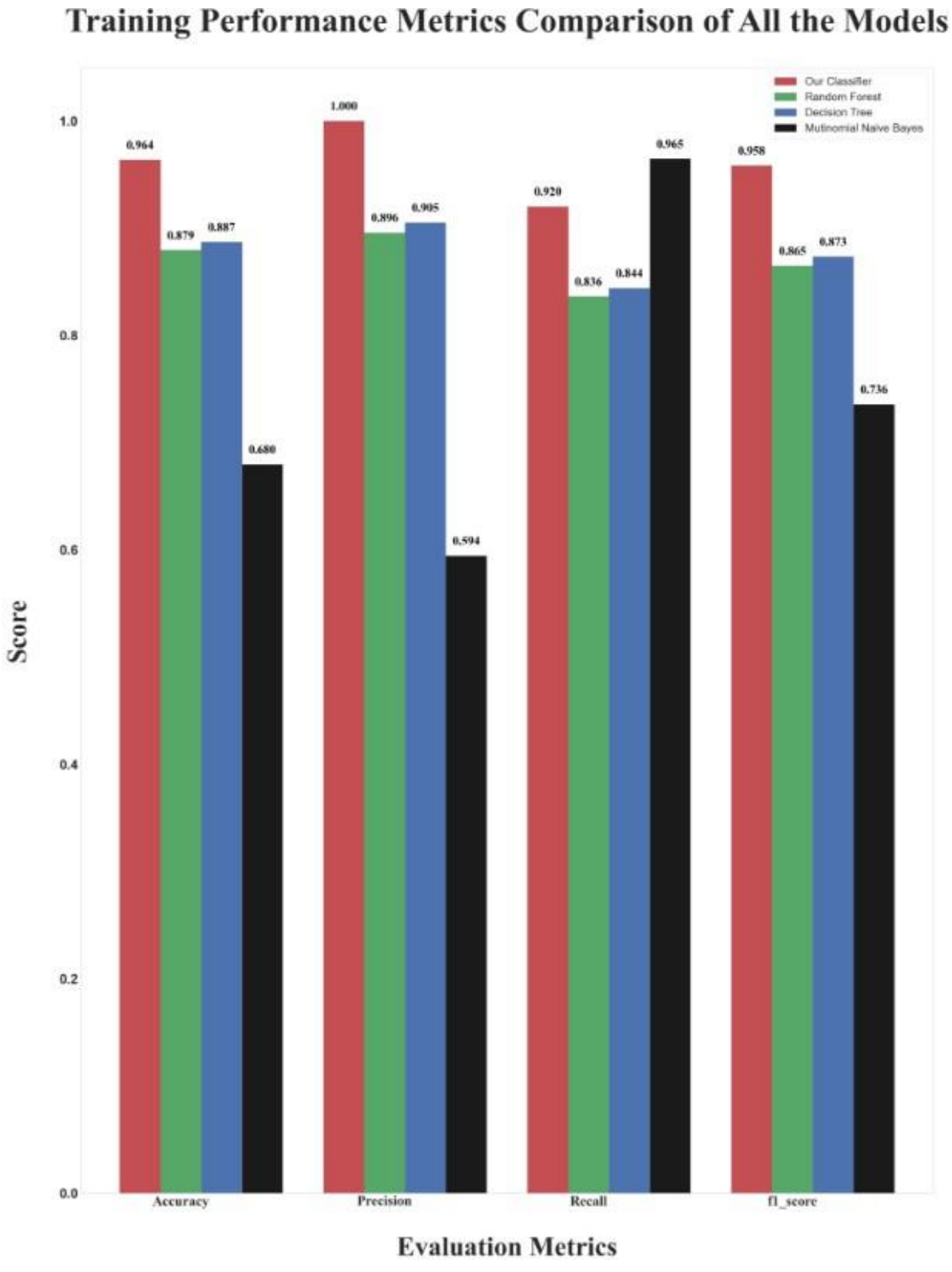


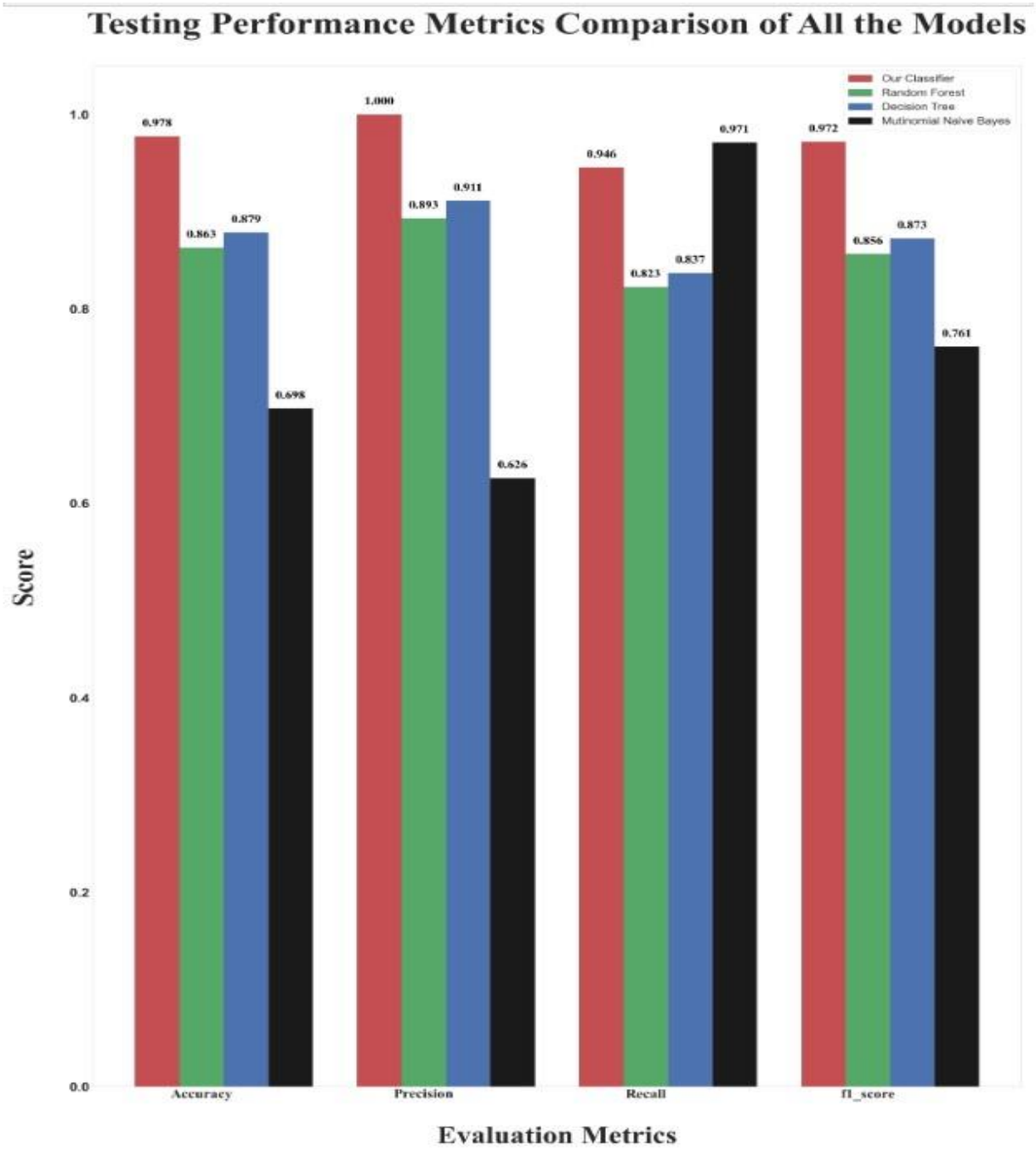


The above figures shows that the our classifier have the best training and test AUC as compared to all the models.

6.4.2 Comparison Charts:

After checking out which model has the best AUC we will made a comparison charts of all the models comparing all the models accuracy, F1-score, recall and precision to find out which model has performed well in which areas. Below figures shows the training set and testing set comparison charts between all the four models.





The comparison charts shows us that our classifier has performed the best when it comes to accuracy, precision and F1-score in both training and testing data and Multinomial Naïve Bayes has performed the best in recall in both training and testing data whereas our classifier was the second best in recall. Overall our classifier has out performed the other three classifiers this comparison chart also shows that so our main motive to create a classifier that perform better than traditional machine learning classifiers to detect fake accounts in Twitter dataset is achieved.

CHAPTER 7: RESULTS

