VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAUM, KARNATAKA-590014

A Project report on

"A Online Media Bot Identification Using Machine Learning Algorithm For Twitter Social Networks"

Submitted in Partial fulfillment of the requirements for the 8th semester

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

By

APOORVA CHAKMA	1SP19CS007
PRANITH JAIN	1SP19CS074
PRASHANT KUMAR	1SP19CS075
RAHUL THAKUR	1SP19CS081

Under the guidance of:

Asst Prof. Jayashri M
Dept of CSE

Department of Computer Science and Engineering

S.E.A. COLLEGE OF ENGINEERING AND TECHNOLOGY

BENGALURU-560049

2022-2023

S.E.A COLLEGE OF ENGINEERING AND TECHNOLOGY

Ekta Nagar, Basavanpura, Virgonagar Post, K. R. Puram, Bengaluru, Karnataka 560059

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify the project work entitled "**A Online media Bot Identification using Machine Learning Algorithm For Twitter Social Networks**" has been successfully carried out by **Mr. APOORVA CHAKMA (1SP19CS007), Mr. PRANITH JAIN(1SP19CS074), Mr. PRASHANT KUMAR (1SP19CS075), Mr. RAHUL THAKUR(1SP19CS081)** of VIII Sem in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Technology of the Visvesvaraya Technological University, Belgaum during the year 2022.Theproject report has been approved as it satisfied the academic requirement in respect of the Project work prescribed forthe Bachelor of Engineering.

Signature of Guide	Signature of HOD	Signature of Principal
Asst Prof. Jaya Shri M	**Dr. B. Loganayagi**	**Dr. B Venkata Narayan**

ACKNOWLEDGEMENT

Firstly, we thank the Management late **Shri A Krishnappa**, Chairman SEA College ofEngineering and Technology for Providing the Necessary infrastructure and creating a goodenvironment.

We would like to express my profound thanks to our respected principal **Dr. B Venkata Narayan** For the encouragement and support given by him.

We would like to express my sincere thanks to our respected **Dr. B. Loganayagi**, **HOD ofCOMPUTER SCIENCE AND ENGINEERING** department, for his assistance and guidance .

We am thankful for the support rendered by my Project guide and coordinator **Prof. Jayashri M** forher valuable suggestions.

We am also obliged, to the faculty members of CSE Department who rendered their valuable assistance for the Project.

And finally,W e would like to express my heart full gratitude to my friends and all those whohave extended their help throughout my Project.

ABSTRACT

Twitter is one of the popular social networking sites which allow the users to express their opinion on various topics like politics, sports, stock market,entertainment etc.It is one of the fastest means of conveying information.It highly influences people's perspective. So it is necessary that tweets are sent by genuine users and not by twitter bots.A twitter bot sends spam messages. Therefore detecting of bots helps to identify spam messages.The proposed bot detection method analyzes Twitter specific user profiles having essential profile - centric features and several activity - centric characteristics.We have constructed a set of filtering criteria and devised an exhaustive bag of words for performing language-based processing.In order to substantiate our research, we have performed a comparative study of our model with the existing benchmark classifiers

INDEX

FIGURE INDEX

Chapter 1: INTRODUCTION

Twitter is a blogging platform that allows users to post tweets, which appear to be short texts of minimum 200 characters. Users on Twitter can participate by replying to tweets, mentioning other users in their tweets, or reposting another user's message, this one is recognized as re tweeting. The basis of automated bot identification is that human account behavior varies from that of a bot. These categorizations may be quantified using typical factors such as, for example, the statistical distribution of phrases used in tweets, the posting rate per day, and the number of followers. A website has indeed been developed to help people find this Twitter bot. It is possible to perform the aforementioned using a machine learning technique. The term for machine learning is research of computational tools that can enhance themselves instantaneously based on empirical evidence and statistics. It is classified under artificial intelligence. Without being supervised, machine learning algorithms construct a conceptual model using observations, referred to as "training data," in order to test hypotheses or make conclusions. Machine learning techniques have been used in a broad array of applications, along with healthcare, phishing emails, speaker identification, and data analysis, where creating modelling techniques to perform the required tasks would be difficult or impossible. Machine learning is strongly linked to operational research, which enables us to create assumptions with desktops; however, not every machine learning is mathematical training.These features are defined by the account used after a tweet and have the benefit of not changing greatly over time because the content of each tweet does not vary significantly. As a consequence, these traits may be assessed for irregularity. Because these accounts may be vacant and newer, characteristics such as join date.

These characteristics track that how user or bot communicates with the Twitter service and how it uses the account. Bots can have different blogging behaviors than authorized user, such as publishing at periodic times or in portions that individuals would find difficult. Besides that, bots typically connect the Twitter platform through the Application programming interface as well as other processes that help automated programs to publish, genuine users, on the other hand, often employ online or mobile interfaces. Because bots may transmit the same text to numerous people, account utilization characteristics can also acquire different statistics by obtaining information statistics, such as analyzing the variance between messages

1.1 LITERATURE SURVEY

1. Fake identities created by humans or bots are detected using machine learning models:-

Supervised machine learning algorithms require a data-set of features with a label classifying each row or outcome. Features are thus the input used by supervised machine learning models to predict an outcome. The predictive results from the trained machine learning models only kept the accuracy 49.75%.The machine learning models were trained to use engineered features without relying on behavioral data.

2. Prediction of twitter bot that hijack the conversation:-

They investigated two characteristics of tweets i.e. temporal information and message diversity. It was found that content polluters in this data-set often timed their tweets together By analyzing the temporal patterns one could infer the presence of bot accounts. It was also found that bots used a small set of URLs in their tweets. They used supervised method to detect the bots on the twitter.They used a Logistic Regression (LR) with a Random Forest(RF).

3. Bot detection using supervised algorithm by Pozzana and Ferrara (2020):-

They used supervised method to detect the bots on the twitter.They used a Logistic Regression (LR) with a Random Forest(RF). The accuracy they got was approx 85%

4. Detection of twitter spam and fake followers:-

Twelve features are generated which available in bot repository data-set such as followers count, friends count, etc., using statistic derivation. Other features as number of hash tags per tweet. The problem of classifying users as bot or human in a twitter they found by comparing the performance of these three approaches. Logical regression, neural network and gradient boosted.

5.Detection on spam messages using Precision, recall and f-measure: -

Studies had shown that most of the span messages were automatically produced by hots. Therefore bot spammer detection reduces the spam messages. Time level entropy and tweet similarity were used as criteria for spammer detection. Precision, recall and (-measure of this method resulted in85%,94% and 90% respectively.

1.2 PROBLEM STATEMENT:

It is proposed that there are several varieties of bots, each with its own distinct behavior.

If the behaviors seen in the training samples are sufficiently dissimilar, these new sorts of bots may be difficult to recognize using supervised classifiers.Given that the types of bots will continue to change in the future, with bot authors changing the behaviors to evade detection

Twitter posts are mostly public and can be easily collected using Twitter's developer platform API. Also, frequent use of hashtags makes it more interesting to draw conclusions.

Twitter Blue is an opt-in, paid subscription that adds a blue check mark to user account and offers early access to select features, like Edit Tweet. In order to detect bots, classification models and techniques such as topic modeling and sentiment analysis can be incorporated. This project will involve feature engineering and will provide a real-world data collection experience.

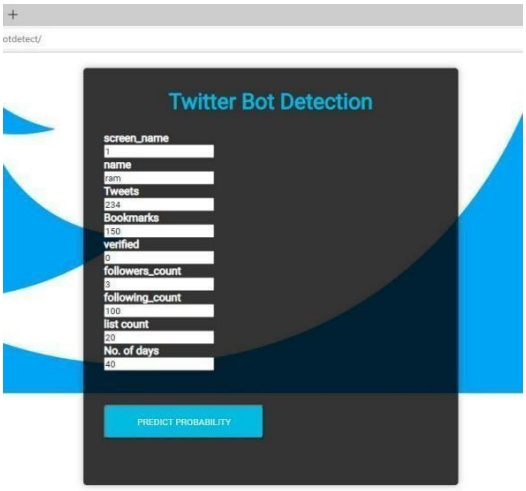


Fig 1 : Twitter Bot Detection

1.3 EXISTING SYSTEM :

revert from the malicious side twitter bots are find out. The data-set trained and tested with the algorithm that provides high accuracy. In website the characteristics are given as input. It checks according to each count which will not be a possibility of being a legitimate account. The system is user friendly because the characteristics to find the twitter bot is easily known. Furthermore, the suggested system has a graphical user interface, which allows users to interact with the system quickly. User can avoid the twitter bots which is has the high probability of being a twitter bot account.

1.4 PROPOSED SYTSEM:

- Evaluate 'out-of-the-box' models and narrow down candidates to 2-3 models based on Cross-validated scoring metrics.
- Consider class-weight balancing (data is ~70/30 split human/bot).
- Refine feature selection and tune candidate model parameters.
- Consider assembling best models with Voting Classifier model.
- Pick best model, and perform full train and test.
- Train best model on full data set.

