CHAPTER-1

INTRODUCTION

Twitter bot is used to produce automated posts, follow Twitter users or serve as spam to entice clicks on the Twitter micro blogging service. In this project, we will use Machine Learning techniques to predict weather an account on Twitter is a Bot or a real user. We will be performing feature engineering, along with feature extraction - selected features out of 20 like name of the twitter account, description, profile image, inactivity, twitter posts etc. which helped us to identify whether an account is bot or non bot. We will implement various algorithms and lastly implement our own custom classification algorithm in order to achieve high accuracy considering more attributes.

Project aims to detect whether an account in twitter is a bot or a real human. Future implementations will be to detect if bot is a good or bad bot. It's a social networking service where users post and interact with messages, "tweets" restricted to 140 characters. There are 310M monthly active twitter users and a total of 1.3 billion accounts have been created and 500M tweets per day. Since it's such an influential platform people have developed Twitter bots. These bots are used to increase number of followers, retweeting, spamming etc. and there are about 48M bots today. Tweepy a REST API is a python dataset to extract data from twitter accounts(CSV file). This dataset has 1056 bots and 1176 users According to study released by University Of California 24% of tweets are done by Bots. Limitation being the coverage of good and bad bot concept isn't detected by our custom classification algorithm.

Software Library

It is a collection of precompiled modules designed to perform a specific task. It aids in code reuse.

Libraries Used

The Python Standard Library provides us with rich libraries to implement varied functions. It contains in-built modules that give access to the system functionalities like File operations.

OpenCV

It is a widely used open source computer vision library. It provides us with thousands of algorithms that are used in general tasks like object detection and identification, object classification, Image recognition from databases, Image correction etc. It is primarily written in C++. There are versions that work well with Python and Java.

Python Image Library

It is a powerful tool which adds image processing options to the Python interpreter. It is capable of performing Image processing functionality like modifying color channels (color space conversions), rotation, resizing and other transforms.

NumPy

Python library used for computational purposes.

Pandas

Is a software library written for the Python programming language for data manipulation and analysis.

Movie.py

It is a python module which aids in video processing/editing operations like trimming, merging and subtitle insertions. It can read and write in different formats like .mp4, .avi, etc.

Sklearn

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines

CHAPTER-2

PROJECT DEFINITION

The aim is to build a machine learning algorithm using several modules of machine learning and detect whether an twitter account is real user or a bot

Following are the Implementation stages involved:

1. Extraction of Data from Kaggle
2. Extraction of data from twitter API called Tweepy
3. Conduct analysis for these two datasets using Decision trees, Random forest, Multinomial Naive Bayes.
4. Build our own Custom Classification Algorithm.
5. Testing.

CHAPTER-3

LITERATURE SURVEY

Twitter is an online news and social networking service where users post and interact with messages, “tweets” restricted to 140 characters. There are 310M monthly active twitter users and a total of 1.3 billion accounts have been created and 500M tweets per day Since it’s such an influential platform people have developed Twitter bots. These bots are used to increase number of followers, retweeting, spamming etc. and there are about 48M bots today.

3.1 Methods for Text Localization and Detection

Tweepy a REST API is a python dataset to extract data from twitter accounts.(CSV file)

This dataset has 1056 bots and 1176 users. According to study released by University Of California 24% of tweets are done by Bots.

3.2 Methods Adopted By the Project

- Decision Trees
- Random Forest
- Multinomial Naive Bayes
- Custom Classification Algorithm

3.3 Other Attributes

Types Of Attributes

•1)Integers:ID,Friends,Followers,Favourites,Statuses,listed count(tagging people)

•2) String: Name,Location,Description,URL,Language

•3)Boolean: Verified Account, Default Profile

•4)Date: CreatedAt

•5)JSON:Status(all tweets by the user in a json format)

•Other Derived attributes like age, account has name bot in it

CHAPTER-4

CUSTOMER REQUIREMENTS SPECIFICATION

4.1 Introduction

4.1.1 Scope

Project aims to detect whether an account in twitter is a bot or a real human. Future implementations will be to detect if bot is a good or bad bot. Twitter is an online news and social networking service where users post and interact with messages, “tweets” restricted to 140 characters. There are 310M monthly active twitter users and a total of 1.3 billion accounts have been created and 500M tweets per day. Since it’s such an influential platform people have developed Twitter bots. These bots are used to increase number of followers, retweeting, spamming etc. and there are about 48M bots today. Tweepy a REST API is a python dataset to extract data from twitter accounts(CSV file). This dataset has 1056 bots and 1176 users According to study released by University Of California 24% of tweets are done by Bots. Limitation being the coverage of good and bad bot concept isn’t detected by our custom classification algorithm.

4.2 Product Perspective

Project aims to detect whether an account in twitter is a bot or a real human. Future implementations will be to detect if bot is a good or bad bot. Twitter is an online news and social networking service where users post and interact with messages, “tweets” restricted to 140 characters. There are 310M monthly active twitter users and a total of 1.3 billion accounts have been created and 500M tweets per day. Since it’s such an influential platform people have developed Twitter bots. These bots are used to increase number of followers, retweeting, spamming etc. and there are about 48M bots today. Tweepy a REST API is a python dataset to extract data from twitter accounts(CSV file). This dataset has 1056 bots and 1176 users According to study released by University Of California 24% of tweets are done by Bots. Limitation being the coverage of good and bad bot concept isn't detected by our custom classification algorithm.

4.2.1 User Characteristics

One of the important problems in social media platforms like Twitter is the large number of social bot accounts which are controlled by automated agents, generally used for malicious activities. These include directing more visitors to certain websites which can be considered as spam, influence a community on a specific topic, spread misinformation, recruit people to illegal organizations, manipulating people for stock market actions, and blackmailing people to spread their private information by the power of these accounts. Consequently, social bot detection is of great importance to keep people safe from these harmful effects. In this study, we approach the social bot detection on Twitter as a supervised classification problem and use machine learning algorithms after extensive data preprocessing and feature extraction operations. Large number of features are extracted by analysis of Twitter user accounts for posted tweets, profile information and temporal behaviors. In order to obtain labeled data, we use accounts that are suspended by Twitter with the assumption that majority of these are social bot accounts. Our results demonstrate that our framework can distinguish between bot and normal accounts with reasonable accuracy.

4.2.2 General Constraints, Assumptions and Dependencies

Hardware Limitations: Program should be run on systems that have 8GB RAM or more. Systems with GPUs are preferred. If not, the video processing could be time consuming.

Software Limitations: Python libraries are mandatory for the platform to run.

Assumptions:

A user having no profile image, more followers than friends, description with a link is more likely to be classified as a Bot. If there is a large count of such cases those are ignored to fetch high accuracy.

Dependencies:

- Works well with Linux and Windows.
- Install Python, pandas,numpy, sklearnlibraries.
- Laptop specifications: 8GB RAM, Windows 10, Intel core i5, at least 3GB HDD free.

Server specifications: Ubuntu 18.04, 32 GB RAM, at least 3GB HDD free.

4.2.3 Risks

- No classification between good and bad bots
- It just classifies bots and real users
- A user having no profile image, more followers than friends, description with a link is more likely to be classified as a Bot. Detecting such specific cases becomes a risk.

4.3 System Architecture

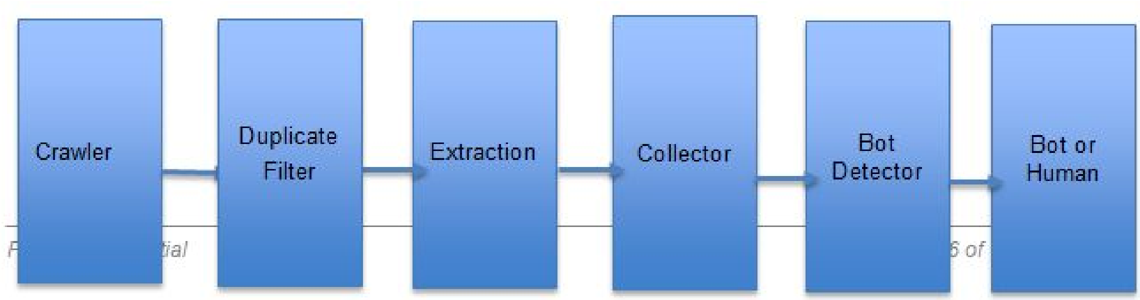


Fig 4.1 Architecture Diagram

Crawler – Crawler stage extracts data from the twitter API called Tweepy.

Duplicate Filter – Involves cleaning of data and removes duplicate account because that affects the accuracy rate.

Extraction – Extraction of attributes that we require in order to perform the analysis.

Collector – Analysis of the textual data like tweets using an algorithms like sentiment analysis.

Bot Detector – This stage finalizes an account to be a bot or a real user.

4.4 Requirements List

4.4.1 Data

Dataset	Size	Data Source	Data Description
Data Collection	156 KB	Twitter REST API	Shape:(100,20) Feature:19 Target: 1(Bot)
Training Set	5MB	Kaggle	Shape:(2797,20) Feature:19 Target: 1(Bot)
Test Set	1 MB	Kaggle	Shape:(575,20) Feature:19 Target: 1(Bot)

4.4.2 Exploratory Data Analysis

Step1: Identifying missing and imbalance in the data

Step 2: Feature Extraction

Step 3: Feature Engineering

Step 4: Dropping unnecessary attributes

4.5.0 External Interface Requirements

4.5.1 Hardware Requirements

- Windows 7, 8, 10, Server 2008, Server 2012, 64 bits
- Any CPU (Intel cores i5 or i7, Xeon recommended).

- A multicore processor, i5-i7 series
- At least 8GB of RAM

4.5.2 Software Requirements

Ubuntu 18.04 or Microsoft Windows operating system will be used during development process. The system will be implemented using Python language.

- Tweepy API is used to extract top 20 tweets and converted to csv file
- Jupyter Notebook version 5
- Numpy
- Python3
- Pandas
- Matplotlib
- Sklearn

The used software tools, their versions and sources are given in the table below:

Software Product & Version	**Source**
Ubuntu 18.04 Operating System	http://www.ubuntu.com/
Windows 7 and above	https://www.microsoft.com/en-in/windows
Python 2.7 and above	https://www.python.org/
Python Image Library	https://pillow.readthedocs.io/en/stable

	/

Table 4.7 Software Version and Source

4.5.3 Communication Interfaces

The following communication interfaces would be required:

- A working internet connection with a download link speed of at least 4 MBps and an upload link speed of at least 2 MBps.

