**A Project Report on**

# Detecting Malicious Twitter Bots using Machine Learning

A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

## Bachelor of Technology

## In

## Computer Science and Engineering


Submitted by

M. Jithin
(20H51A0599)

B. Sneha
(20H51A05B2)

M. Venusri
(20H51A05J0)

Under the esteemed guidance of

Dr. S. Kirubakaran
(Professor, Department of CSE)




**Department of Computer Science and Engineering**

**CMR COLLEGE OF ENGINEERING& TECHNOLOGY**
(UGC Autonomous)
*Approved by AICTE *Affiliated to JNTUH *NAAC Accredited with $A^{+}$ Grade

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.


**2020- 2024**

# CMR COLLEGE OF ENGINEERING & TECHNOLOGY

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING


## CERTIFICATE

This is to certify that the Mini Project II report entitled **" DETECTING MALICIOUS TWITTER BOTS USING MACHINE LEARNING"** being submitted by M.Jithin (20H51A0599), B.Sneha(20H51A05B2), M.Venusri (20H51A05J0) in partial fulfillment for the award of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out his/her under my guidance and supervision.

The results embodies in this project report have not been submitted to any other University or Institute for the award of any Degree.

**Dr. S. Kirubakaran**
**Professor**
**Dept. of CSE**

**Dr. Siva Skandha Sanagala**
**Associate Professor and HOD**
**Dept. of CSE**

# ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Dr. S. Kirubakaran, Professor** , Department of Computer Science and Engineering for his valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Dr. Siva Skandha Sanagala,** Head of the Department of Computer Science and Engineering, CMR College of Engineering and Technology, who is the major driving forces to complete my project work successfully.

We are very grateful to **Dr. Ghanta Devadasu**, Dean-Academics, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Major Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the **Teaching & Non- teaching** staff of Department of Computer Science and Engineering for their co-operation

We express our sincere thanks to **Shri. Ch. Gopal Reddy**, Secretary, CMR Group of Institutions, for his continuous care.

Finally, We extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

M. Jithin - 20H51A0599
B. Sneha - 20H51A05B2
M. Venusri - 20H51A05J0

# TABLE OF CONTENTS

## List of Figures

## List of Tables

# Abstract

Twitter play a significant role in our daily lives, offering a wide range of opportunities to their users. However, Twitter and online social networks (OSNs) in general are increasingly being utilized by automated accounts, commonly known as bots, as they continue to gain immense popularity across various user demographics. Malicious twitter Bots detection is required to detect real users from fraudulent users because it leads to spreading of spam messages and engage in fraudulent activities. To overcome this, we are going to differentiate bots from legitimate users using feature extraction techniques and find malicious bots and tweets using machine learning algorithm and deep learning architecture known as VGG19 which is combined with the convolutional neural network (CNN) in order to identify whether the tweets are posted by bots or real users and also identifying malicious twitter bots along with malicious URLs. By following these techniques, we can identify the account as bots or real user and prevent spreading of malicious content in the society. The deep learning architecture combined with convolutional neural network to evaluate over a series of experiments using two large real Twitter datasets and compare the experimental results with other machine learning algorithms and provide advantages over other existing techniques like logistic regression targeting the identification of malicious users in social media.

# CHAPTER 1
# INTRODUCTION

# CHAPTER 1

# INTRODUCTION

## 1.1 PROBLEM STATEMENT

To develop a machine learning-based system for the detection of malicious Twitter bots. The primary goal is to create a model that can accurately classify Twitter accounts as either genuine human users or malicious bots based on their behavior, content, and interactions on the platform, because

- Malicious bots often spread fake news, propaganda, and disinformation, which can have real-world consequences.
- Bots can be used to manipulate public opinion, interfere in elections, and undermine democratic processes.
- Some malicious bots engage in cyberbullying and harassment, which can harm individuals and discourage constructive online interactions.

The primary purpose of developing a malicious Twitter bot detection system is to maintain a healthy, secure, and authentic online environment while countering the negative impacts of automated malicious actors.

.