- Understand psychology behind bot creation
- Implement different machine learning algorithms
- Hashing technique to quickly organize the user accounts in clusters of abnormally correlated accounts
- Custom classification algorithm
- Do not consider Attributes like id, status_count, default_profile, image etc. there should be no correlation between them
- Consider attributes like followers,friends, verified account etc.
- Other attributes like name, description, status for feature extraction.
- Existing solutions use naive bayes and decision trees
- We are planning to implement our own custom algorithm based on few feature extraction
- Aiming to get highest accuracy and differentiate between real twitter accounts and bots

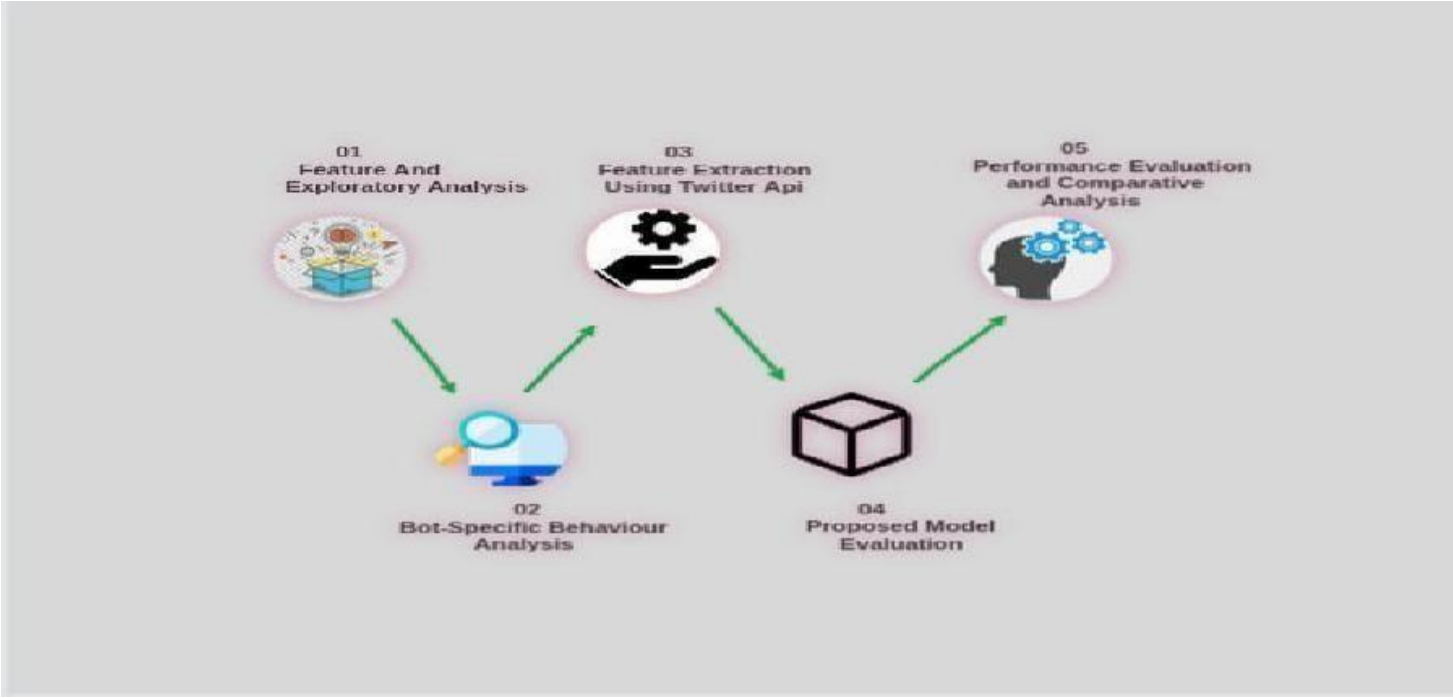


Fig 2 : Proposed Architecture

1.5 ADVANTAGES OF PROPOSED SYSTEM:

- Due to machine learning techniques, it improves accuracy of fake account detection systems. (The network or computer is constantly monitored for any invasion or attack.

(Twitter's major advantage is, Twitter has limited message size of 140 characters per tweet, it can include a message or link on your website as it is free and also free for the advertisements, you do not have to face the problem with bunch of posters like the other social networking

- (The major advantage of Support Vector Machines that classify our composite data model.

Chapter 2: SYSTEM DESIGN

2.1 Block Diagram

A block diagram is a specialized, high- level flowchart used in engineering. It is used to design new systems or to describe and improve existing ones. Its structure provides a high-level overview of major system components, key process participants, and important working relationships.

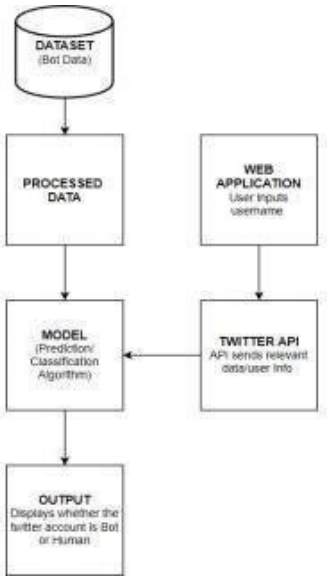


Fig 3: Block Diagram

2.1 Use case Diagram

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified. When the initial task is complete, use case diagrams are modelled to present the outside view.

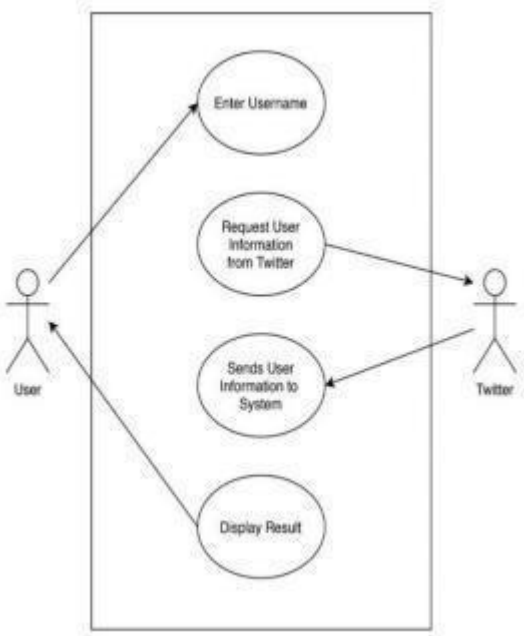


Fig 4: Use Case Daigram

Chapter 3: PROJECT METHODOLOGY

3.1 METHODOLOGY:

- The goal of this project is to use machine learning classification models to detect whether a Twitter user is a bot based on account-level information (e.g. number of followers, number of tweets, etc.).
- This approach will not look at the actual contents of tweets.
- After exploring the data and identifying and engineering some potential features, we will evaluate several classification models to find the best one for Twitter bot detection.
- We will be searching for models that have balanced scores between precision and recall and strong ROC AUC scores -- while we want the model to accurately label bots as often as possible, we also want to reduce miss classification and not simply label everything as a bot
- Models to be evaluated KNearestNeighbors, LogisticRegression, NaiveBayes (Gaussian, Bernoulli,Multinomial),DecisionTree,RandomForest,bagofwords.

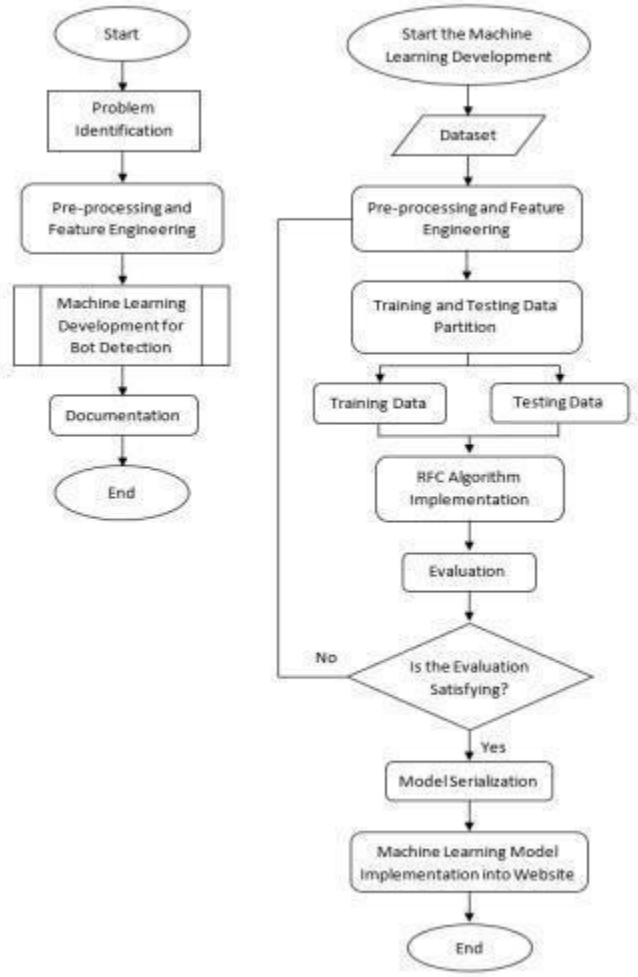


Fig 5: Flow Chart Implementation

3.2 THEORETICAL FRAMEWORK

A. Classification

Type Classification is a process of assigning a category or label that has been defined as data that does not yet have a category. In general, there are three types of the data classification process, namely binary, multi class, and multi label classification

- Binary classification is a process of classifying each element in a group into two groups or categories.
- Multi class classification is a classification process involving more than two classes. However,the multi class classification creates an assumption that each given sample
- A multi label classification is a classification process that puts samples into a set of targets. This classification predicts the properties of data that are not mutually exclusive. Examples of this classification are found in document classification.