4.6 User Interfaces

- A working internet browser (Google Chrome 70.0.3538 and above, Microsoft Edge 42.17134 and above, Internet Explorer 11 or Mozilla Firefox 63.0 and above).
- The screen resolution should be 1920x1080 (recommended).
- The webpage consists of two buttons namely upload and download. The upload button

4.7 Performance Requirements

This section shall specify both static and dynamic numerical requirements placed on the software or on human interactions with the software, as a whole.

Static numerical requirements may include:-

- The number of terminals to be supported
- The number of simultaneous users to be supported
- Number of files and records to be handled
- Sizes of tables and files
- Dynamic numerical requirements may include:-
- The number of locations to which the system caters
- The number of transactions
- Tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions
- Compatibility between heterogeneous environments (for e.g. Open Systems, Mainframes, Mid-range systems, etc.)

- Interconnection between various networks (LAN, WAN, Internet, etc.)
- All the requirements should be stated in measurable terms.

4.7.1 Static Requirements

· Number of files and records to be handled : 1 account at a time

4.7.2 Dynamic Requirements

· Image processing should be optimized so it should not take time more than 2 minutes

4.8 System Requirement Specification

4.8.1 Hardware Requirements

Particulars	**Client System**	**Web Server**	**Database Server**
OS	Any OS	Linux Server	Linux Server
RAM	At least 2 GB	16GB	16GB
Processor	At least dual core 2.3 GHz	Quad core i5 7th gen+ or equivalent.	Quad core i5 7th gen+ or equivalent.

Software	IE ver 11+ Edge ver 17+ Firefox Desktop ver 62+ Chrome Desktop ver 69+ macOS Safari ver 12+ iOS Safari ver 11.4+ Any Opera Mini browser. Chrome for android ver 69+ UC browser ver 11.8+ Samsung Internet ver 7.2+ iOS ver 8+		Python 3.7.0

4.8.2 SOFTWARE REQUIREMENTS

Technology	Why do we need it?	Advantages over existing ones	Cost of Scaling
Python Libraries	Use of modules	Enables better focus on main requirements	NA
Jupyter Notebook	Powerful way to write and iterate on your Python code for data analysis	NA	NA
Pandas, Numpy, Sklearn		NA	NA
Matplotlib		NA	NA

4.9 Special Characteristics

Specific requirements in this area could include the need to:

- Each module assigned with a specific functionality
- Restrict communications between some areas of a program
- Deadlock Avoidance

4.10 Help

A shared repository will be created on GitHub which contains a document which neatly explains the steps to be performed to execute the code.

4.11 Other Requirements

Certain requirements may, due to the nature of the software and the user organization, be placed in separate categories as indicated below:

4.11.1 Site Adaptation Requirements

None

4.11.2 Safety Requirements

It is advised to avoid using copyright content as it may lead to copyright infringement. In case of malfunction, service should automatically stop.

4.12 Packaging

Entire codebase will be available on GitHub. It could be pulled or downloaded as a zip file. A make file will be written to ensure the execution of all the modules smoothly.

CHAPTER-5

HIGH LEVEL DESIGN

5.1 Introduction

5.1.1. Overview

The project deals detecting whether an account is a bot or a real human. A times the bots extract human information which could be very dangerous and harmful. This is obtained by implementing few machine learning algorithms

5.1.2 Purpose

The goal of HLD or a high-level design document is to give the internal logical design of the actual program code. HLD describes the class diagrams with the methods and relations between classes and program specs. The code can then be developed directly from the low-level design document with minimal debugging and testing. Other advantages include lower cost and easier maintenance.

5.1.3 Scope

Project aims to detect whether an account in twitter is a bot or a real human. Future implementations will be to detect if bot is a good or bad bot. Twitter is an online news and social networking service where users post and interact with messages, “tweets” restricted to 140 characters. There are 310M monthly active twitter users and a total of 1.3 billion accounts have been created and 500M tweets per day. Since it’s such an influential platform people have developed Twitter bots. These bots are used to increase number of followers, retweeting, spamming etc. and there are about 48M bots today. Tweepy a REST API is a python dataset to extract data from twitter accounts(CSV file). This dataset has 1056 bots and 1176 users According to study released by University Of California 24% of tweets are done by Bots. Limitation being the coverage of good and bad bot concept isn’t detected by our custom classification algorithm.

5.2.0 Design Description

The application is built is a Bottom Up manner. The application comprises of modules for:

- Generation of frames from Video.
- Localizing subtitle text area in the frames.
- Inpaint Localized frames.
- Generate Inpainted video from frames.
- Extract audio from original video and add it to the output video.
- News Video: Real Time application. Dynamically place an overlay on the news scroll bar. Add audio to it.

5.2.1 Master Class Diagram

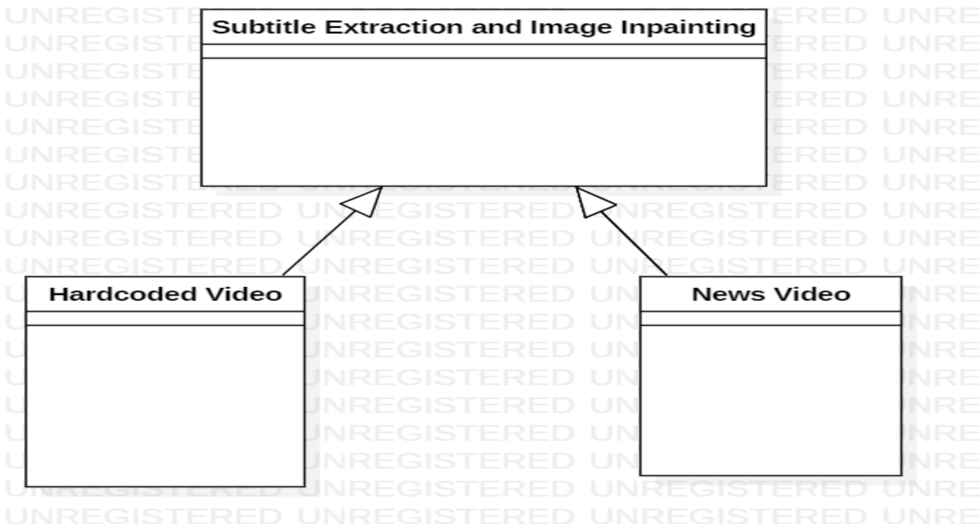


Fig 5.1 Master Class Diagram

5.2.2 Module 1 - Hardcoded Video

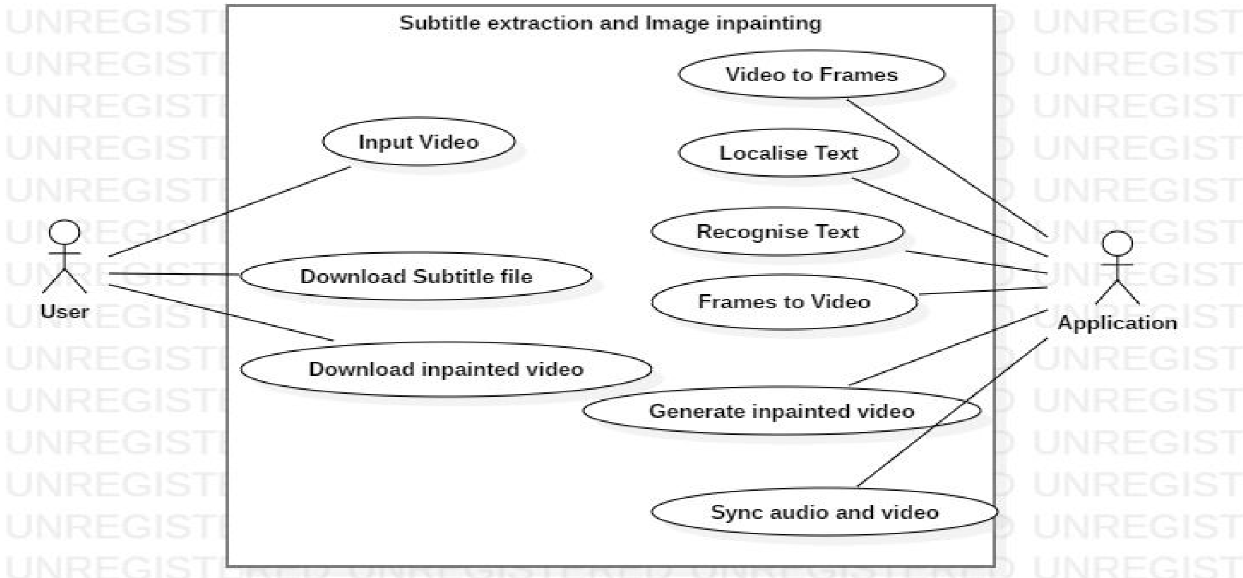
5.2.2.1 Description

Dataset contains videos that have hardcoded subtitles.

Procedure in Order:

- Generation of frames from Video.
- Localizing subtitle text area in the frames.
- Inpaint Localized frames.
- Generate Inpainted video from frames.
- Extract audio from original video and add it to the output video.

5.2.2.2 Use Case Diagram



5.2 Use Case Diagram

Use Case Item	Description
Video to Frames	The video is broken down into several thousand frames for making the processing easier.
Localize text	Subtitle text portion is localized and is marked for inpainting.
Recognize text	Use Tesseract OCR to detect text.
Frames to Video	Frames are converted to video. Audio is synced to it.
Inpainted video	The marked area in the image is inpainted using inpainting algorithms.

Table 5.1 Use Case item and description

5.2.2.2 Class Diagram

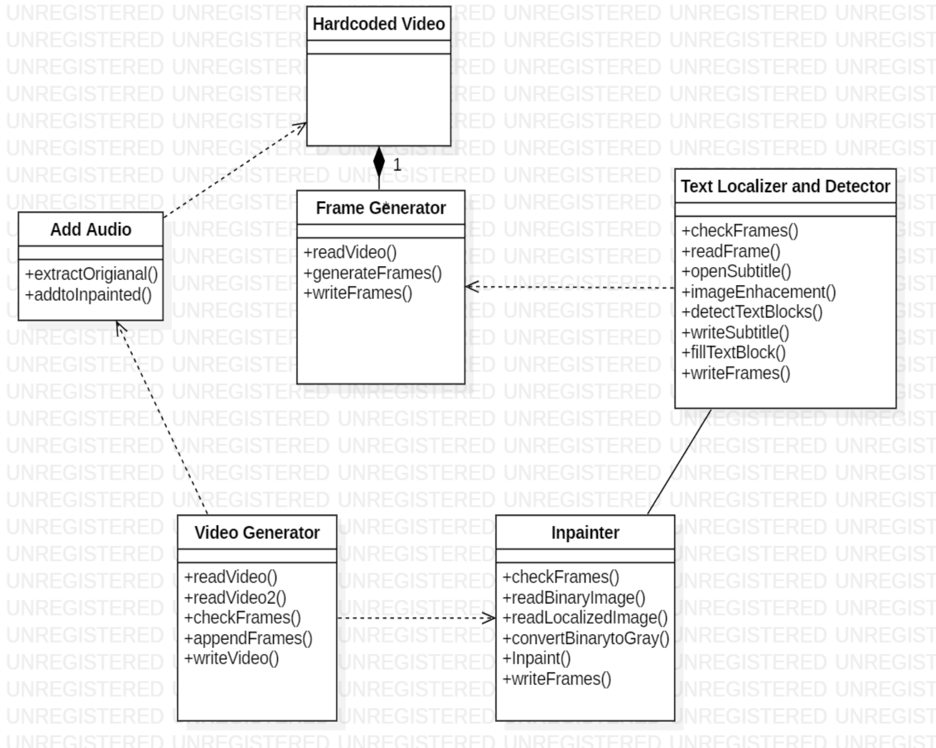


Fig 5.3 Class Diagram

5.2.2.2.1 Class Description

· Frames Generator class :

o Video selected is broken down into many frames.

o The frames per second is based on the use case.

o It allows a wider range of algorithms to be applied.

o Leads to faster processing.

o Reduces noise and distortion present in the video.

o Writes frames as output.