The final output from our classifier predicting fake/bot accounts from the data that was given to our classifier for making the predictions was stored in submission.csv and it will be of the form as shown below figure. In the below figure the class label 'bot' represent whethers a given users is fake or a real's users. The entry 1 represent a fake user i.e., bot and 0 represent a real user i.e., nonbot.

	A	B
1	id	bot
2	1.32E+09	1
3	2.56E+09	1
4	3.48E+08	1
5	8.56E+08	1
6	8.33E+17	1
7	7.14E+17	1
8	2.31E+09	1
9	7.43E+17	1
10	85430866	1
11	8.28E+17	1
12	2.39E+08	1
13	5.38E+08	1
14	2.57E+09	1
15	3E+09	1
16	3.41E+09	1
17	7.02E+08	1
18	3.29E+08	1
19	1.26E+09	1
20	8.07E+17	1
21	3.04E+09	1
22	2.58E+09	1
23	8.46E+17	1
24	8.54E+08	1
25	7.19E+17	1
26	2.42E+09	1
27	1.39E+09	1

submission

In project implementation we have made comparison charts between our classifier and Decisions Trees, Randoms Forests & Multinomials Naïve Bayes Classifiers which showed us that our classifier have a best training and tests accuracy, precision, F1-scores as compared to the other models and it also showed that our classifier have the second best training and test recall as compared to the other classifiers and we also made a combined ROC curve of all the models which showed that our model has the best train and test AUC as compared to all the other classifiers. So all this clearly shows that our model predict the fake/bot accounts better than all the other models.