Machine Learning

Machine learning is a technique that enables the system to learn from data compared to using direct programming so that it can deliver relevant results .

Random Forest Algorithm

Random forest was first introduced by Leo Breiman . The random forest classifier is the development of the decision tree. It consists of a combination of many decision trees, with each tree relying on independent random vector values with an equivalent distribution of each tree .

Twitter Social Media

Twitter is a micro-blogging social network that allows its users to send and read short messages up to 140 words, which are then called tweets . Jack Dorsey founded this social media in 2006. Unlike social media such as Facebook or MySpace, on Twitter, the relationship between to follow an account and the followers are not reciprocal. It means that an account can follow other accounts without automatically be followed bythe account it follows.

Bots and Twitter Bot Types

In general, bot means an application that performs tasks automatically. In social media, bot domain i a social media accounts programmed to perform social media activities automatically, so they look like real humans. According to research from the University of Southern California, at least 9% to 15% of active Twitter users are bots . Until 2017, there were 319 million active users each month. It means there are almost 48 million bot accounts spread on the Twitter social network. Factors that influence bot growth include Twitter API support, bot development cycles that can be created quickly, Twitter public platforms, and the flexibility to create as many accounts as possible. According to the Digital Forensic Research (DFR)of the Atlantic Council Lab, there are several features indicating that an account is a bot, includingamplification, anonymity, activity, similarity, and description of "bot" in the account .

Whereas the Twitterbot types based on account activity are as follows:

• Informative, i.e., a bot that functions to disseminate information to users. For example, bots that publishfacts, earthquake information, and write poetry content as well as humor content .

• Spammers, i.e., bots that work to broadcast spam content .

• Fake Followers, i.e., bots that act as shadow followers for an account. The purpose of using fake followersis to create an image that an account seems to have prominent popularity .

Waterfall Method

This method emphasizes the planning and scheduling process before starting the system development. This method is best used if the product definition is clear, the project is short lived, technology is known, and resources are available. The advantages of this method are organized documentation, proper to be used for known needs, and easily understood. The weakness of this method is the need for appropriate management, and small mistakes will be a big problem if not noticed from the beginning of development, high risk, and not a good model for intricate work.

Classification Evaluation

A metric evaluation is a set of metrics used to measure a classifier's performance. Different metrics measure different classifier characteristics. Evaluation metrics consist of three types, namely, threshold, opportunity, and ranking metrics

Chapter 4 : Implementation

4.1 Feature Selection / Extraction

Fig 6 :Feature Extraction

Feature Selection / Extraction Out of these attributes, screen name, name, description, status, verified and listed count are binary attributes. And the rest are numerical values. Apart from listed count and verified attributes, other attributes were made as binary-valued by comparing them with some words. We found that there were a set of words that were found to be there in the descriptive fields of a bot account. Some of those words are bot, prison, paper, follow me, tweet me, swag, bang, b0t, magic, face, wizard, etc. So, if any of these words are present in those above-mentioned descriptive fields, then there is a high chance of that account being classified as a bot.

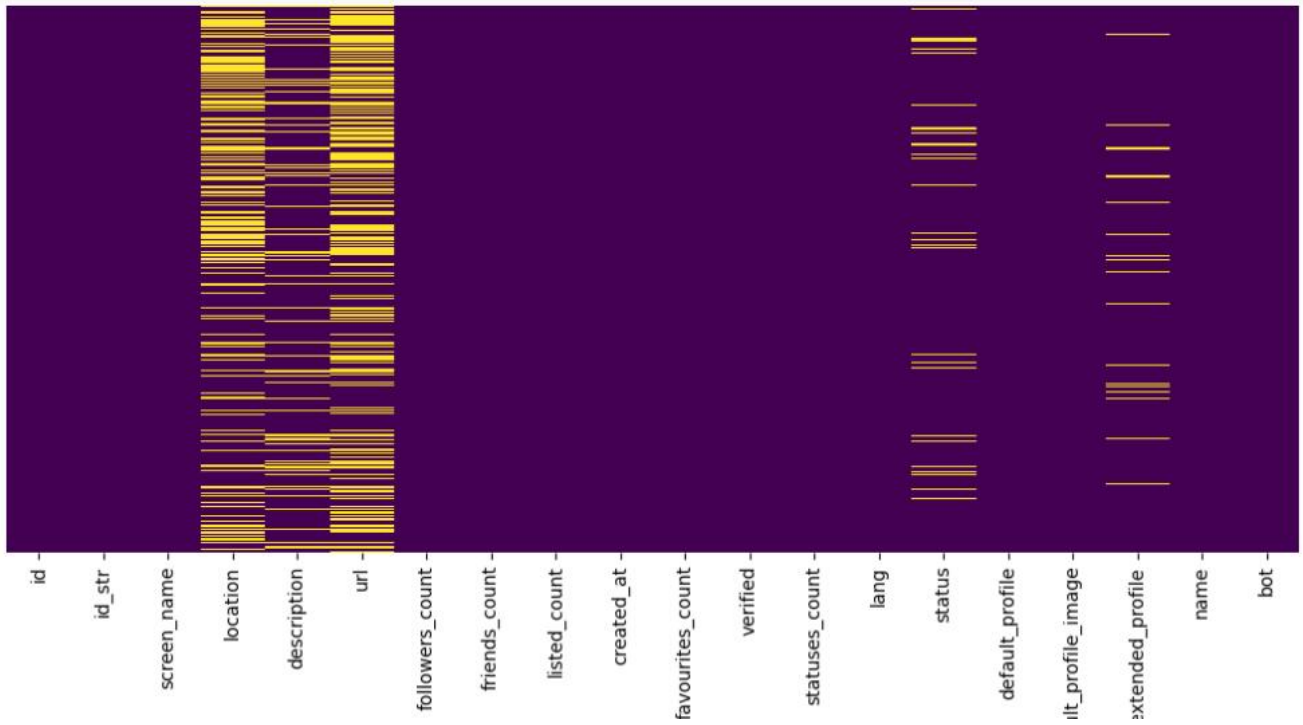


Fig 7 : Feature distribution

4.2 Data Set

We Collect The Twitter Bot Accounts data set from Kaggle, we are hoping to find features in account-level information that can aid in Twitter bot detection.

The data set is comprised of approximately 37,000 Twitter users,labeled bot or human,with account-levelinformation like: number of favorites/likes number of tweets number of followers,number of friends (accounts their following) whether or not the profile is still in default mode and more

In this notebook, I'll be exploring some of these provided features as well as transforming the data to create some interesting interactions that might aid in creating a predictive classification model

4.3 Behaviour of Human vs bot

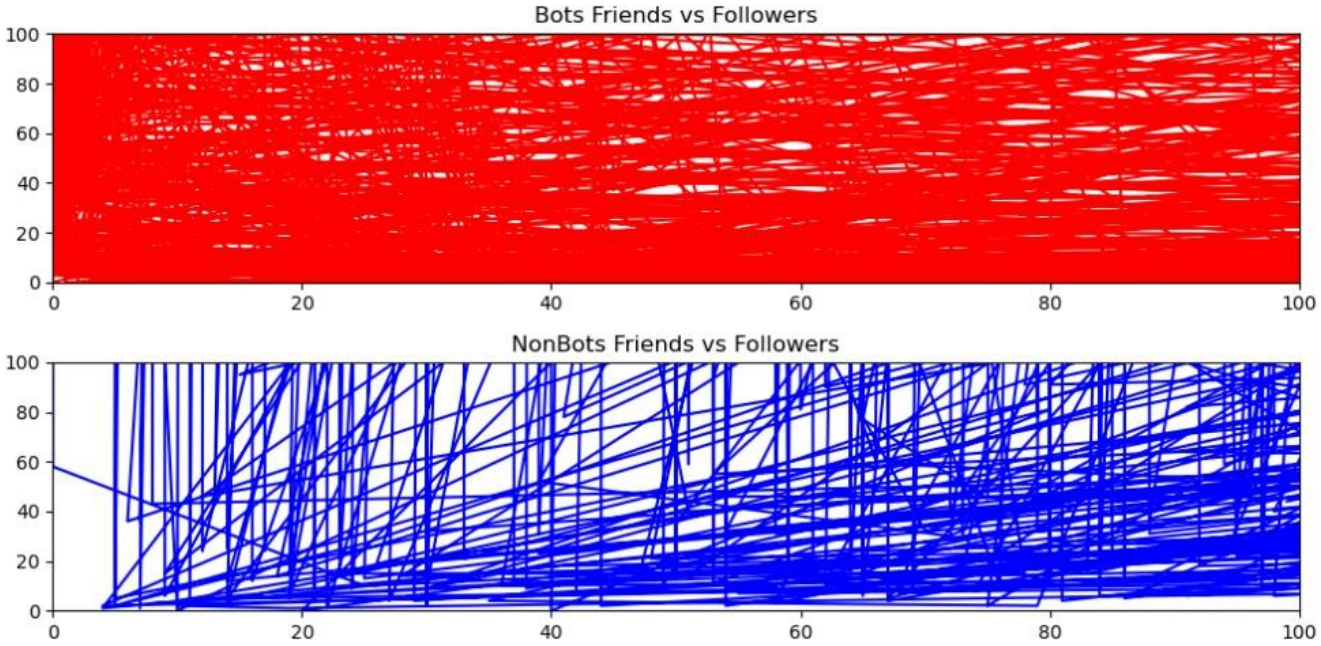
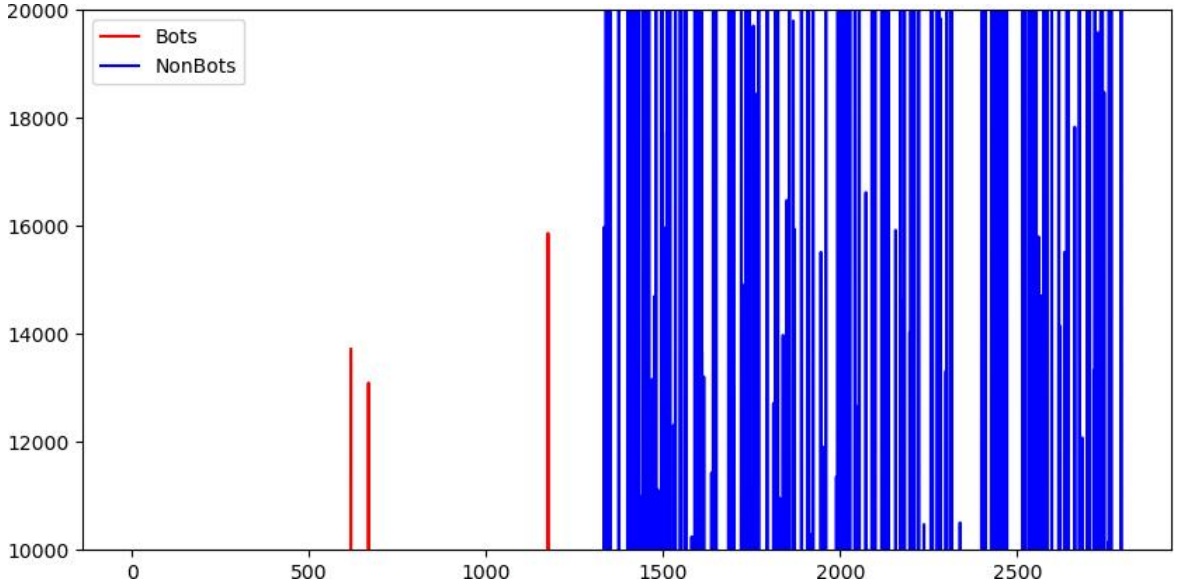