· Text Localizer Class and Detector Class:

o Dependent on Frame generator class.

o Subtitle text is localized.

- o Subtitles are written into the text file.
- o Text area is marked for removal and inpainting.
- o Text box is filled.
- o Write localized frames.
- · Inpainter Class:
- o Frames are checked
- o Read as Binary Image
- o Read localized image
- o Convert Binary image to Gray
- o Inpaint the frames
- o Write frames
- · Video Generator:
- o Dependency on Inpainter Class
- o Read Video
- o Read Video2
- o Check if frames available
- o Append Frames into video
- o Write Video
- o Dependency on Add Audio class
- · Add Audio
- o Dependency on Hardcoded Video class
- o Extracts audio from input video
- o Adds audio to the output of Inpainted class

5.2.2.3.2 Sequence Diagram

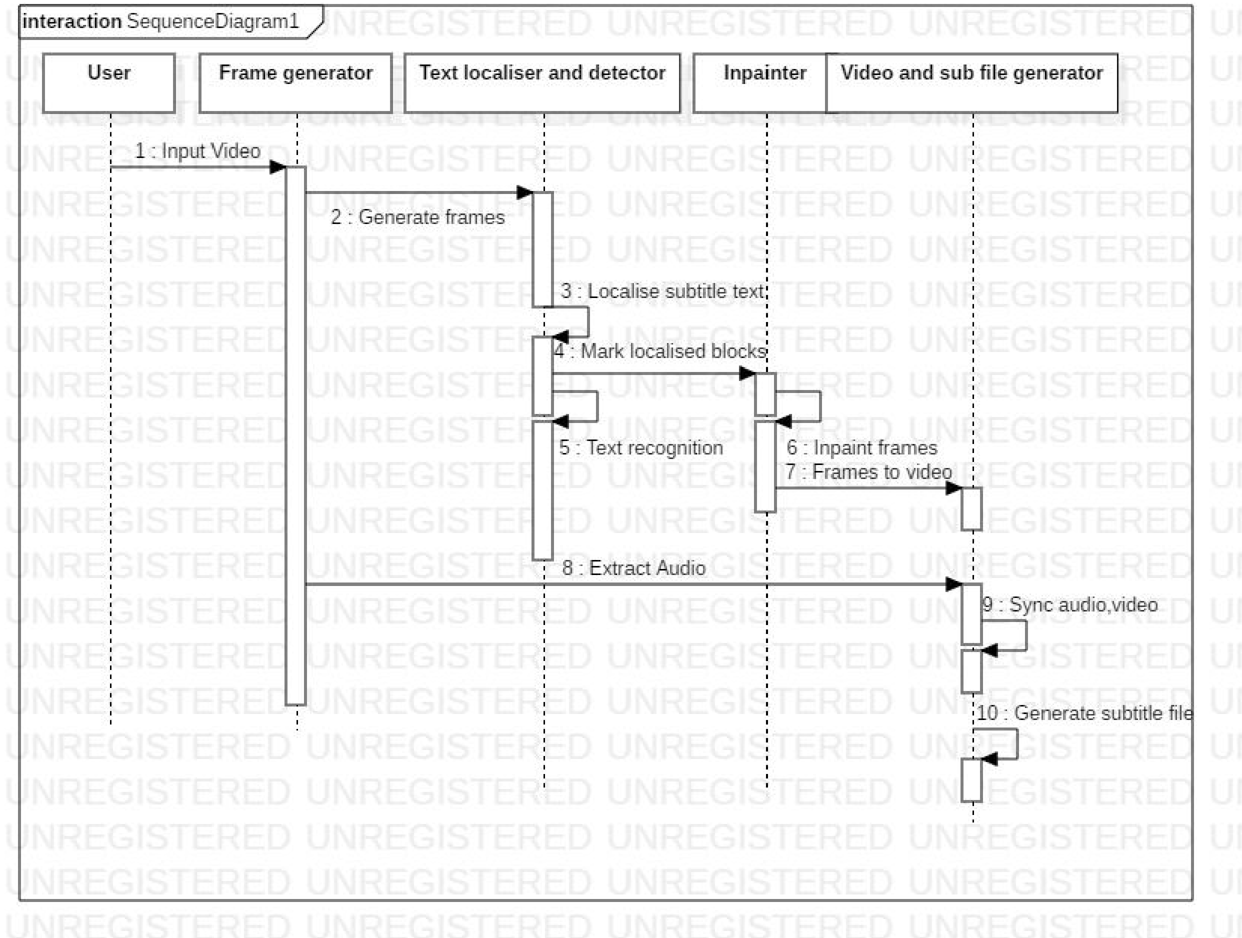


Fig 5.4 Sequence Diagram

5.2.2 Module 2 – News Video

5.2.3.1 Description

Dataset contains news videos taken from the same channel to maintain consistency across the dataset. The aim is to place a static overlay on the news scroll bar. This is a Real Time application of the Product.

5.2.2.2 Use Case Diagram

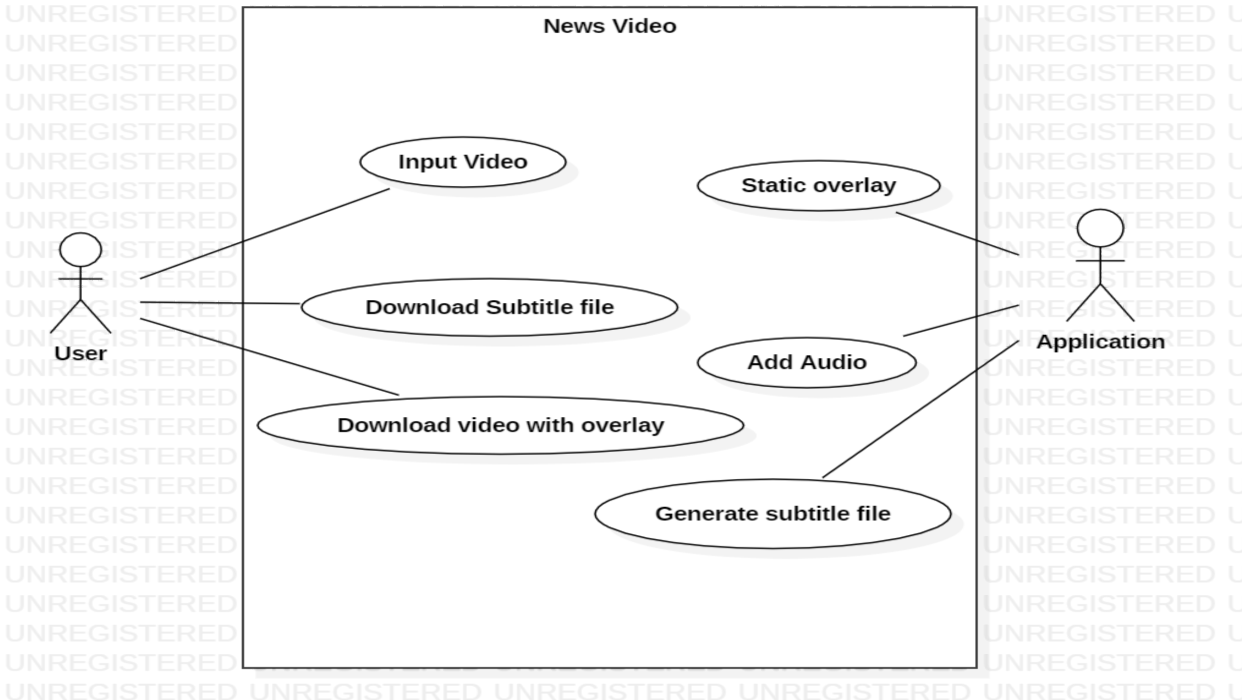


Fig 5.5 Use Case Diagram – Module 2

5.2.2.2 Class Diagram

5.3 ER Diagrams

Entity	Function	Type
Frame generator	Generate frames from videos for easy processing.	Input : .avi Output : .jpeg
Text Localizer	Identify location of subtitle text. Mark text blocks.	Input : .jpeg Output : .jpeg
Text Detector	Recognize subtitle text and store in text file.	Input : .jpeg Output : .txt
Inpainter	Inpaint the localized area to retain data integrity.	Input : .jpeg Output : .jpeg

Video generator	Create video from frames and sync audio.	Input : .jpeg Output : .avi

Table 5.2 ER Model

5.4 User Interface Diagrams

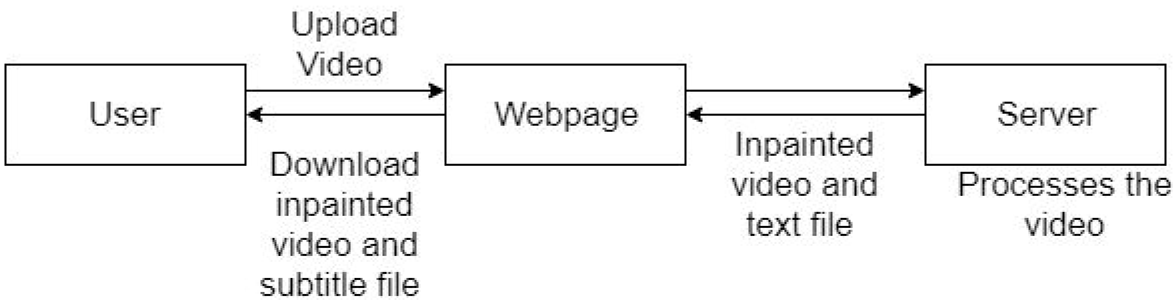


Fig 5.7 User Interface Diagram

5.5 Help

A shared repository will be created on GitHub which contains a document which neatly explains the steps to be performed to execute the code.