## 1.2 RESEARCH OBJECTIVE:

- Bot detection is required to detect fraudulent users and shield real users from false information and malevolent intent.
- Driving engagement and providing helpful information when a certain keyword or hashtag triggers a bot response.
- Spreading spam: Social media bots are often used for illicit advertising purposes by spamming the social web with links to commercial websites.
- Mitigating Economic Impact: Bots can impact businesses and markets by spreading rumors affecting stock prices or engaging in fraudulent activities. Detecting and preventing such activities can help mitigate economic losses.

## 1.3 PROJECT SCOPE:

The scope of the project is to develop a framework for identifying malicious Twitter bots using Logistic Regression and VGG19. The study aims to address the increasing number of users who hide their identities for malicious purposes and to provide a more effective method for detecting abusive bots on Twitter. In this we proposed VGG19 to Recognize Twitter Bots .VGG-19 is VGG stands for Visual Geometry Group; VGG-19 is a convolutional neural network that is 19 layers deep. You can load a pretrained version of the network trained on more than a million images.

ADVANTAGES:

- High security
- High accuracy
- High efficiency

# CHAPTER 2
# BACKGROUND WORK

# CHAPTER 2
# BACKGROUND WORK

## 2.1 USING MACHINE LEARNING TO DETECT FAKE IDENTITIES: BOTS VS HUMANS

### 2.1.1 INTRODUCTION:

There are a growing number of people who hold accounts on social media platforms (SMPs) but hide their identity for malicious purposes. Unfortunately, very little research has been done to date to detect fake identities created by humans, especially so on SMPs. In contrast, many examples exist of cases where fake accounts created by bots or computers have been detected successfully using machine learning models. In the case of bots these machine learning models were dependent on employing engineered features, such as the "friend-to-followers ratio."These features were engineered from attributes, such as "friend-count"and "follower-count,"which are directly available in the account profiles on SMPs. The research discussed in this paper applies these same engineered features to a set of fake human accounts in the hope of advancing the successful detection of fake identities created by humans on SMPs. Identity deception on big data platforms (like social media) is an increasing problem, due to the continued growth and exponential evolvement of these platforms. Social media is one of the preferred means of communication and has become a target for spammers and scammers alike . Cyberthreats like spamming, which involves the sending of unsolicited emails, are common in email applications. These same threats - and more - now emerge on social media platforms (SMPs), although in different manifestations. Much can be learned about people's behaviour and needs through analysing their interactions with one another. Habits and topics of conversations can be evaluated to deliver a better service or product to customers and ultimately to people at large. The same information can however also be used against people, very often in a deceptive way. For example, a cluster of people may influence an opinion when the other participants in the conversation are unaware that the "people" in the cluster are not real .Since the detection of fake social engagement is quite challenging, this vulnerability is greatly abused.

### 2.1.2 MERITS,DEMERITS AND CHALLENGES:

**MERITS:**

- **Classification Power:** Supervised machine learning algorithms excel in classification tasks, making them well-suited for identifying fake human identities on SMPs.
- **Feature Engineering:** Supervised machine learning allows for the engineering of features that describe the identity of an account, its relationships, and its behavior. This comprehensive feature set can improve the accuracy of detection.
- **Adaptability:** Supervised machine learning models can adapt to new data and patterns, allowing them to continuously improve their detection capabilities as new deceptive tactics emerge.

**DEMERITS:**

- **Labeling Challenges:** Supervised machine learning algorithms require labeled data for training. However, obtaining labeled datasets of deceptive human identities on SMPs can be challenging and may require manual effort.
- **Data Quality:** The quality of training data can significantly impact the performance of supervised machine learning algorithms. Noisy or biased data may lead to inaccurate or biased predictions.
- **Feature Engineering Complexity:** Engineering effective features to describe identity, relationships, and behavior on SMPs can be complex and time-consuming. Designing robust features that capture the nuances of human deception requires domain expertise.
- **Model Overfitting**: Without careful regularization and validation techniques, supervised machine learning models may overfit to the training data, resulting in poor generalization performance on unseen data.

**CHALLENGES:**

- **Real-Time Detection:** Detecting identity deception in real-time on SMPs presents a significant challenge. Supervised machine learning models may require computational resources and time for processing, which could hinder real-time detection capabilities.