Fig 8 : Bot vs Non Bot Followers



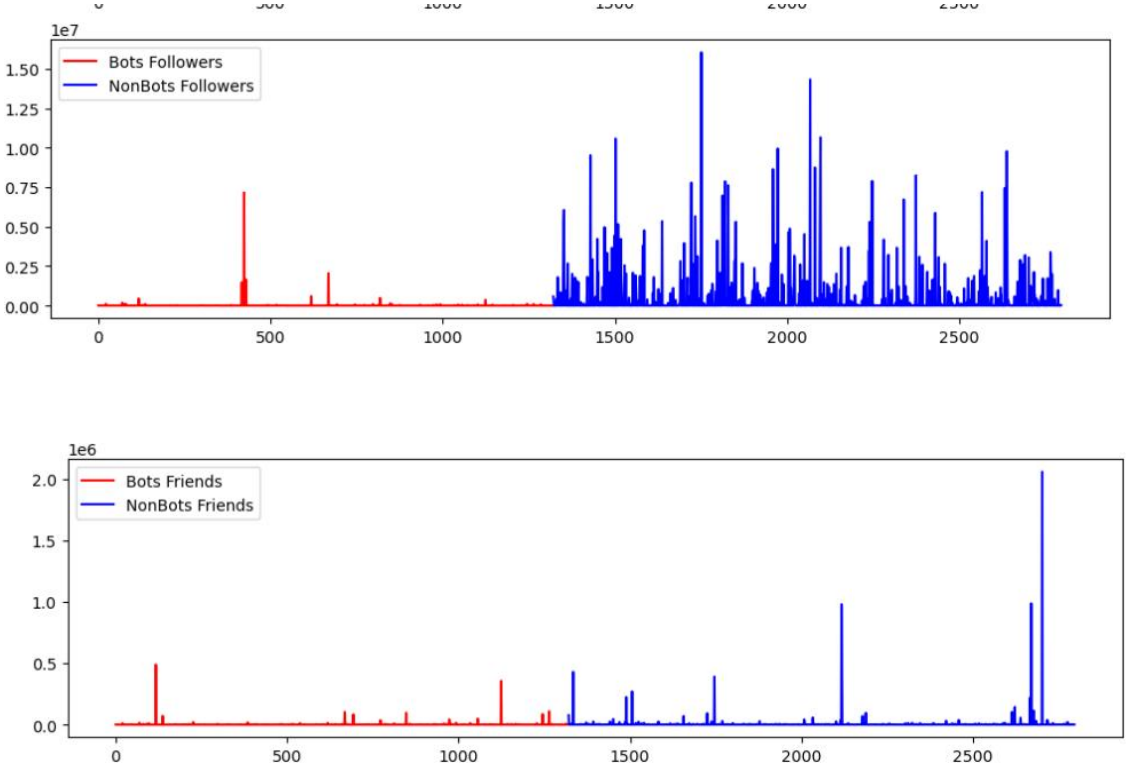


Fig 9:Bot vs Non Bot Friends

4.4Model:

To build our model, we first trained our data set on different algorithms to find the most optimum algorithm to go forward with. The algorithms used were as follows:

1)Decision tree

2)Logistic regression

3)KNN classifier

4) Bag of words

5) Naive Bayes

6)Random Forest

The data set was trained on these Six different algorithms. Before passing the data set to these algorithms, the data set was pre-processed. Then the most efficient algorithm was selected to be the model for our system. We found the most efficient and accurate algorithm was the algorithm. Bag of wordst is a Supervised Algorithm. This generally provides higher accuracy than the normal decision tree as was observable in our experiment too. It used different attributes like friends count, followers count, listed count, etc. to predict the result as a bot or a genuine account. Eighty percentage of the data set was used to train the model and the rest twenty percentage to test the trained model. It was found that the most important attribute in the list of attributes is the 'geo location' attribute. Most of the time, the model predicted the account as a bot whenever the verified attribute was 'FALSE'. It's not necessary for accounts specified as 'FALSE verified' to be a bot

and vice versa. Also, most of the bot accounts predicted were not popular. That's because these bot account generally remain in a stealth mode. The model built using the algorithm was quite efficiently able to predict the account as a bot or a genuine account.

4.5 Performance Evaluation Parameters

The performance of the classification methods can be found by using Accuracy, F-Score, Cross-entropy, Recall, and Precision. These parameters are helpful to evaluate the performance of supervised machine learning algorithms, based on the element from a matrix known as the confusion matrix or contingency table. A confusion matrix is typically used for allowing visualization of the performance of an algorithm. From the classification viewpoint, terms such as 'True Positive (TP)', 'False Positive (FP)', 'True Negative (TN)', 'False Negative (FP)' are used to compare labels of classes in this matrix, as shown in Table 1. True Positive represents positive reviews that were classified as positive by the classifier, whereas False Positive is predicted as negative but is actually classified as negative. Conversely, True Negative represents negative reviews that were classified as negative by the classifier, whereas False Negative is predicted as positive actually classified as negative. According to the data of the confusion matrix, precision, recall, f-measure, and accuracy are used for evaluating the performance of classifiers.

Precision

This is defined as the ratio of the number of reviews correctly classified as positive to the total number of reviews that are truly positively classified.

$$\textbf{Precision} = \frac{TP}{TP+FP}$$

Recall

This is defined as the ratio of the number of reviews correctly classified as positive to the total number of reviews that are classified positively.

$$\textbf{Recall} = \frac{TP}{TP+FN}$$

Accuracy

This is the ratio of the reviews that are correctly classified to the total number of reviews.

Accuracy = TP+TFTP+FP+TN+FN(9)

F-score

This is a combined measure for precision and recall.

F-Score = (Precision*Recall)(Precision+Recall)(10)

Cross-entropy

Cross-entropy or log loss is used further to measure the performance of the classification models. The output of log loss is a probability value between 0 and 1.

Chapter 5

SYSTEM REQUIREMENT SPECIFICATIONS

Prerequisites exam is simple for undertaking improvement. Prerequisites need to be archived, vast, quantifiable, and testable and characterised to some extent of detail adequate for framework plan. Necessities can be engineering, underlying, social, realistic, and beneficial. A software requirements specification (SRS), product requirements specific is a far attaining depiction of the planned reason and the weather for programming being worked on.

Functional Requirements

The tools to execute the Python programs can be many, among that we can go with Visual Studio, Anaconda Navigator (Jupyter Notebook) or any IDLE based on Python. The online tool from Google can be an effective solution towards the execution of Python coding.

5.1Approach 1: Microsoft Visual Studio

This is an integrated development environment (IDE) from the Microsoft Organization which is basically used for the development and execution of the programs. More efficient and powerful applications such as Website Development, Mobile Application Development and other Web- based Apps can be designed very effectively and easily. It supports for productive design, Development of Cross-platform Application and (Artificial Intelligence) AI based power tools.