5.6 Alternate Design Approach

The product's main task is to extract subtitles and inpaint the image. The limitation arises when the subtitle text is of a different color. Frames are generated to improve the processing speed. Inpainting helps retain data integrity.

CHAPTER-6

LOW LEVEL DESIGN

6.1 Introduction

6.1.1 Overview

The project deals detecting whether an account is a bot or a real human. A times the bots extract human information which could be very dangerous and harmful. This is obtained by implementing few machine learning algorithms

6.1.2 Purpose

The goal of LLD or a low-level design document is to give the internal logical design of the actual program code. LLD describes the class diagrams with the methods and relations between classes and program specs. The code can then be developed directly from the low-level design document with minimal debugging and testing. Other advantages include lower cost and easier maintenance.

6.1.3 Scope

Project aims to detect whether an account in twitter is a bot or a real human. Future implementations will be to detect if bot is a good or bad bot. Twitter is an online news and social networking service where users post and interact with messages, "tweets" restricted to 140 characters. There are 310M monthly active twitter users and a total of 1.3 billion accounts have been created and 500M tweets per day. Since it's such an influential platform people have developed Twitter bots. These bots are used to increase number of followers, retweeting, spamming etc. and there are about 48M bots today. Tweepy a REST API is a python dataset to extract data from twitter accounts(CSV file). This dataset has 1056 bots and 1176 users According to study released by University Of California 24% of tweets are done by Bots. Limitation being the coverage of good and bad bot concept isn't detected by our custom classification algorithm.

6.2.0 Design Description

Crawler – Crawler stage extracts data from the twitter API called Tweepy.

Duplicate Filter – Involves cleaning of data and removes duplicate account because that affects the accuracy rate.

Extraction – Extraction of attributes that we require in order to perform the analysis.

Collector – Analysis of the textual data like tweets using an algorithms like sentiment analysis.

Bot Detector – This stage finalizes an account to be a bot or a real user.

CHAPTER-7

IMPLEMENTATION AND PSEUDO CODE

After data analysis we 4 implemented algorithms

1. Decision Trees
2. Random Forest
3. Multinomial Naive Bayes
4. Custom Classification Algorithm

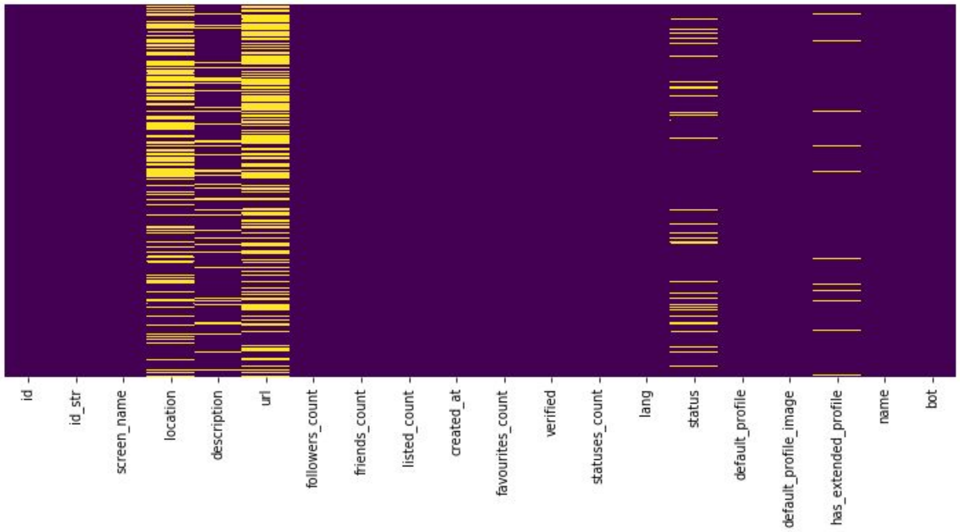
7.1 Kaggle Dataset Implementation

Step 1: Download the data and do some exploratory analysis

```
In [2]: filepath = 'C:/Users/Raghu/Downloads/BotDetection/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]
```

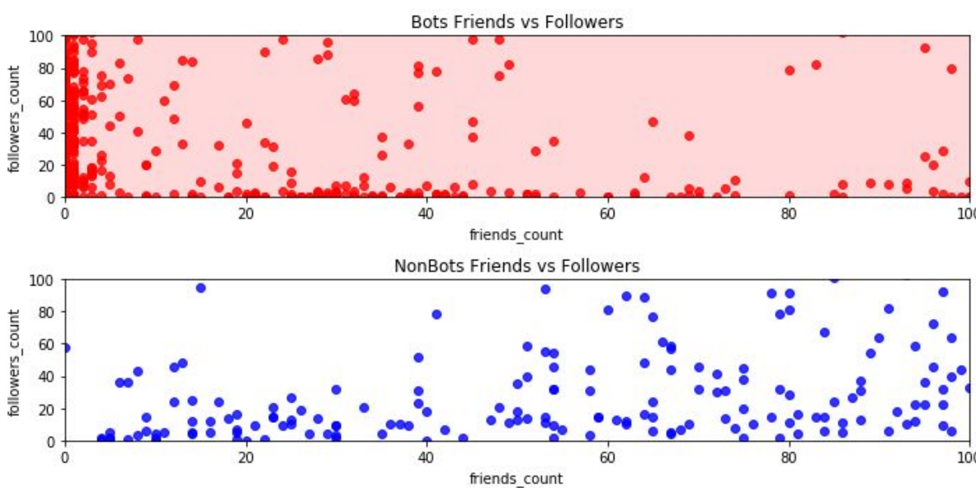
Step 2: The following heat map shows the missing values i.e all the missing values in yellow and all the non missing values in purple.

The location description and url have maximum missing values while the status, default profile image have less missing values.



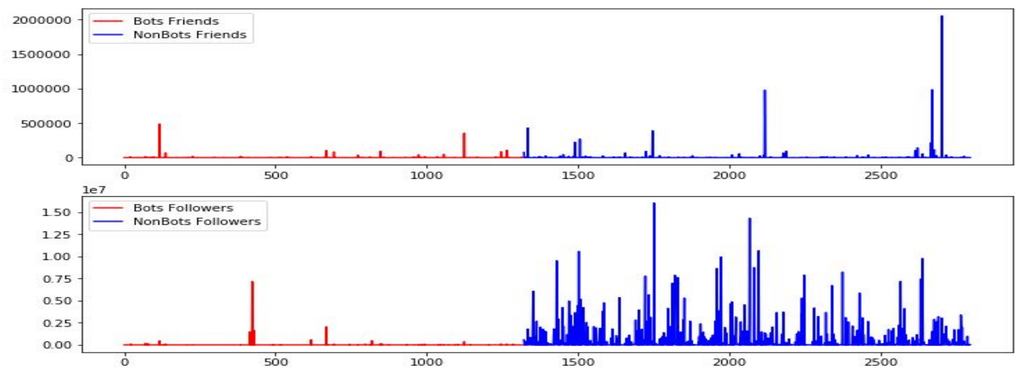
Step 3: The differentiation between bots and non bots indicate that the bots have more followers and they have less friends.

While the non bots have lot of friends and have followers also.



Step 4: Identifying imbalance in the data

The plot indicates that there is imbalance in the data



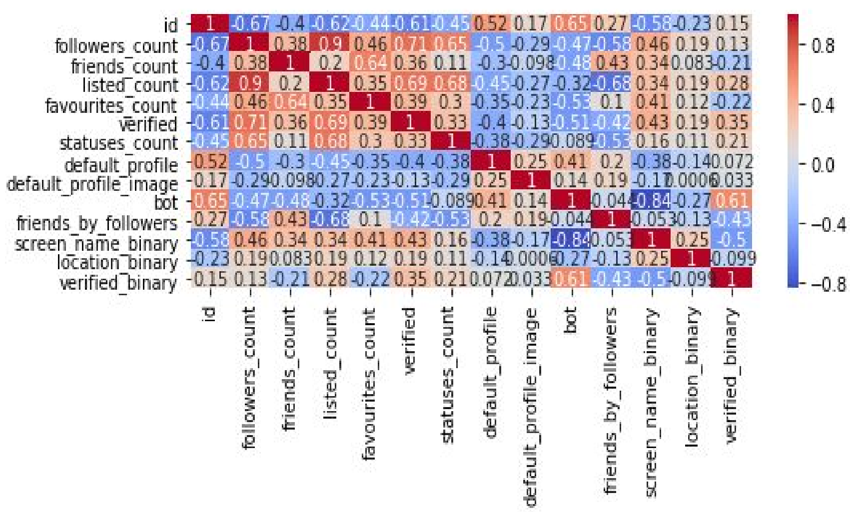
Step 5: Identify feature independence using spearman correlation.

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

	id	followers_count	friends_count	listed_count	favourites_count	verified	statuses_count	default_profile	default_profile_image
id	1.000000	-0.672925	-0.402346	-0.615005	-0.439430	-0.611899	-0.451945	0.522990	0.166601
followers_count	-0.672925	1.000000	0.375522	0.896126	0.457363	0.709732	0.649117	-0.496899	-0.293838
friends_count	-0.402346	0.375522	1.000000	0.204403	0.641529	0.356452	0.111118	-0.296358	-0.097607
listed_count	-0.615005	0.896126	0.204403	1.000000	0.349059	0.694340	0.684976	-0.447376	-0.269035
favourites_count	-0.439430	0.457363	0.641529	0.349059	1.000000	0.394227	0.295108	-0.348043	-0.226956
verified	-0.611899	0.709732	0.356452	0.694340	0.394227	1.000000	0.333278	-0.404650	-0.132298
statuses_count	-0.451945	0.649117	0.111118	0.684976	0.295108	0.333278	1.000000	-0.375918	-0.289999
default_profile	0.522990	-0.496899	-0.296358	-0.447376	-0.348043	-0.404650	-0.375918	1.000000	0.246979
default_profile_image	0.166601	-0.293838	-0.097607	-0.269035	-0.226956	-0.132298	-0.289999	0.246979	1.000000
bot	0.652131	-0.468430	-0.483105	-0.318445	-0.526228	-0.508555	-0.089018	0.407748	0.139669
friends_by_followers	0.270435	-0.577157	0.427638	-0.681034	0.104797	-0.419815	-0.533971	0.197929	0.190986
screen_name_binary	-0.576100	0.458213	0.342145	0.338698	0.408864	0.434177	0.162213	-0.377572	-0.166388
location_binary	-0.228328	0.189675	0.082692	0.188797	0.120941	0.191922	0.105333	-0.138378	0.000596
verified_binary	0.150100	0.130717	-0.210592	0.281360	-0.220894	0.346505	0.207384	0.072351	0.033021

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



Step 6: Perform feature engineering on screen name, description, status as we cannot perform correlation for categorical data. So these attributes are considered for feature engineering while attributes likeverified_listed_count for feature extraction.