- **Dynamic Nature of Deception:** Human deceptive tactics evolve over time, making it challenging to develop supervised machine learning models that can effectively detect new and emerging deceptive behaviors.
- **Privacy Concerns:** Analyzing user data to detect identity deception raises privacy concerns. Ensuring that detection algorithms comply with privacy regulations and respect user confidentiality is essential.
- **Adversarial Attacks:** Malicious actors may actively try to evade detection by crafting deceptive identities specifically designed to fool supervised machine learning algorithms. Adversarial training techniques may be necessary to mitigate this threat.

### 2.1.3 IMPLEMENTATION:

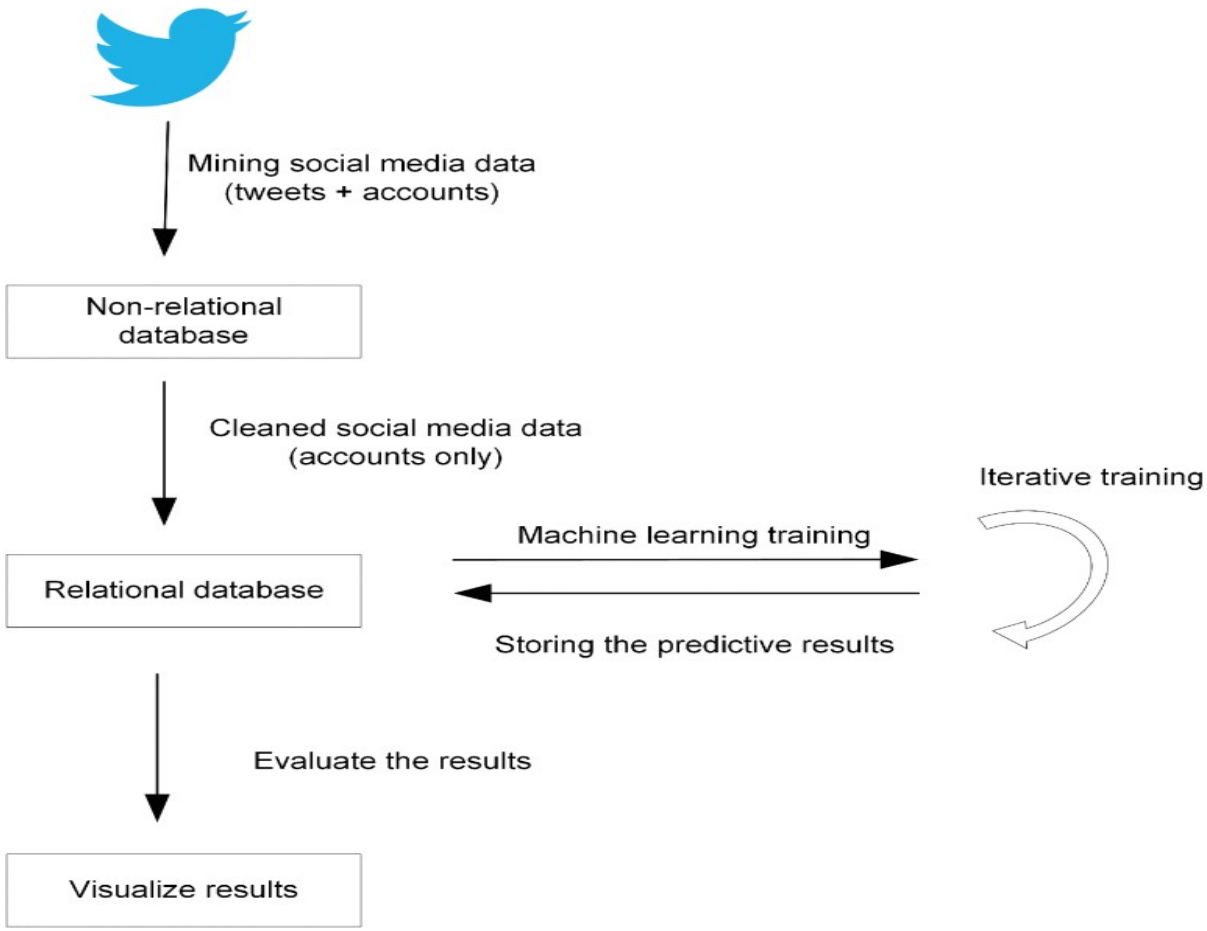


**Fig 2.1.3.1: Identify fake identities**

## 2.2 REAL-TIME DETECTION OF CONTENT POLLUTERS IN PARTIALLY OBSERVABLE TWITTER NETWORKS

### 2.2.1 INTRODUCTION:

Content polluters, or bots that hijack a conversation for political or advertising purposes are a known problem for event prediction, election forecasting and when distinguishing real news from fake news in social media data. Identifying this type of bot is particularly challenging, with state-of-the-art methods utilising large volumes of network data as features for machine learning models. Such datasets are generally not readily available in typical applications which stream social media data for real-time event prediction. In this work we develop a methodology to detect content polluters in social media datasets that are streamed in real-time. Applying our method to the problem of civil unrest event prediction in Australia, we identify content polluters from individual tweets, without collecting social network or historical data from individual accounts. We identify some peculiar characteristics of these bots in our dataset and propose metrics for identification of such accounts. We then pose some research questions around this type of bot detection, including: how good Twitter is at detecting content polluters and how well state-of-the-art methods perform in detecting bots in our dataset. Bots and content polluters in online social media affect the sociopolitical state of the world, from meddling in elections to influencing US veterans. In late September 2017, Twitter admitted to Congress that it had found 200 Russian accounts that overlapped with Facebook accounts which were used to sway Americans and create divisions during the elections held in 2016. Of course, some bots are useful as well, for instance accounts that will tweet alerts to people about natural disasters. The problem arises when they try to influence people or spread misinformation. The importance of detecting bots in online social media has produced an active research area on this topic . State-of-the-art methods for bot detection use historical patterns of behaviour and a rich feature set including textual, temporal, and social network features, to distinguish