The major contributions from this product are:

- Project Scaling ability and support for the complexity
- .NET and C++ Platform to work with any code integrity
- Real-time coding experience
- Automatic Code Writing tool (IntelliCode)
- Sharing Multiple Screen on Single Platform
- Unified cloud Support

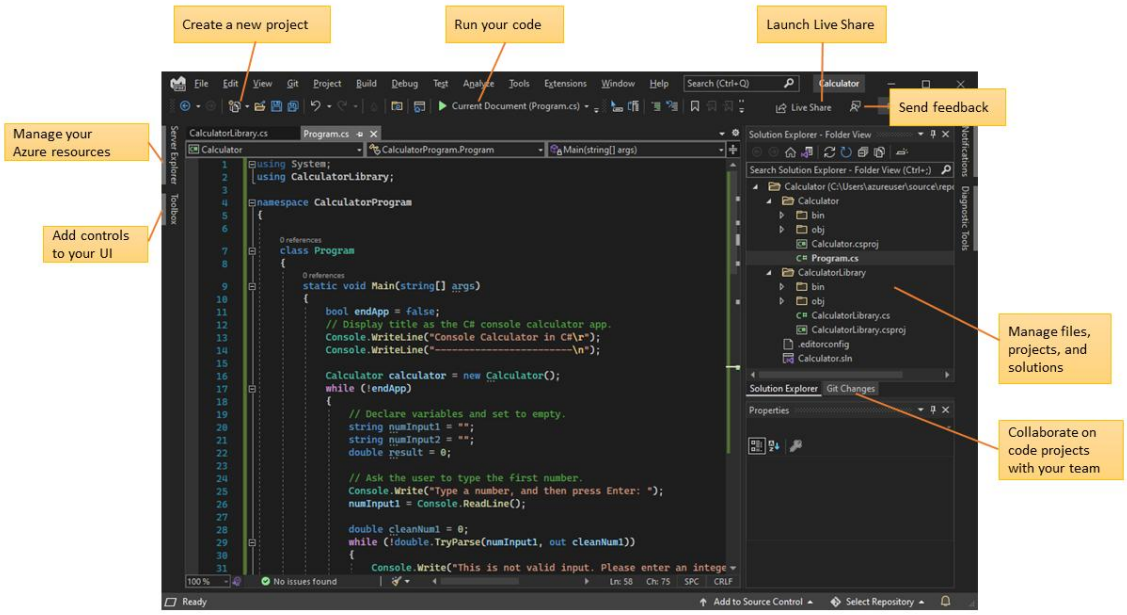


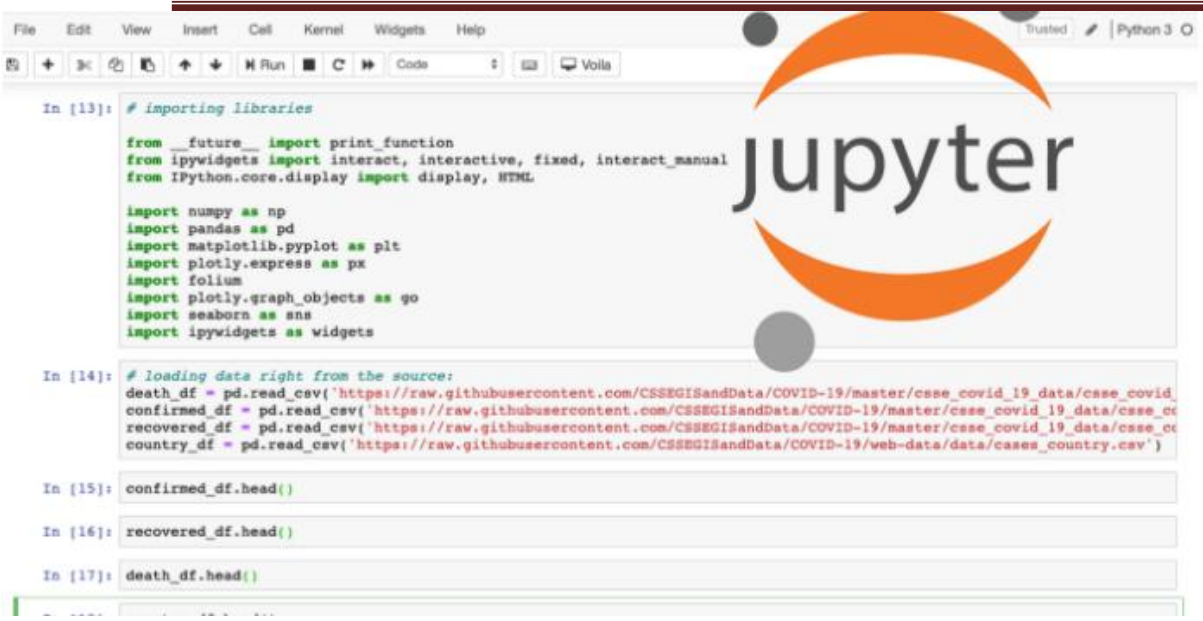
Fig 10: Visual Studio

5.2 Approach 2: Jupyter Notebook (Anaconda Navigator)

This tool is also known as IPython Notebook, and it is Open-Source Distribution Software and provides the platform for development of web applications, computational interactive and specific environment for the users to create notebook documentations. It supports for individual code execution, browser based interoperability, can plot various graphs using python libraries and also support for many open source libraries like Bootstrap, jQuery, Tornado, Matplotlib, Seaborn and others.

The features of Jupyter Notebook can be listed as:

- Flexible Notebook Interface
- Useful tool in Machine learning, Deep learning and Ai based Application and model Design.
- Creating and sharing the computational Documents.



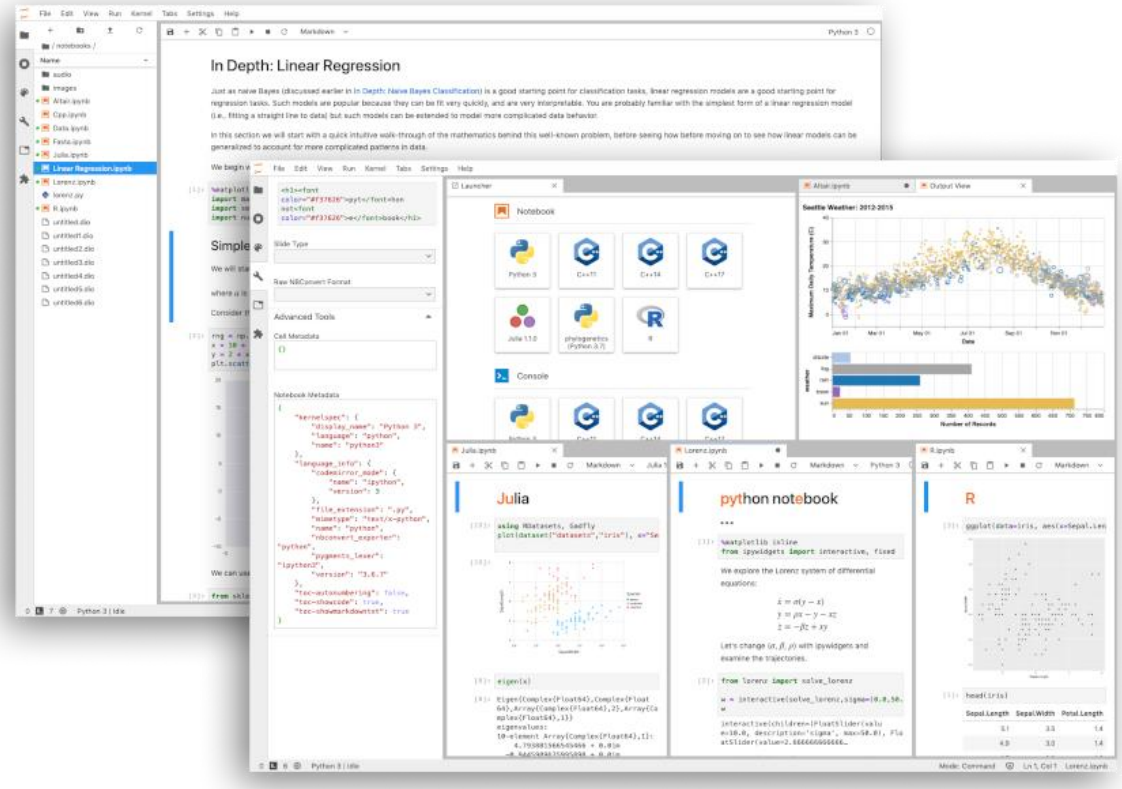


Fig 11 : Jupyter Notebook

5.3 Approach 3: Python IDLE

Python IDLE (Python Integrated Development and Learning Environment) help is writing the code very effectively and efficiently and helpful tool to the Python learning who wants to start from the scratch and beginners can have an advantage to execute the code easily. This is a powerful interpreter and compiler to run the code.

It's an Interactive Interpreter also known as shell, which executes the python written code, reads the input, evaluate the statements and print the output on the standard output screen provided.

File Editor Help to edit the code, save the program in text files and store as .py file.