To perform this feature engineering we created a bag_of_words model which identifies whether an account in twitter is a bot or not. So we have converted screen_name, description, status into binary using our own countvectorizer algorithm.

The listed count binary wherever it is above 20000 is considered as false and created a new feature.

Performing Feature Engineering

```
[16]: #filepath = 'https://raw.githubusercontent.com/jubins/ML-TwitterBotDetection/master/FinalCode/kaggle_data/'
filepath = 'C:/Users/Raghu/Downloads/BotDetection/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)
```

Performing Feature Extraction

```
[17]: 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_co
```

Step 7: Implementing the first model - Decision Tree

Its carried out based on the feature engineering and extraction which we have done and shows an accuracy of 88% on training data and on test data 87%.

```
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))
```

```
Trainig Accuracy: 0.88707
Test Accuracy: 0.87857
```

Step 8: The multinomial Naive Bayes performs poorly.

Step 9: Even random Forest gives less accurate results.

Hence we implemented our own custom classification algorithm.

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

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

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

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

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

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

```
C:\Users\Raghu\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: Deprecation
is an internal NumPy module and should not be imported. It will be removed in a future NumPy re
  from numpy.core.umath_tests import inner1d

Trainig Accuracy: 0.86612
Test Accuracy: 0.86190
```

Step 10 : Implementation of Custom Classification Algorithm

Here we created our own bag_of_models as earlier and created our own vectors and we also considered other features like the buzz feed in the users that identifies whether its a real user and finally considered the listed count and predicted the bots.

Later plotted Receiver Operating Characteristic (ROC) curve for our own custom classification model.

The accuracy for train and test data is respectively 96 and 93 percent which proves that the custom classification model is way better that all other models.

```
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
```

8.2 Tweepy Dataset Implementation

Step 1: Consider the attributes which determine a twitter account to be a bot or non bot and start classifying based on that.

```
def createOutput(data, isbot):
    header = ['id', 'id_str', 'screen_name', 'location', 'description', 'url',
              'followers_count', 'friends_count', 'listed_count', 'created_at',
              'favourites_count', 'verified', 'statuses_count', 'lang', 'status',
              'default_profile','default_profile_image', 'has_extended_profile',
              'name']
    d = {}
    for key in header:
        if key not in data.keys():
            d[key] = ""
        elif key == 'status':
            d[key] = str(data[key])
        else:
            d[key] = data[key]

    df = pd.DataFrame(d, columns= header, index=np.arange(1))
    df['bot'] = isbot
    return df

def get_bots_list():
    bots_list = []
    for bots in tweepy.Cursor(api.list_members, '01101010', 'bot-list').items():
        bots_list.append (bots._json['screen_name'])
    return bots_list[:50]

def real_users_list():
    real_users = []
    for users in tweepy.Cursor(api.list_members, 'Scobleizer', 'most-influential-in-tech').items():
        real_users.append (users._json['screen_name'])
    return real_users[:50]
```

Step 2: Go to twitter developer section and create an app based on machine learning requirement.

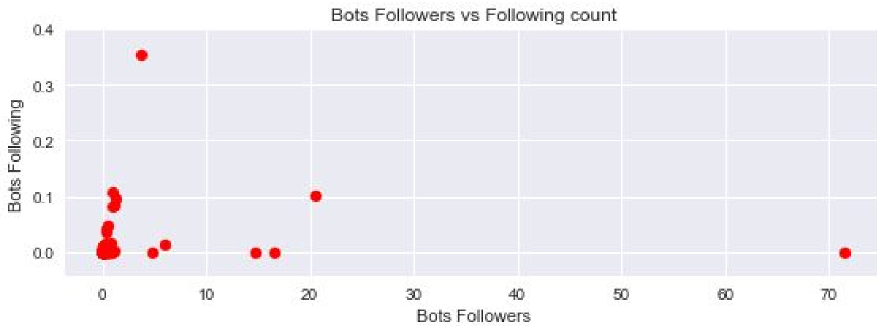
The app will generate us access key and token and consumer key and token which is unique for each individual

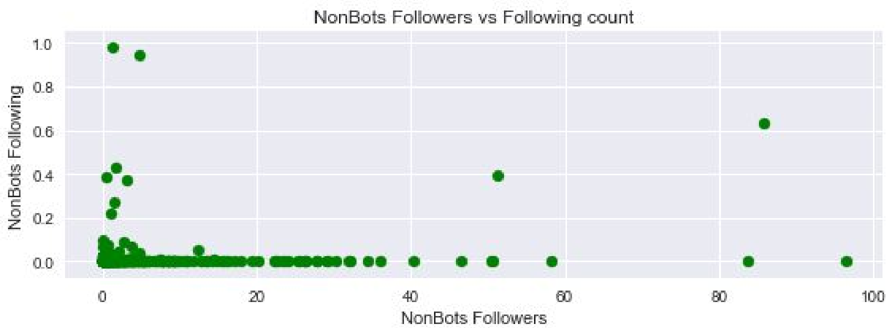
This is used to extract data from Twitter using tweepy.

```
start=time.time()
#Twitter credentials
consumer_key='##################################'
consumer_secret='###############################'
access_key = '##################################'
access_secret = '###############################'

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key,access_secret)
api = tweepy.API(auth)

user_list, filename = get_user_list()

df = pd.DataFrame()
for i,users in enumerate(user_list, start=1):
    isbot=0
    if(i<=50):
        isbot=1
    data = api.get_user(users)._json
    data_df1 = createOutput(data, isbot)
    df = pd.concat([data_df1, df], axis= 0, ignore_index = True)

df.to_csv(filename, encoding='utf-8')
print ("All records are saved to csv. \nDuration: "+str(time.time()-start)+" seconds.")
```

Step 3: After the classification plot is drawn as shown below indicating the friends:followers ratio.

This shows that the bots have more followers than friends and non bots have both friends and followers.





Step 4:

Create a bot identification criteria based on attributes and classify the dataset randomly into train and test dataset.

```
#Creating Bots identifying condition
#bots[bots.listedcount>10000]
condition = (bots.screen_name.str.contains("bot", case=False)==True)|(bots.description.str.contains("bot", case=False)==True)|(bot
bots = vectorize_bots(bots, condition)
print("Bots shape: {0}".format(bots.shape))

#Creating NonBots identifying condition
condition = (nonbots.screen_name.str.contains("bot", case=False)==False)| (nonbots.description.str.contains("bot", case=False)==Fa
nonbots = vectorize_nonbots(nonbots, condition)
print("Nonbots shape: {0}".format(nonbots.shape))
```

```
Bots shape: (1056, 24)
Nonbots shape: (1176, 24)
```

```
]: #Joining Bots and NonBots dataframes
df = pd.concat([bots, nonbots])
print("DataFrames created.")
```

```
DataFrames created.
```

```
]: #Splitting data randomly into train_df and test_df
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df, test_size=0.2)
print("Randomly splitting the dataset into training and test, and training classifiers.\n")
```

```
Randomly splitting the dataset into training and test, and training classifiers.
```

Final Step: Test using the decision tree classifier .

It gave an accuracy of 91%.

Using Decision Tree Classifier

```
#Using Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

clf = DecisionTreeClassifier(criterion='entropy')

#80%
X_train = train_df[['screen_name_binary', 'description_binary', 'location_binary', 'verified_binary']] #train_data
y_train = train_df['bot'] #train_target

#20%
X_test = test_df[['screen_name_binary', 'description_binary', 'location_binary', 'verified_binary']] #test_Data
y_test = test_df['bot'] #test_target

#Training on decision tree classifier
model = clf.fit(X_train, y_train)

#Predicting on test data
predicted = model.predict(X_test)

#Checking accuracy
print("Decision Tree Classifier Accuracy: {0}".format(accuracy_score(y_test, predicted)))
```

```
Decision Tree Classifier Accuracy: 0.9172259507829977
```

CHAPTER-8

TEST PLAN

8.1 Test Strategies

The Test strategy may be for the following types of tests: -

· Stress Testing: It is a subset of Performance testing. We check the amount of load the system can bear. Greater the file size, more the time it takes to process. If the file size is very huge, it could crash the system. Stress testing was performed by using exceptional scenarios where the server was exposed to unusual/incorrect data and we checked how it was handled.

· Unit Testing: In unit testing, the interface, local modules,independent paths and algorithms are checked. Most of the system is comprised of modules with dedicated algorithms. They are checked for efficiency. For unit testing, we use individual frames generated. A video generates 'n' frames out of which we pick one. The chosen frame is used for text localization. The localized frame is inpainted. The output frame is obtained.

· User Friendliness: It is a measure of how easy the usage of interface is. All the options should be clearly visible and usable. The webpage is very simple. It consists of two options - Upload and download. The buttons are easily locatable to the user. Navigation across the web pages is made user friendly.

· Error recovery: This includes an important case of missing frame or interruption in the execution of the code. Error recovery is done by restarting the session and the code.

· Error Handling: If the code gets interrupted while execution, the buffer output has to be cleared and it has to be restarted.

· Batch Testing: Our mobile application is very user driven, there is not much scope for creating scripts that automate the testing process by running a batch of commands sequentially. Uploading and downloading of the video and subtitle file is to be done manually.

· Function Testing: Every module is individually tested to ensure it works as required. Any errors will be taken care of.

· System and Integration Testing: When the above units/modules were combined, some defects were discovered. A part of the testing team used the method of black-box testing to test the completeness of the functionalities and the white-box testing team tested the validity of each code segment and the contribution of each line to the final output. There were defects like improper localization, audio not being synced properly with the video. The system testing phase was a verification of the existing documentation to confirm that the implemented functionality as a whole integrated unit/system does not contradict what was mentioned in the documentation.

8.2 Performance Criteria

Static Requirements:

· Image data transfer through internet connection makes performance measures crucial

· The number of simultaneous users to be supported : n users at n systems

· Number of files and records to be handled: 1 video at a time.

Dynamic numerical requirements may include:-

· The number of locations to which the system caters : n systems

· Tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions

· Image processing should be optimized so it should not take time more than 2 minutes. Every frame is assessed for localization whether it has subtitle text or not.