CHAPTER 8: CONCLUSION AND FUTURE SCOPE

Twitter fake account detection is a program used to predict whether a Twitter account is fake or real by knowing few basic information about the Twitter account. In this project, we used various Machines Learnings techniques to predicts weathers an account on Twitter is a fakes or a reals user. We a have performed significant amount s of feature engineering, along with feature extractions & then selected the best features out of 20 given to us in the dataset which helped us to identify whether an account is fake/bot or real/non-bot. We implemented various algorithm and finally implemented our own which gave us AUC>95%. Our framework will be able to identify whether a twitter user is a bot s or a human. We can extend our work to other social media platform like Facebooks. Our work will safeguard oneself and an organization from false information, malicious content and ensure their brand value.

CHAPTER 9: REFERENCES

[1] “Top Trending Twitter Topics for 2011 from What the Trend,” http://blog.hootsuite.com/top-twittertrends-2011/, Dec. 2011.

[2] “Twitter Blog: Your World, More Connected,” http://blog. twitter.com/2011/08/your-world-moreconnected.html, Aug. 2011.

[3] Alexa, “The Top 500 Sites on the Web by Alexa,” http:// www.alexa.com/topsites, Dec. 2011.

[4] “Amazon Comes to Twitter,” http://www.readwriteweb.com/amazon_comes_to_twitter.

[5] “Best Buy Goes All Twitter Crazy with @Twelpforce,” http://twitter.com/in_social_media/

[6] “Barack Obama Uses Twitter in 2008 Presidential Campaign,” http://twitter.com/Obama/, Dec. 2009. [7] J. Sutton, L. Palen, and I. Shlovski, “Back-Channels on the Front Lines: Emerging Use of Social Media in the 2007 Southern California Wildfires,” Proc. Int’l ISCRAM Conf., May 2008.

[8] A.L. Hughes and L. Palen, “Twitter Adoption and Use in Mass Convergence and Emergency Events,” Proc. Sixth Int’l ISCRAM Conf., May 2009.

[9] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang, “Measurement and Classification of Humans and Bots in Internet Chat,” Proc. 17th USENIX Security Symp., 2008.

[10] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, “Your Botnet Is My Botnet: Analysis of a Botnet Takeover,” Proc. 16th ACM Conf. Computer and Comm. Security, 2009.

[11] S. Gianvecchio, Z. Wu, M. Xie, and H. Wang, “Battle of Botcraft: Fighting Bots in Online Games with Human Observational Proofs,” Proc. 16th ACM Conf. Computer and Comm. Security, 2009.

[12] A. Java, X. Song, T. Finin, and B. Tseng, “Why We Twitter: Understanding Microblogging

Usage and Communities," Proc. Ninth WebKDD and First SNA-KDD Workshop Web Mining and Social Network Analysis, 2007.

[13] B. Krishnamurthy, P. Gill, and M. Arlitt, "A Few Chirps about Twitter," Proc. First Workshop Online Social Networks, 2008.

[14] S. Yardi, D. Romero, G. Schoenebeck, and D. Boyd, "Detecting Spam in a Twitter Network," First Monday, vol. 15, no. 1, Jan. 2010.

[15] A. Mislove, M. Marcon, K.P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," Proc. Seventh ACM SIGCOMM Conf. Internet Measurement, 2007.

[16] S. Wu, J.M. Hofman, W.A. Mason, and D.J. Watts, "Who Says What to Whom on Twitter," Proc. 20th Int'l Conf. World Wide Web, pp. 705-714, 2011.

[17] H. Kwak, C. Lee, H. Park, and S. Moon, "What Is Twitter, a Social Network or a News Media?" Proc. 19th Int'l Conf. World Wide Web, pp. 591-600, 2010.

[18] I.-C.M. Dongwoo Kim, Y. Jo, and A. Oh, "Analysis of Twitter Lists as a Potential Source for Discovering Latent Characteristics of Users," Proc. CHI Workshop Microblogging: What and How Can We Learn From It?, 2010.

[19] D. Zhao and M.B. Rosson, "How and Why People Twitter: The Role Micro-Blogging Plays in Informal Communication at Work," Proc. ACM Int'l Conf. Supporting Group Work, 2009.

[20] https://www.anaconda.com/

[21] https://iopscience.iop.org/article/10.1088/1742- 6596/1950/1/012006

[22] https://www.ijitee.org/wpcontent/uploads/papers/v9i7/G5919059720.pdf