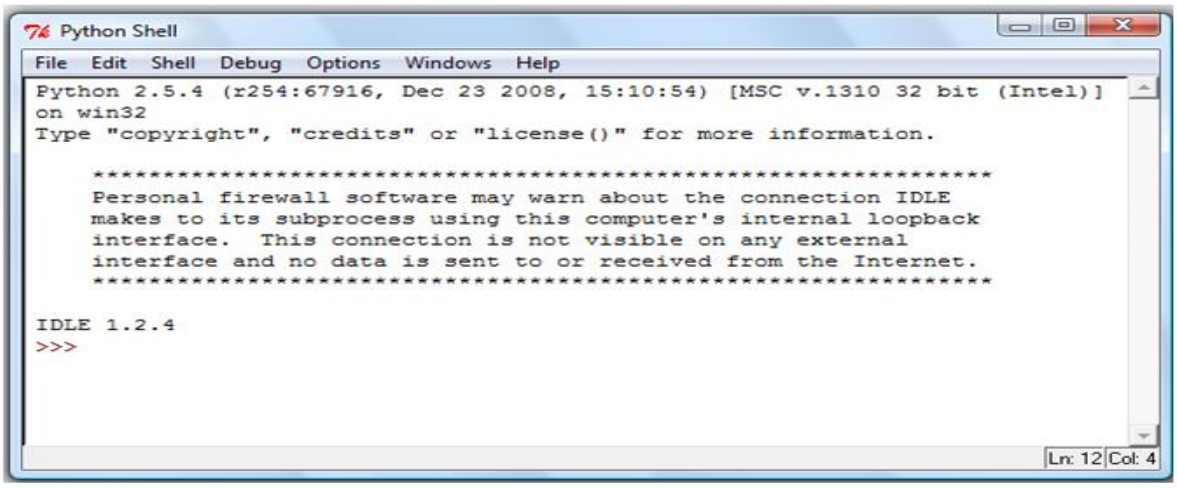


Fig 12 :Python IDE

5.4 Approach 4: Google Colab

Google Colab, also called as Colab in short is a powerful Machine Learning, Deep Learning and Data Analysis Tool that allows mixing the Python script along with text document. Rich support for Plotting the graphs, Diagram, Charts, Import Images, HTML Tags Support and LATEX format API conversions. Additional functional is it works on cloud model where document can be accessed and run on any platform independent of framework design and operating system. The runtime support for Virtual Hard Disk space and 12GB of RAM to execute the application is very excited feature of Colab. The uploading of files is very easy in this application so that it connects to the runtime.

Some of the important feature is:

- Remote Desktop Connection
- Runtime Environment
- Dataset Upload Features
- I/O operations and Operating System API Support
- General Processing Unit (GPU) availability

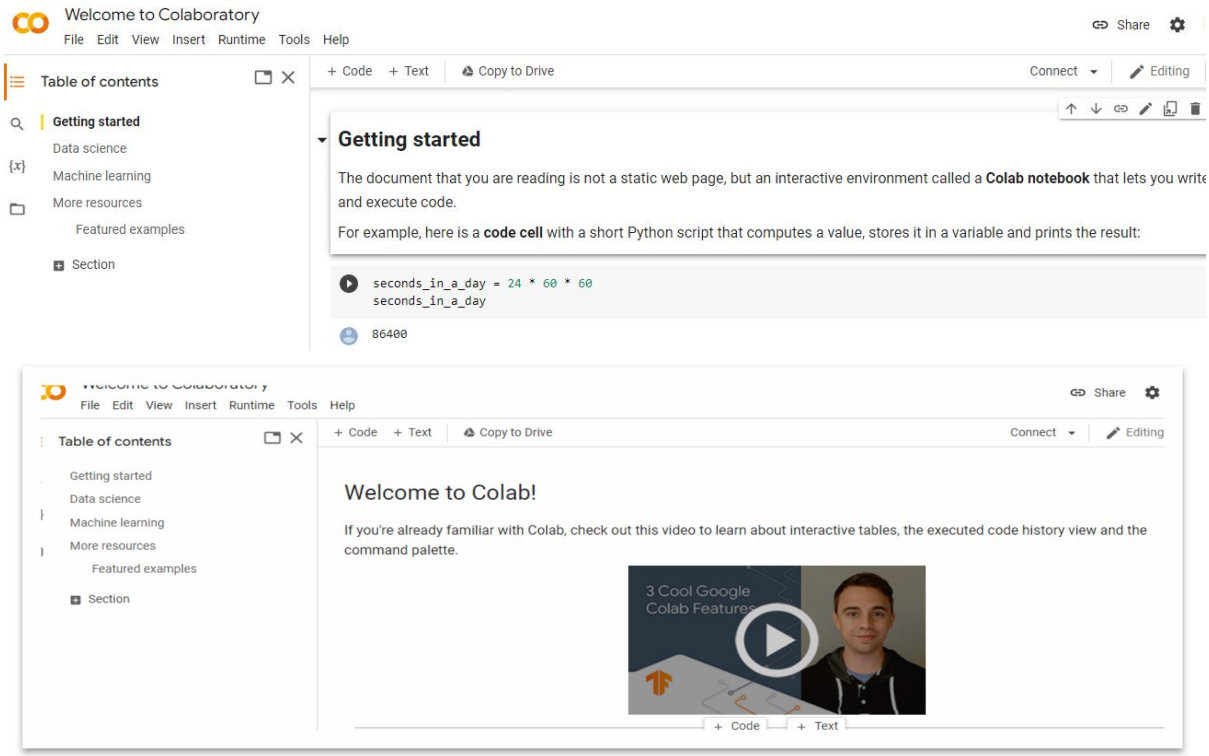


Fig 13 : Google colab

Chapter 6:

Code Snippet

```
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
import time
import sklearn.metrics as metrics


filepath = '/root/Documents/MachineLearning-Detecting-Twitter-Bots/FinalProjectAndCode/
kaggle_data/'
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]

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')
plt.plot(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')
plt.plot(nonbots.friends_count, nonbots.followers_count, color='blue', label='NonBots')
plt.xlim(0, 100)
```

```
plt.xlim(0, 100)
plt.ylim(0, 100)
plt.tight_layout()
plt.show()


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

df.corr(method='spearman')

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

#filepath = 'https://raw.githubusercontent.com/jubins/ML-TwitterBotDetection/master/FinalCode/
kaggle_data/'
filepath = '/root/Documents/MachineLearning-Detecting-Twitter-Bots/FinalProjectAndCode/
kaggle_data/'
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)
```

```
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']

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')
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
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')
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')

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  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_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 = '/root/Documents/MachineLearning-Detecting-Twitter-Bots/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',encoding = "ISO-8859-1",engine="python"
,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)



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()
```

CHAPTER 7

SYSTEM TESTING

Testing is extremely important for quality assurance and ensuring the products reliability. The success of testing for programmer flaws in largely determined by the experience. Testing might be a crucial component in ensuring the proposed systems quality and efficiency in achieving its goal. Testing is carried out at various phases during the system design and implementation process with the goal of creating a system that is visible, adaptable and secure.

Testing is an important element of the software development process. The testing procedure verifies whether the generated product meets the requirements for which it was intended.

7.1 Test objectives

• Testing may be a defined as a process of running a programme with the goal of detecting a flaw.

• An honest case is one in which there is a good chance of discovering a mistake that hasn't been detected yet.

• A successful test is one that uncovers previously unknown flaw. If testing is done correctly, problems in the programme will be discovered. Testing cannot reveal whether or not flaws are present. It can only reveal the presence of software flaws.

7.2 Testing principles

A programmer must first grasp the fundamental idea that governs software testing before applying the methodologies to create successful test cases. All testing must be able to be tracked back to the customer's specification.

7.3 Testing design

Any engineering product is frequently put to the test in one of two ways:

7.3.1 White Box Testing

Glass container checking out is every other call for this kind of checking out. By understanding the necessary characteristic that the product has been supposed to do, checking out is regularly accomplished that proves every characteristic is absolutely operational at the same time as additionally checking for faults in every characteristic. The take a look at case layout technique that leverages the manage shape of the procedural layout to create take a look at instances is used on this take a look at case

7.3.2 Black Box Testing

Tests are regularly finished on this checking out via way of means of understanding the indoors operation of a product to make certain that each one gears mesh, that the indoors operation operates reliably in step with specification, and that each one inner additive had been nicely exercised. It is in most cases worried with the software's practical needs.

7.4 Testing Techniques

A software testing template should be established as a set of stages in which particular test suit design techniques are defined for the software engineering process.

The following characteristics should be included in every software testing strategy:

¬ Testing begins with the modules and extends to the mixing of the full computer-based system.

¬ At different periods in time, different testing approaches are applicable.

¬ Testing is carried out by the software's developer and an independent test group. A software developer can use a software testing strategy as a route map. Testing might be a collection of actions that are prepared ahead of time and carried out in a methodical manner. As a result, a software testing template should be established as a set of stages in which particular test suit design techniques are defined for the software engineering process. The following characteristics should be included in every software testing strategy: Testing begins at the module level and progresses to entire computer-based system are mixing.

¬ At different periods in time different testing approaches are applicable.

¬ Testing is carried out by the software's developer and a separate test group.

7.5 Levels of Testing

Testing is frequently omitted at various stages of the SDLC. They are as follows:

7.5.1 Unit Testing

Unit testing checks the tiny piece of software that makes up the module. The white box orientation of the unit test is maintained throughout. Different modules are tested alongside the requirements created throughout the module design process. The aim of unit testing is to inspect the inner logic of the modules, and it is used to verify the code created during development phase. It is usually done by the module's developer. The coding phase is sometimes referred to as coding and unit testing because of its tight association with coding. Unit tests for many modules are frequently run in simultaneously.