8.3 Test Environment

· Each system testing the application should have an Internet connection with upload speed of at least 2MBps and download speed of at least 4 MBps.

· Localhost should be up and running.

· The latest versions of Python, OpenCV and Tesseract-OCR should be installed on the system where the application is tested.

· The system testing the application should have a RAM of at least 8GB. Lower RAM will take more time to process the video. It could even crash.

Risk#	Risk	Nature of Impact	Contingency Plan
1.	Abnormal termination of code during execution.	Single point of failure leads to restarting modules that are dependent on it.	Integrate the code such that dependencies are fewer.
2.	Lower RAM – <8GB	slow execution	Get higher RAM (Preferably >8GB)
3	Loss of internet connection / Low speed (<2MBps)	Abnormal termination	Check internet connection. If greater spells of disconnection, get stable internet connection with higher speed and restart the code. Preferable Internet speed: 4MBps upload 2MBps – download

8.4 Test Data

Dataset is generated by carefully choosing videos with hardcoded subtitles. Dataset for the project is not available online. Videos were downloaded from different places.

A video is chosen in random and processed.

There also exists a Video with hardcoded subtitles and with no subtitles. It will aid in the validation process.

CHAPTER-9

TESTING

The testing of this project was an ongoing process. Each module was tested as and when it was completed. If the desired output was not received, necessary changes were made. A final testing was done to check if the modules work properly when run in parallel.

The first phase of testing included checking if a video could be converted into frames and if it could be generated back from these frames without any loss of information. The second phase was to localize and detect the text regions of individual frames. Some image enhancements and manipulations were done to improve the accuracy of Tesseract OCR. The third phase included the testing of the inpainting algorithm. The fourth phase included the extraction of audio from the original video and addition of this audio to the generated inpainted video. The last phase of testing was done to ensure the smooth and efficient running of the individual modules in parallel.

The test cases used for the testing of the project are:

- Videos containing subtitle text of different font sizes – The modules worked perfectly for different font sizes.

- A video which had hardcoded subtitles and the same video without hard coded subtitles – On comparing the results obtained by our inpainting technique to the original video, we got an RMS value of 0.13, that is, there was an 87% accuracy in our inpainting technique.

- Videos with subtitles with a specific background around the subtitle text – The modules worked perfectly for a video with a background for the subtitle text.

· Different intensity of white for the subtitles – A test case video with subtitle text intensity closer to the lower bound and a bit lower than our subtitle text intensity range was used. This resulted in non-precise inpainting results.

Each of the mentioned test cases were run through all the modules separately and in parallel. The results obtained did not vary.

How good our prediction is ?

Our Prediction on Test Data		
id	screen_name	bot
119509520	chrisbrown	0
856303860	94_kichi_bot	1

Actual twitter Account says

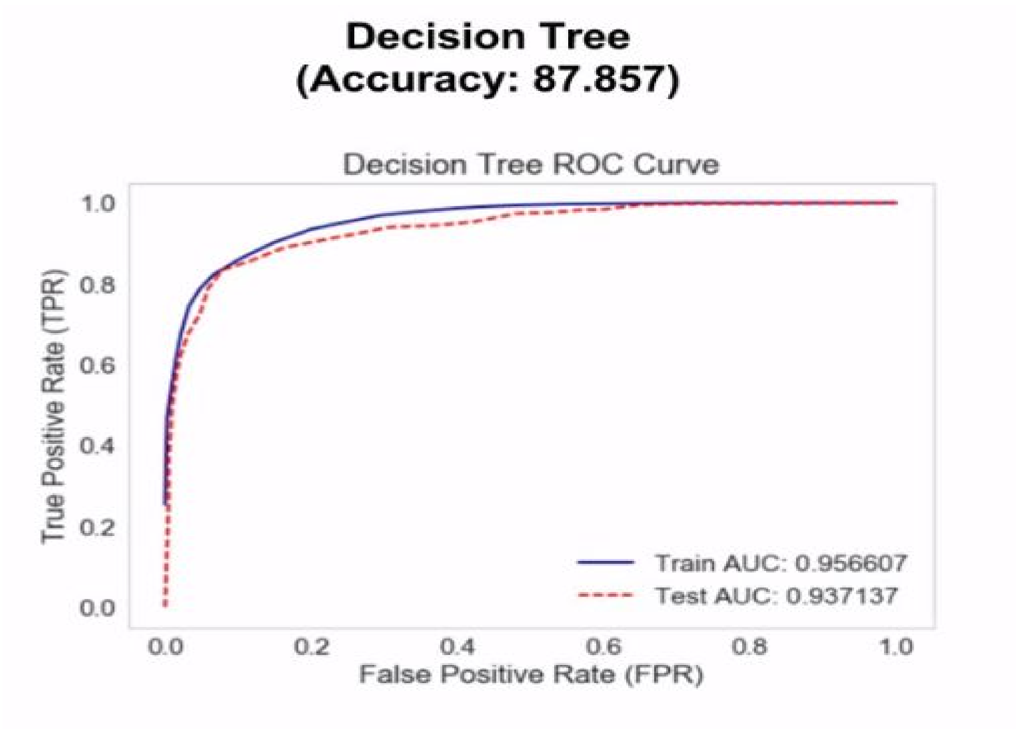
CHAPTER-10

RESULTS AND DISCUSSION

The result of the 4 models is as given below for the training and test dataset**.**

Accuracy Score	**Training Test**	**Test Set**
Decision Tree	0.8824	0.8785
Multinomial Naive Bayes	0.5421	0.5631
Random Forest	0.8252	0.7916
Custom Classification Algorithm	0.9646	0.9385