automated bots from real human users. However, for real-time application using large streamed datasets, such methods can be prohibitive due to the sheer volume, velocity, and incompleteness of data samples. In this work we develop a new method to detect one particular type of social bot – content polluters – in streamed microblog datasets such as Twitter. Content polluters are bots that attempt to subvert a genuine discussion by hijacking it for political or advertising purposes. As we will show, these

bots are a major concern for applications such as real-time event prediction, such as social unrest, from social media datasets.

### 2.2.2 MERITS,DEMERITS AND CHALLENGES:

**MERITS:**

- **Temporal Pattern Recognition:** The algorithm may excel in recognizing temporal patterns of content polluters, allowing for the identification of periodic or recurring deceptive behaviors over time.
- **Network Visualization:** Utilizing network visualization techniques can provide intuitive insights into the structure and dynamics of Twitter networks, aiding in the identification of suspicious clusters or communities associated with content polluters.
- **Dynamic Adaptation:** Algorithms capable of analyzing temporal patterns and visualizing network structures can adapt to evolving tactics employed by content polluters, ensuring effective detection in dynamic social media environments.
- **Comprehensive Insights:** By combining temporal pattern recognition and network visualization, researchers can gain comprehensive insights into the behavior and interactions of content polluters, facilitating the development of more accurate detection methods.

**DEMERITS:**

- **Data Complexity:** Analyzing temporal patterns and visualizing network structures may introduce complexity, requiring sophisticated algorithms and techniques to handle large-scale Twitter data effectively.
- **Interpretation Challenges**: Understanding and interpreting temporal patterns and network visualizations may pose challenges, particularly in the presence of noise or ambiguities in the data.
- **Resource Intensiveness:** Processing and analyzing temporal data and network structures may require significant computational resources and time, potentially limiting the scalability of the detection system.

- **Algorithmic Complexity:** Implementing algorithms for temporal pattern recognition and network visualization may involve complex mathematical and computational techniques, increasing the difficulty of implementation and maintenance.

**CHALLENGES:**

**Real-Time Processing:** Analyzing temporal patterns and visualizing network structures in real-time presents a significant challenge, as it requires efficient algorithms capable of processing streaming data with low latency.

**Partial Observability:** Partially observable Twitter networks may contain missing or incomplete temporal data, hindering the effectiveness of algorithms reliant on complete temporal patterns.