7.5.2 Integration Testing

Integration testing is the second level of quality assurance. This type of testing integrates different components in program like modules also to check the interface problems. Many tested modules are combined into subsystems and tested as a result of this. The purpose of this test is to see if all of the modules are properly integrated. Integration testing may be divided into three categories:

• Top-Down Integration:

Top-Down integration is a method of gradually constructing a Programme structures. Modules are connected by working their way down the control Hierarchy, starting with the module having the most control. Bottom-Up Integration:

Construction and testing using autonomous modules begin with Bottom-up integration, as the name suggests.

• Regression Testing:

It is a subset of previously executed tests to ensure that Modifications have not propagated unexpected side effects during this competition of an Integration test strategy.

7.5.3 Functional Testing: The business and technical requirements, system documentation, and user guides all specify that functional tests must be conducted to ensure that the functions being tested are available. The following items are the focus of functional testing:

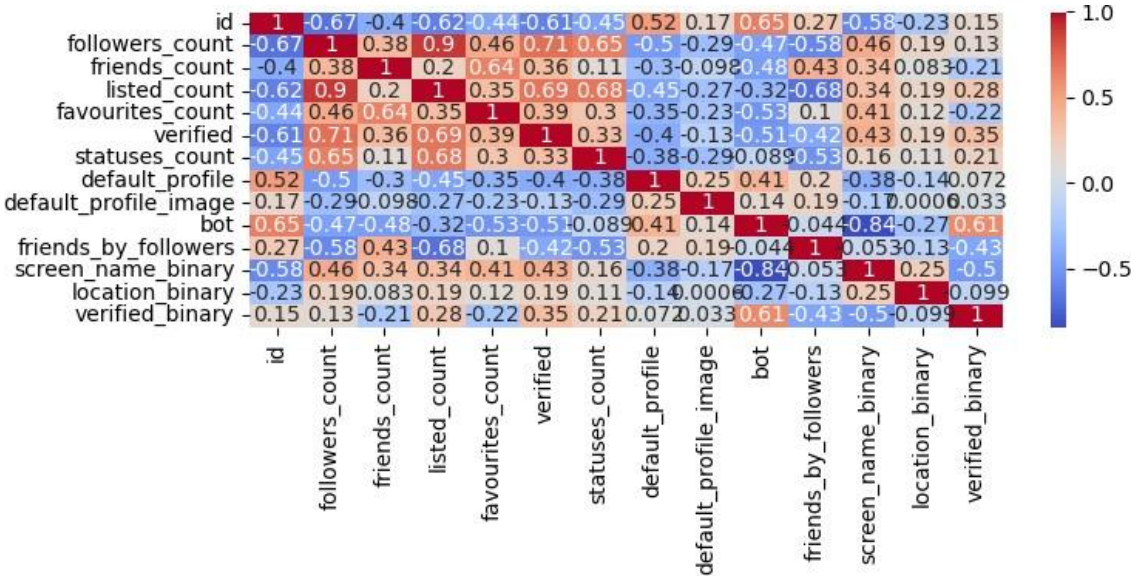
7.5.4 Validation Testing Validation may be characterized in a lot of ways; however, one easy definition is that validation is a hit whilst software program plays in a manner that clients may fairly expect. The affordable expectation is said with inside the software program requirement specification that is a record that lists all the software program's user-seen attributes. Validation standards are a segment of the specification. The statistics on this component serves as the premise for the validation trying out strategy.

7.5.5 Alpha Testing

Software developer can't know how a customer will utilize a programme ahead of time. Instructions to be utilized could be misconstrued, a peculiar combination of knowledge could be employed on a regular basis, and a result that was clear to the tester could be unclear to a field user. It's impractical to conduct a formal acceptance test with all users if the programmed is designed as a product that will be used by many people. Most software developers utilize alpha and beta testing to detect bugs that only the most experienced users seem to be aware of. At the developer's premises, a customer does the trial.