ROC curve for the Decision tree model

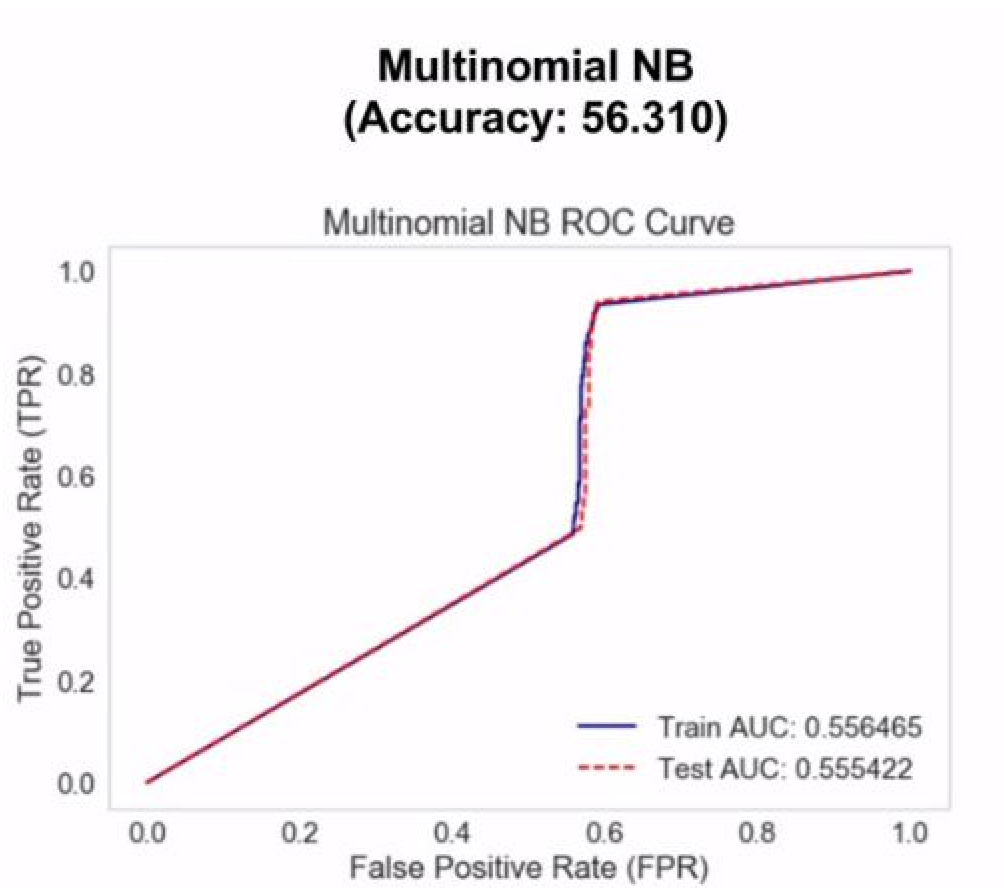


Has an accuracy of 95% on training data

Has an accuracy of 93% on test data

The blue line indicates the training data curve and red test data curve

ROC curve for the Multinomial Naive Bayes model



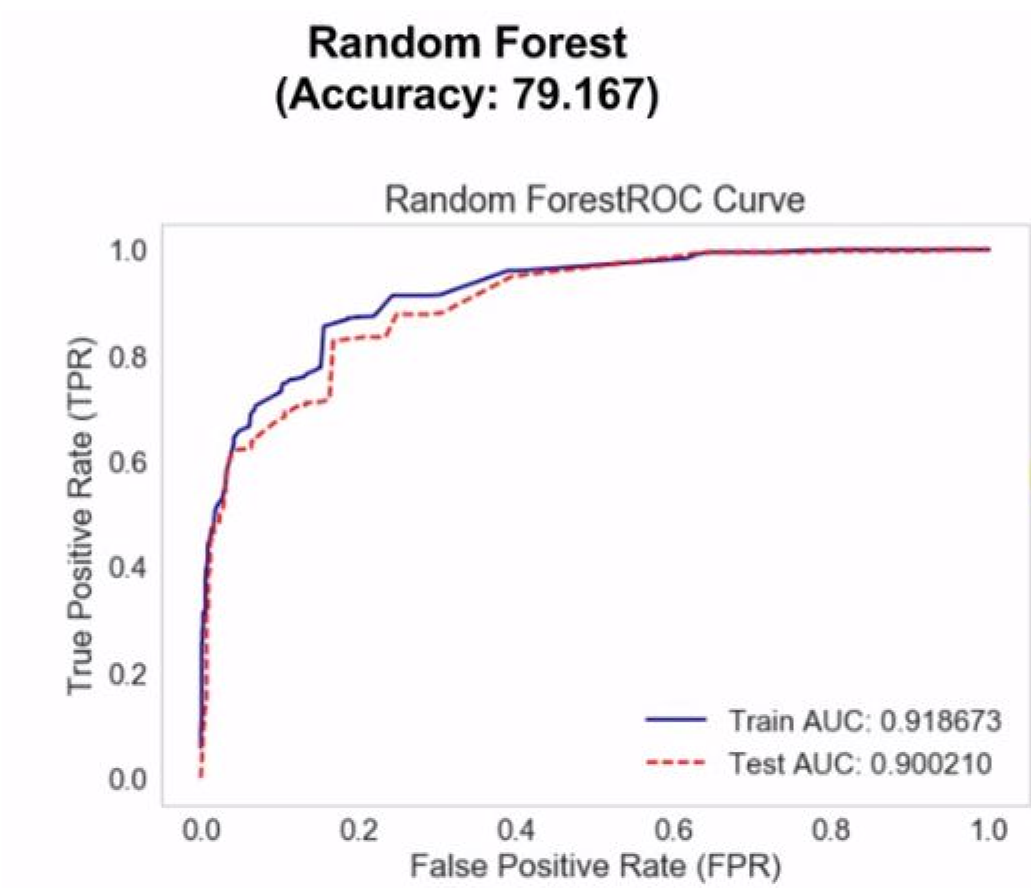
This has low accuracy

Has an accuracy of 55% on training data

Has an accuracy of 55% on test data

The blue line indicates the training data curve and red test data curve

ROC curve for the Random Forest model



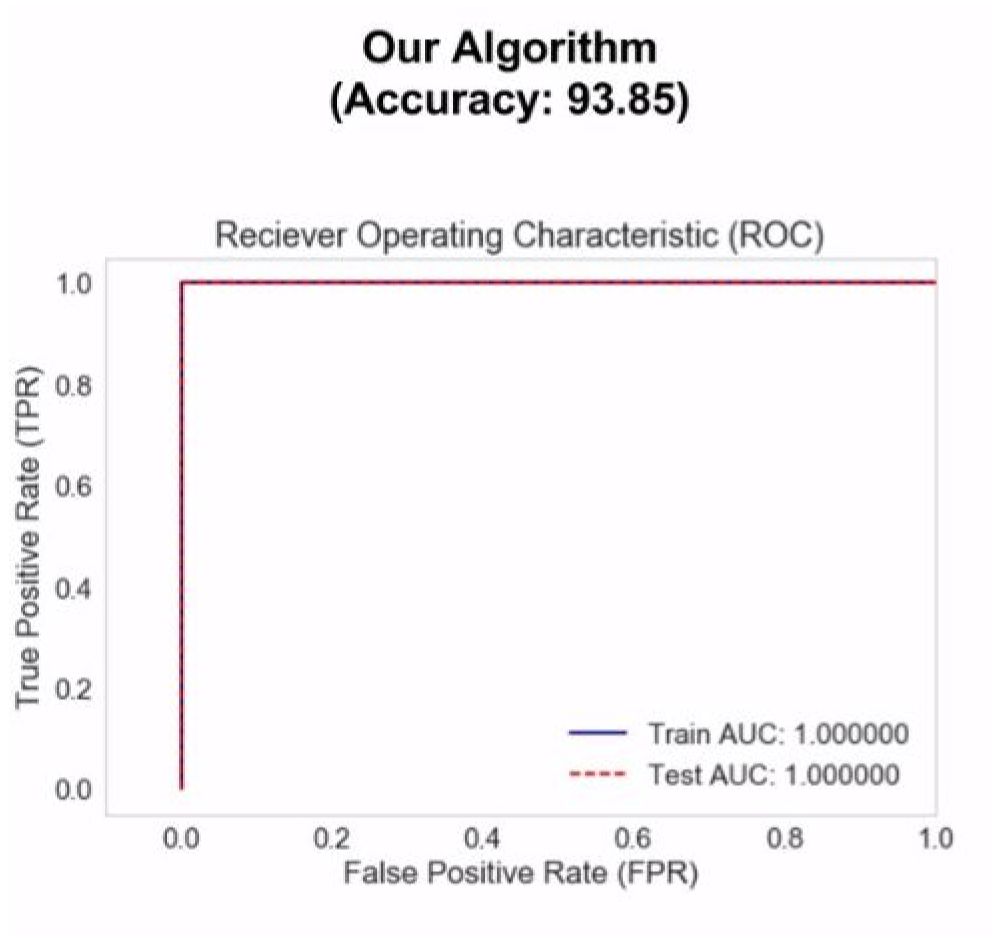
Comparatively has better accuracy than Multinomial Naive Bayes

Has an accuracy of 91% on training data

Has an accuracy of 90% on test data

The blue line indicates the training data curve and red test data curve

ROC curve for the Custom Classification Algorithm



Has best accuracy out of all the models

Has an accuracy of 96% on training data

Has an accuracy of 93% on test data

The blue line indicates the training data curve and red test data curve

CHAPTER-11

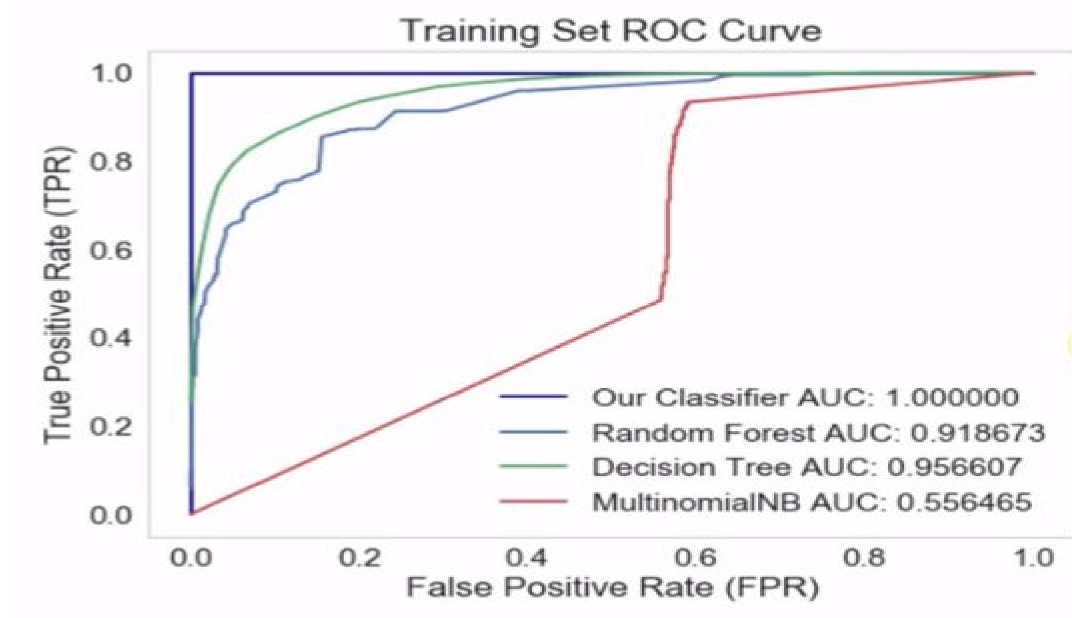
SNAPSHOTS

The final output of all the models

Below image shows the ROC curve for all 4 models

Blue curve being our custom classification algorithm showing highest accuracy

This is for the training dataset

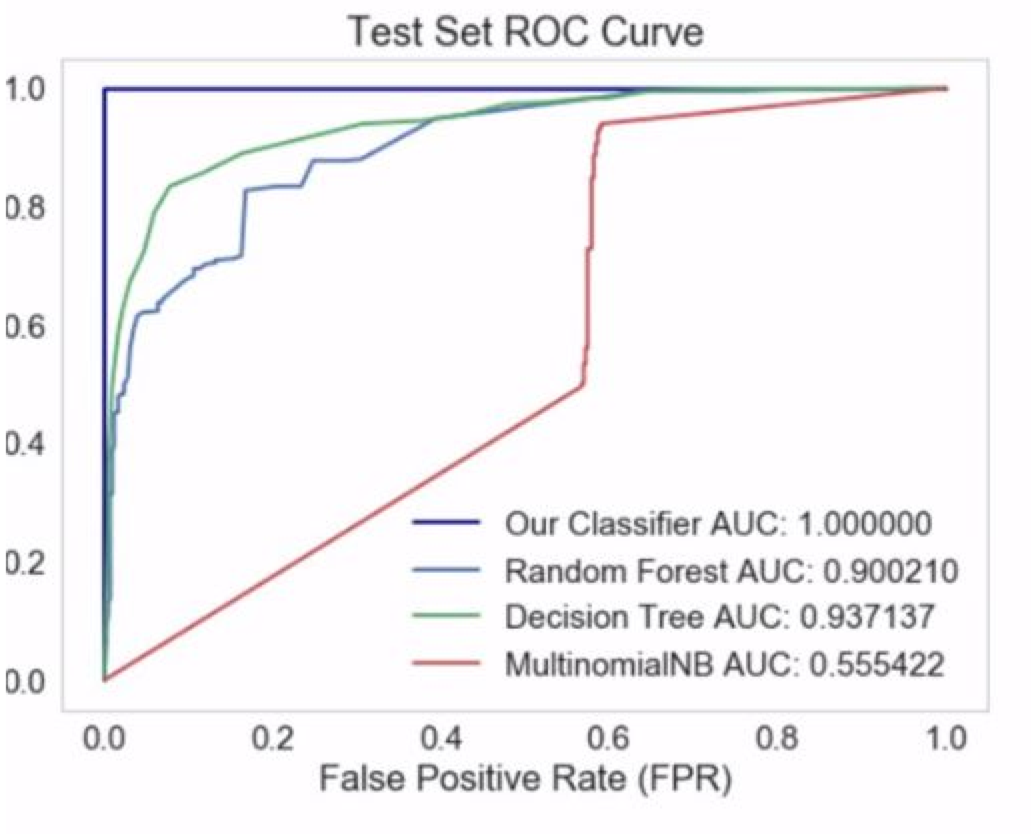


The final output of all the models

Below image shows the ROC curve for all 4 models

Blue curve being our custom classification algorithm showing highest accuracy

This is for the tes dataset



CHAPTER-12

CONCLUSIONS

Given the prevalence of sophisticated bots on social media platforms such as Twitter, the need for improved, inexpensive bot detection methods is apparent. We proposed a novel contextual LSTM architecture allowing us to use both tweet content and metadata to detect bots at the tweet level. From a single tweet, our model can achieve an extremely high accuracy exceeding 96% AUC. We show that the additional metadata information, though a weak predictor of the nature of a Twitter account per se, when exploited by LSTM decreases the error rate by nearly 20%. In addition to this, we propose methods based on synthetic minority oversampling that yield a near perfect user-level detection accuracy (> 99% AUC). Both these methods use a very minimal number of features that can be obtained in a straightforward way from the tweet itself and its metadata, while surpassing prior state of the art.