**Privacy Concerns:** Analyzing temporal data and network structures for detection purposes raises privacy concerns, necessitating careful consideration of ethical and legal implications, particularly regarding user consent and data protection regulations.

**Adversarial Manipulation:** Malicious actors may actively attempt to manipulate temporal patterns or network structures to evade detection, posing a constant challenge to the effectiveness of detection algorithms.

### 2.2.3 IMPLEMENTATION:

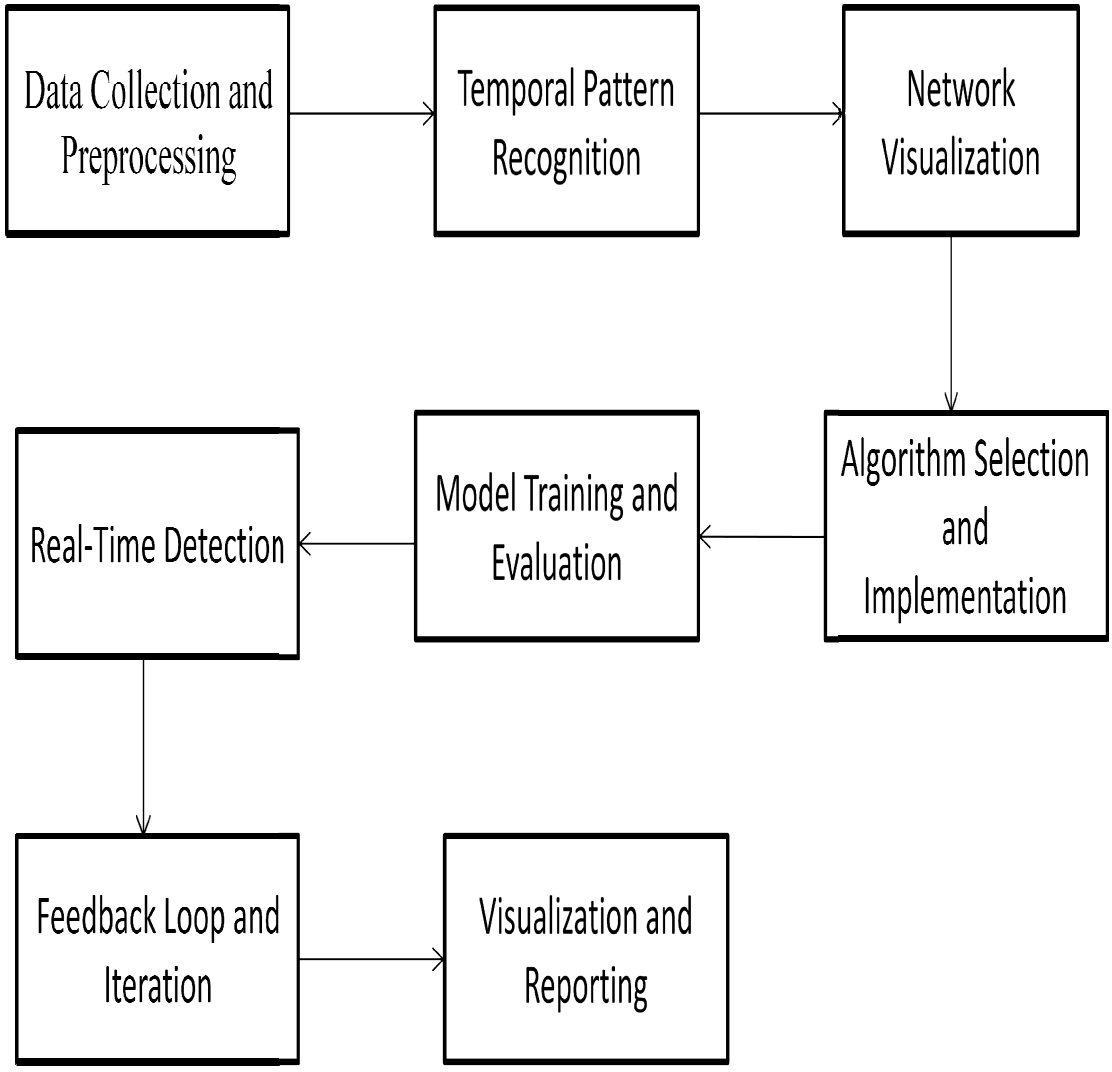


**Fig 2.2.3.1: Content Polluters in Twitter Networks**

## 2.3 DETECTING FAKE FOLLOWERS IN TWITTER: A MACHINE LEARNING APPROACH

### 2.3.1 INTRODUCTION:

Twitter popularity has fostered the emergence of a new spam marketplace. The services that this market provides include: the sale of fraudulent accounts, affiliate programs that facilitate distributing Twitter spam, as well as a cadre of spammers who execute large scale spam campaigns. In addition, twitter users have started to buy fake followers of their accounts. In this paper we present machine learning algorithms we have developed to detect fake followers in Twitter. Based on an account created for the purpose of our study, we manually verified 13000 purchased fake followers and 5386 genuine followers. Then, we identified a number of characteristics that distinguish fake and genuine followers. We used these characteristics as attributes to machine learning algorithms to classify users as fake or genuine. We have achieved high detection accuracy using some machine learning algorithms and low accuracy using others. Twitter has become a popular media hub where people can share news, jokes and talk about their moods and discuss news events. In Twitter users can send Tweets instantly to his/her followers. Also, Tweets can be retrieved using Twitter‟s real time search engine [1]. The ranking of tweets in this search engine depends on many factors, one of which is the user‟s number of followers. Twitter‟s popularity has made it an attractive place for spam and spammers of all types. Spammers have various goals: spreading advertising to generate sales, phishing or simply just compromising the system‟s reputation. Given that spammers are increasingly arriving on twitter, the success of real time search services and mining tools lies in the ability to distinguish valuable tweets from the spam storm [1]. There are various ways to fight spam and spammers such as URL blacklists, passive social networking spam traps, manual classification to generate datasets used to train a classifier that later will be used to detect spam and spammers [2]. So what is Twitter spam? As Twitter describes it in their website [3], Twitter spam is “a variety of prohibited behaviors that violate the Twitter Rules.” Those rules include among other things the type of behavior Twitter considers as spamming, such as:

- Posting harmful links (including links to phishing or malware sites).

- Aggressive following behavior (mass following and mass un-following for attention), particularly by automated means.
- Abusing the @reply or @mention function to post unwanted messages to users .
- Creating multiple accounts (either manually or using automated tools).
- Having a small number of followers compared to the number of people one is following.
- Posting repeatedly to trending topics to try to grab attention.
- Repeatedly posting duplicate updates.
- Posting links with unrelated tweets.

### 2.3.2 MERITS,DEMERITS AND CHALLENGES:

**MERITS:**

- **Robustness:** Support Vector Machine (SVM) is known for its ability to handle complex decision boundaries and high-dimensional data, making it robust in detecting patterns even in noisy datasets.
- **Interpretability**: Simple Logistic regression provides straightforward interpretability, allowing stakeholders to understand the influence of each feature on the classification of fake followers.
- **Adaptability:** Instance-based classifiers like 1-nearest neighbor are flexible and can adapt to changes in the dataset without retraining, making them suitable for dynamic environments.
- **Low Training Overhead**: Instance-based classifiers do not require an explicit training phase, reducing the computational overhead associated with model training and allowing for quick deployment.

**DEMERITS:**

- **Sensitivity to Noise:** Instance-based classifiers can be sensitive to noisy data and outliers, which may lead to misclassifications if the dataset contains significant noise.
- **Scalability:** Instance-based classifiers store the entire training dataset, which can be memory-intensive and may lead to scalability issues with large datasets.

- **Limited Generalization:** SVM and Logistic regression may struggle with generalizing patterns in the data if the feature space is not carefully chosen or if the dataset is highly imbalanced.
- **Complexity:** SVM models, especially with non-linear kernels, can be computationally expensive and may require tuning of hyper-parameters, which adds complexity to the modeling process.