CHAPTER 8

RESULT



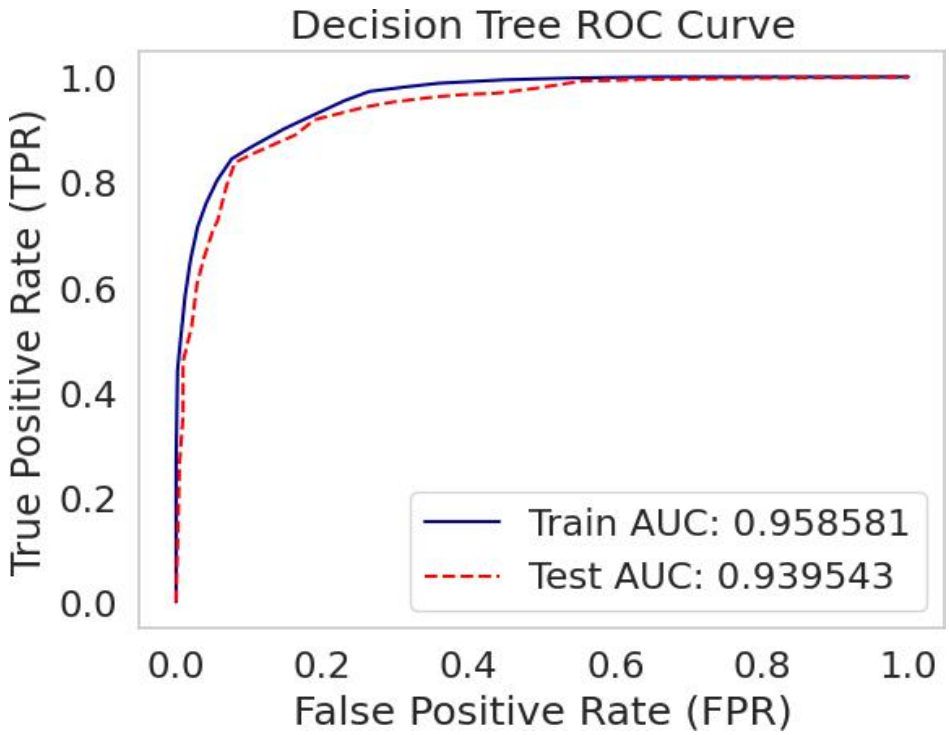


Fig 14 : ROC of Decision Tree

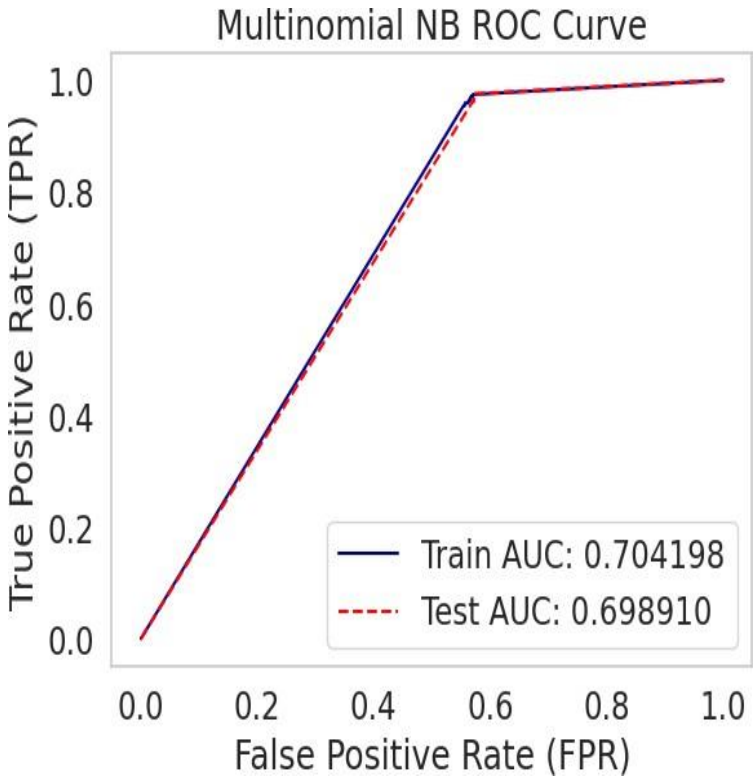


Fig 15: ROC of Multinomial Naive Bayes

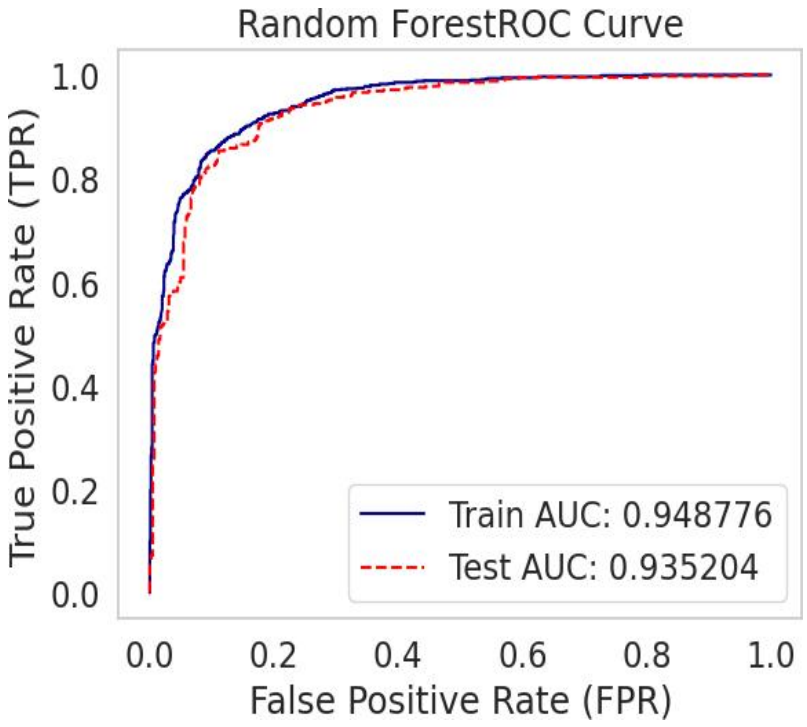


Fig 16: ROC of Random Forest

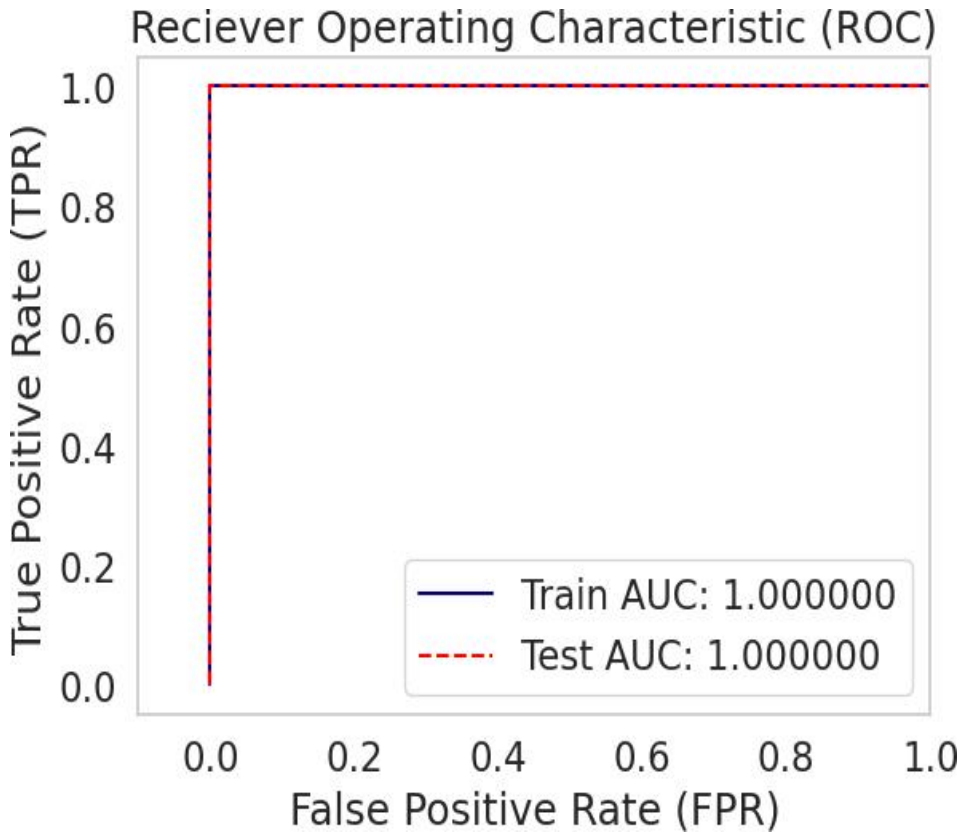


Fig 17 : ROC of Proposed Bag of Word Classifier

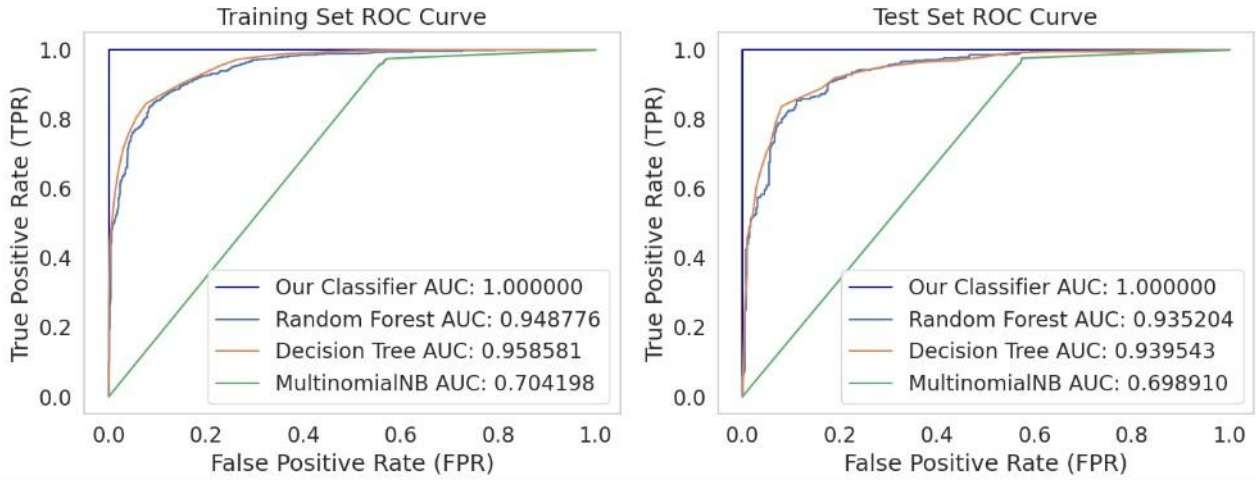


Fig 18 : Test and Train of Different Classifier

CONCUSION

Our end goal is to build a system for Twitter users to identify whether an account is a bot or not. To achieve this goal, we have designed and implemented a system that takes an account's username as input and classifies it as a human user or a bot. The praposed classification algorithm was used to build the model which gives an accuracy rate of . Our idea is to allow the users to check whether the information from an unverified and unknown source is a bot or not before blindly spreading it without more research. Spam bots are usually used to influence people'sopinions on various topics with misinformation and rumors. By having this system, we can prevent such influence.

REFERENCES

1. Stefan Wojcik, "Bots in the Twitter sphere": http://www.pewinternet.org/2018/04/09/bots-in the-twittersphere/

2. Chris Baraniuk, "How Twitter Bots Help Fuel Political Feuds": https://www.scientificamerican.com/article/howtwitter-bots-help-fuel-political-feuds/

3. Chengcheng Shao et al., "The spread of low-credibility content by social bots": https://arxiv.org/pdf/1707.07592.pdf

4. The tweepy Python library: http://www.tweepy.org

5. Twitter's developer resources to learn about the API: https://developer.twitter.com

6. Asbjan Ottesen Steinskog et al., "Twitter Topic Modeling by Tweet Aggregation": http://www.aclweb.org/anthology/W17- 0210 Last Modified: June 25, 201

7. Clark, E.M.; Williams, J.R.; Jones, C.A.; Galbraith, R.A.; Danforth, C.M.; Dodds, P.S. Sifting robotic from organic text: A natural language approach for detecting automation on Twitter. J. Comput. Sci. 2016, 16, 1–7. [CrossRef]

8. He, D.; Liu, X. Novel competitive information propagation macro mathematical model in online social network. J. Comput. Sci. 2020, 41, 101089. [CrossRef]

9. Jain, S.; Sinha, A. Identification of influential users on Twitter: A novel weighted correlated influence measure for COVID-19. In Chaos, Solitons Fractals; Elsevier: Amsterdam, The Netherlands, 2020; Volume 139, pp. 1–8. [CrossRef]

10. Przybyła, Piotr. (2019). Detecting Bot Accounts on Twitter by Measuring Message Predictability .https://www.aclweb.org/anthology/R191065.pdf [2]. de Andrade, Norberto & Rainatto, Giuliano & Lima, Fonttamara & Silva Neto, Genésio & Paschoal, Denis. (2019). Machine Learning and Bots detection on Twitter. International Journal of Science and Research (IJSR).

11. 8. 001-011. [3]. Ranjana Battur & Nagaratna Yaligar: Twitter Bot Detection using Machine Learning Algorithms

13. Jurgen Knauth: Language-Agnostic Twitter Bot DetectionKantepe and M. C. Ganiz, "Preprocessing framework for Twitter bot detection," 2017International Conference on Computer Science and Engineering (UBMK), Antalya, Turkey, 2017, pp. 630-634, doi: 10.1109/UBMK.2017.8093483.

14. Wetstone, Jessica and Sahil R. Nayyar. “I Spot a Bot: Building a binary classifier to detect bots on Twitter.” (2017). http://cs229.stanford.edu/proj2017/final-reports/5240610.pdf

15. Chavoshi, Nikan, H. Hamooni and A. Mueen. “On-Demand Bot Detection and Archival System.” Proceedings of the 26th International Conference on World Wide Web Companion (2017): n. pag. https://api.semanticscholar.org/CorpusID:8175106

16. Abou Daya, Abbas & Salahuddin, Mohammad & Limam, Noura & Boutaba, R. (2020). BotChase: Graph-Based Bot Detection Using Machine Learning. IEEE Transactions on Network and Service Management. PP. 10.1109/TNSM.2020.2972405. https://www.researchgate.net/publication/338779142_BotChase_Graph_Based_Bot_Detection_Using_Machine_Learnin