CHAPTER-13

FURTHER ENHANCEMENTS

In the future, we plan to make our system open source, and to implement a Web service (for example, an API) to allow the research community to perform tweet-level bot detection using it. From a research standpoint, we plan to use the proposed framework to scrutinize social media conversation in different contexts, in order to determine the extent of the interference of bots with public discourse, as well as to understand how their capabilities and sophistication evolve over time.

REFERENCES/BIBLIOGRAPHY

- Eiman Alothali, Nazar Zaki, Elfadil A. Mohamed, Hany Alashwal, "Detecting Social Bots on Twitter: A Literature Review", *Innovations in Information Technology (IIT) 2018 International Conference on*, pp. 175-180, 2018.
- Ryutaro Ushigome, Mio Suzuki, Tao Ban, Takeshi Takahashi, Daisuke Inoue, Takeshi Matsuda, Michio Sonoda, "Establishing Trusted and Timely Information Source using Social Media Services", *Consumer Communications & Networking Conference (CCNC) 2019 16th IEEE Annual*, pp. 1-2, 2019.
- https://ieeexplore.ieee.org/document/8093483
- https://www.kaggle.com/housemusic/twitter-bot-detection-data-collection
- https://www.kaggle.com/charvijain27/detecting-twitter-bot-data
- https://www.kaggle.com/bitandatom/social-network-fake-account-dataset/home
- Norah Abokhodair, Daisy Yoo, and David W McDonald. 2015. Dissecting a social botnet: Growth, content and influence in Twitter. In Proc. of the 18th ACM Conf. on Computer Supported Cooperative Work & Social Computing. ACM, 839–851.
- Terrence Adams. 2017. AI-Powered Social Bots. arXiv preprint arXiv:1706.05143 (2017).
- Lorenzo Alvisi, Allen Clement, Alessandro Epasto, Silvio Lattanzi, and Alessandro Panconesi. 2013. Sok: The evolution of sybil defense via social networks. In Proc. IEEE Symposium on Security and Privacy (SP). 382–396.
- Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. 2013. Joint Language and Translation Modeling with Recurrent Neural Networks.. In EMNLP, Vol. 3. 0.
- Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. 2004. A study of the behavior of several methods for balancing machine learning training data. ACM Sigkdd Explorations Newsletter 6, 1 (2004), 20–29.
- Alessandro Bessi and Emilio Ferrara. 2016. Social bots distort the 2016 US Presidential election online discussion. First Monday 21, 11 (2016).
- Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In Prov. 22nd Intl. ACM Conf. World Wide Web (WWW). 119–130.

- Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. 2012. Aiding the detection of fake accounts in large scale social online services. In 9th USENIX Symp on Netw Sys Design & Implement. 197–210.
- Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research 16 (2002), 321–357.
- Zi Chu, Steven Gianvecchio, Haining Wang, and Sushil Jajodia. 2010. Who is tweeting on Twitter: human, bot, or cyborg?. In Proc. 26th annual computer security applications conf. 21–30.
- Zi Chu, Steven Gianvecchio, Haining Wang, and Sushil Jajodia. 2012. Detecting automation of twitter accounts: Are you a human, bot, or cyborg? IEEE Tran Dependable & Secure Comput 9, 6 (2012), 811–824.
- Eric Clark, Chris Jones, Jake Williams, Allison Kurti, Michell Nortotsky, Christopher Danforth, and Peter Dodds. 2015. Vaporous Marketing: Uncovering Pervasive Electronic Cigarette Advertisements on Twitter. arXiv preprint arXiv:1508.01843 (2015).
- Eric Clark, Jake Williams, Chris Jones, Richard Galbraith, Christopher Danforth, and Peter Dodds. 2016. Sifting robotic from organic text: a natural language approach for detecting automation on Twitter. Journal of Computational Science 16 (2016), 1–7.
- Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. 2017. The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race. In Proceedings of the 26th International Conference on World Wide Web Companion. International World Wide Web Conferences Steering Committee, 963–972.
- Clayton Allen Davis, Onur Varol, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. 2016. Botornot: A system to evaluate social bots. In Proceedings of the 25th International Conference Companion on World Wide Web. International World Wide Web Conferences Steering Committee, 273–274.
- Juan Echeverría and Shi Zhou. 2017. The 'Star Wars' botnet with >350k Twitter bots. arXiv preprint arXiv:1701.02405 (2017).

- Aviad Elyashar, Michael Fire, Dima Kagan, and Yuval Elovici. 2013. Homing socialbots: intrusion on a specific organization's employee using Socialbots. In Proc. IEEE/ACM Intl. Conf. on Advances in Social Networks Analysis and Mining. 1358–1365.
- Emilio Ferrara. 2015. Manipulation and abuse on social media. ACM SIGWEB Newsletter Spring (2015), 4.
- Emilio Ferrara. 2017. Disinformation and social bot operations in the run up to the 2017 French presidential election. First Monday 22, 8 (2017).
- Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. 2016. The rise of social bots. Commun. ACM 59, 7 (2016), 96–104.
- Michelle C Forelle, Philip N Howard, Andrés Monroy-Hernández, and Saiph Savage. 2015. Political bots and the manipulation of public opinion in Venezuela. (2015).
- Daniel Gayo-Avello. 2017. Social Media Won't Free Us. IEEE Internet Computing 21, 4 (2017), 98–101.
- Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).
- Yoav Goldberg. 2016. A Primer on Neural Network Models for Natural Language Processing. J. Artif. Intell. Res.(JAIR) 57 (2016), 345–420.
- Guofei Gu, Junjie Zhang, and Wenke Lee. 2008. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic.. In NDSS, Vol. 8. 1–18.
- David Gunning. 2017. DARPA: Explainable Artificial Intelligence (XAI). (2017). https://www.darpa.mil/program/explainable-artificial-intelligence
- Stefanie Haustein, Timothy D Bowman, Kim Holmberg, Andrew Tsou, Cassidy R Sugimoto, and Vincent Larivière. 2016. Tweets as impact indicators: Examining the implications of automated "bot" accounts on Twitter. Journal of the Association for Information Science and Technology 67, 1 (2016), 232–238.
- Haibo He and Edwardo A Garcia. 2009. Learning from imbalanced data. IEEE Transactions on knowledge and data engineering 21, 9 (2009), 1263–1284.

- Cong Duy Vu Hoang, Trevor Cohn, and Gholamreza Haffari. 2016. Incorporating Side Information into Recurrent Neural Network Language Models.. In HLTNAACL. 1250–1255.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation 9, 8 (1997), 1735–1780.
- Philip N Howard and Bence Kollanyi. 2016. Bots,# strongerin, and# brexit: Computational propaganda during the uk-eu referendum. Browser Download This Paper (2016).
- Vineet John. 2017. A Survey of Neural Network Techniques for Feature Extraction from Text. arXiv preprint arXiv:1704.08531 (2017).
- Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15). 2342–2350.
- Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2015. Visualizing and understanding recurrent networks. arXiv preprint arXiv:1506.02078 (2015).
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems. 1097–1105
- Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. Nature 521, 7553 (2015), 436–444.
- Kyumin Lee, Brian David Eoff, and James Caverlee. 2011. Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter.. In Proc. 5th AAAI Intl. Conf. on Web and Social Media.
- Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2015. Visualizing and understanding neural models in nlp. arXiv preprint arXiv:1506.01066 (2015).
- Brian D Loader and Dan Mercea. 2011. Networking democracy? Social media innovations and participatory politics. Information, Communication & Society 14, 6 (2011), 757–769.

- Panagiotis T Metaxas and Eni Mustafaraj. 2012. Social media and the elections. Science 338, 6106 (2012), 472–473.
- Tomas Mikolov and Geoffrey Zweig. 2012. Context dependent recurrent neural network language model. SLT 12 (2012), 234–239.
- Zachary Miller, Brian Dickinson, William Deitrick, Wei Hu, and Alex Hai Wang. 2014. Twitter spammer detection using data stream clustering. Information Sciences 260 (2014), 64–73.
- Silvia Mitter, Claudia Wagner, and Markus Strohmaier. 2013. A categorization scheme for socialbot attacks in online social networks. In Proc. of the 3rd ACM Web Science Conference.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013).
-] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. Nature 518, 7540 (2015), 529–533.
- Bjarke Mønsted, Piotr Sapieżyński, Emilio Ferrara, and Sune Lehmann. 2017. Evidence of Complex Contagion of Information in Social Media: An Experiment Using Twitter Bots. Plos One 12, 9 (2017).
- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In Empirical Methods in Natural Language Processing (EMNLP). 1532–1543. http://www.aclweb.org/anthology/ D14-1162
- Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. 2017. Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. arXiv preprint arXiv:1708.08296 (2017).
- Saiph Savage, Andres Monroy-Hernandez, and Tobias Höllerer. 2016. Botivist: Calling Volunteers to Action using Online Bots. In Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing. ACM, 813–822.

- Chengcheng Shao, Giovanni Luca Ciampaglia, Onur Varol, Alessandro Flammini, and Filippo Menczer. 2017. The spread of fake news by social bots. arXiv preprint arXiv:1707.07592 (2017).
- David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529, 7587 (2016), 484–489.
- Tao Stein, Erdong Chen, and Karan Mangla. 2011. Facebook immune system. In Proc. of the 4th Workshop on Social Network Systems. ACM, 8.
- VS Subrahmanian, Amos Azaria, Skylar Durst, Vadim Kagan, Aram Galstyan, Kristina Lerman, Linhong Zhu, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. 2016. The DARPA Twitter bot challenge. Computer 49, 6 (2016), 38–46.
- Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition. 1–9.
- Ivan Tomek. 1976. Two modifications of CNN. IEEE Trans. Systems, Man and Cybernetics 6 (1976), 769–772.
- Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y Zhao. 2013. You are how you click: Clickstream analysis for sybil detection. In Proc. USENIX Security. Citeseer, 1–15.
- Gang Wang, Manish Mohanlal, Christo Wilson, Xiao Wang, Miriam Metzger, Haitao Zheng, and Ben Y Zhao. 2013. Social turing tests: Crowdsourcing sybil detection. In Proc. of the 20th Network & Distributed System Security Symposium (NDSS).
- Dennis L Wilson. 1972. Asymptotic properties of nearest neighbor rules using edited data. IEEE Transactions on Systems, Man, and Cybernetics 2, 3 (1972), 408–421.
- Samuel C Woolley. 2016. Automating power: Social bot interference in global politics. First Monday 21, 4 (2016).

- Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y Zhao, and Yafei Dai. 2014. Uncovering social network sy

APPENDIX A DEFINITIONS, ACRONYMS AND ABBREVIATIONS

CCA – Custom Classification Algorithm

CCA - It is building our own classifier based on the feature extraction and engineering which meets our requirements.

Here features which we use as our custom features are name, description, status etc. apart from considering things like friends and followers