**CHALLENGES:**

- **Feature Selection:** Identifying relevant features for detecting fake followers can be challenging, as it requires a deep understanding of the characteristics and behaviors associated with fake accounts.
- **Data Quality:** Ensuring the quality and reliability of training data is crucial for building accurate classifiers, as noisy or biased data can lead to poor performance.
- **Model Interpretability:** While Logistic regression provides interpretability, SVM models may lack transparency, making it difficult to understand the reasoning behind classification decisions.
- **Adversarial Attacks:** Malicious actors may actively try to circumvent detection algorithms by crafting sophisticated fake accounts designed to evade classification, posing a constant challenge to detection accuracy.

### 2.3.3 IMPLEMENTATION:

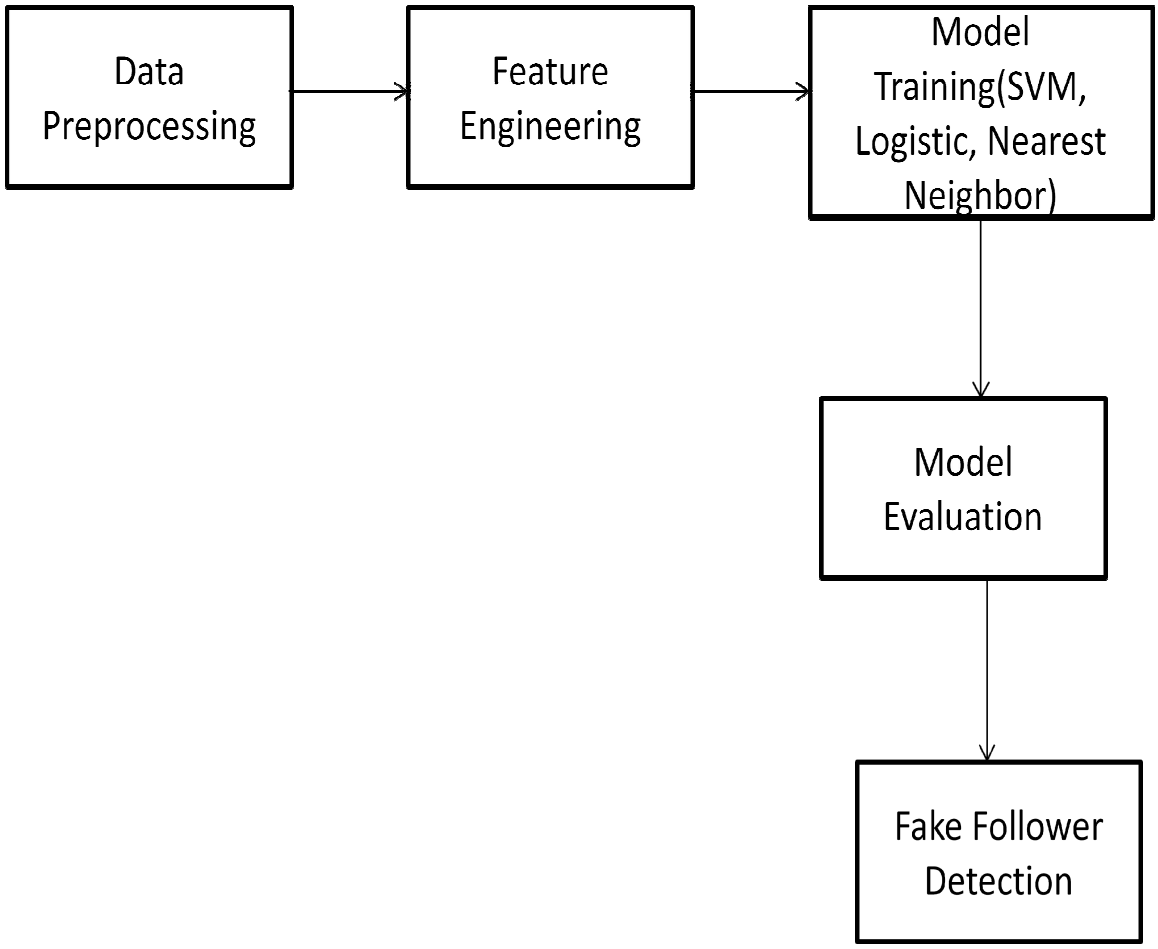


**Fig 2.3.3.1: Fake Followers in Twitter**

# CHAPTER 3
# PROPOSED SYSTEM

# CHAPTER 3

# PROPOSED SYSTEM

## 3.1 OBJECTIVE OF PROPOSED MODEL

The main objective is to develop a framework to detect malicious bots on social media especially in Twitter effectively. Bots detection is required to detect real users from fraudulent users because it leads to spreading of spam messages and engage in fraudulent activities. Therefore, early detection of bots is necessary in social media. This paper mainly focuses on Twitter bots detection. To overcome this, we are going to differentiate bots from legitimate users using feature extraction techniques and find malicious bots and tweets using machine learning algorithm. And then we are going to use a deep learning architecture to identify whether the tweets are posted by bots or real users. By following these techniques we can identify the account as bots or real user and also detect the malicious urls present in the tweet so that to prevent spreading of malicious content in the society.

## 3.2 ALGORITMS USED FOR PROPOSED MODEL

The algorithms used are logistic regression a machine learning algorithm and VGG19 a deep learning algorithm. These two are used to detect the bots and malicious urls.

**Logistic Regression:**

Logistic regression is a statistical method used for binary classification tasks, particularly in situations where the outcome variable is categorical and has only two possible outcomes. The fundamental principle behind logistic regression is to model the relationship between the independent variables and the probability of a particular outcome occurring. Unlike linear regression, which predicts continuous values, logistic regression employs the logistic function to transform the output into a probability score between 0 and 1. This probability score represents the likelihood of a sample belonging to one of the two classes.

In logistic regression, the model estimates the coefficients for each independent variable, which determines the influence of that variable on the probability of the outcome. These coefficients are optimized during the training process using techniques like maximum likelihood estimation.

Once the model is trained, it can predict the probability of the outcome for new observations based on their feature values.

One of the key advantages of logistic regression is its simplicity and interpretability. The coefficients of the model provide insights into the strength and direction of the relationships between the independent variables and the outcome. Additionally, logistic regression can handle both linear and nonlinear relationships between the independent variables and the log-odds of the outcome.

**VGG19:**

VGG19 is a convolutional neural network architecture known for its depth and simplicity. Developed by the Visual Geometry Group (VGG) at the University of Oxford, VGG19 is composed of 19 layers, including 16 convolutional layers and 3 fully connected layers. Its architecture follows a straightforward design principle of using small 3x3 convolutional filters with a stride of 1 and max-pooling layers with a 2x2 window and stride of 2.

One of the main advantages of VGG19 is its simplicity and uniformity, making it easy to understand and implement. The repeated pattern of convolutional layers followed by max-pooling layers contributes to its effectiveness in capturing hierarchical features from input images. Additionally, the use of small convolutional filters allows VGG19 to learn a rich set of features while keeping the number of parameters manageable.

VGG19 has demonstrated strong performance on various computer vision tasks, including image classification, object detection, and image segmentation. Its deep architecture enables it to learn complex patterns and representations, leading to state-of-the-art performance on benchmark datasets like ImageNet.

## 3.3 DESIGNING

### 3.3.1 UML DIAGRAM

**A. Class diagram:-**

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. A class with three sections, in the diagram, classes is represented with boxes which contain three parts:

The upper part holds the name of the class

The middle part contains the attributes of the class

The bottom part gives the methods or operations the class can take or undertake

**Class diagram:**

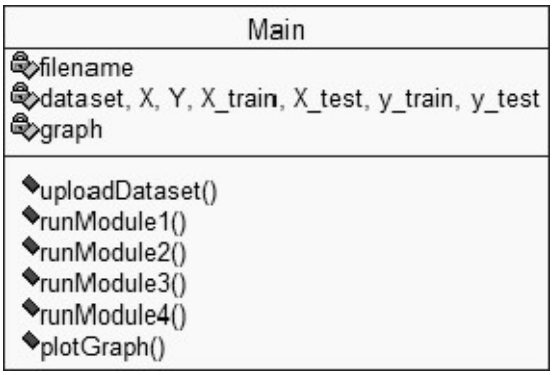


**Fig 3.3.1.1: Class Diagram**

**B. Use case diagram:-**

A **use case diagram** at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

**Use case diagram:**

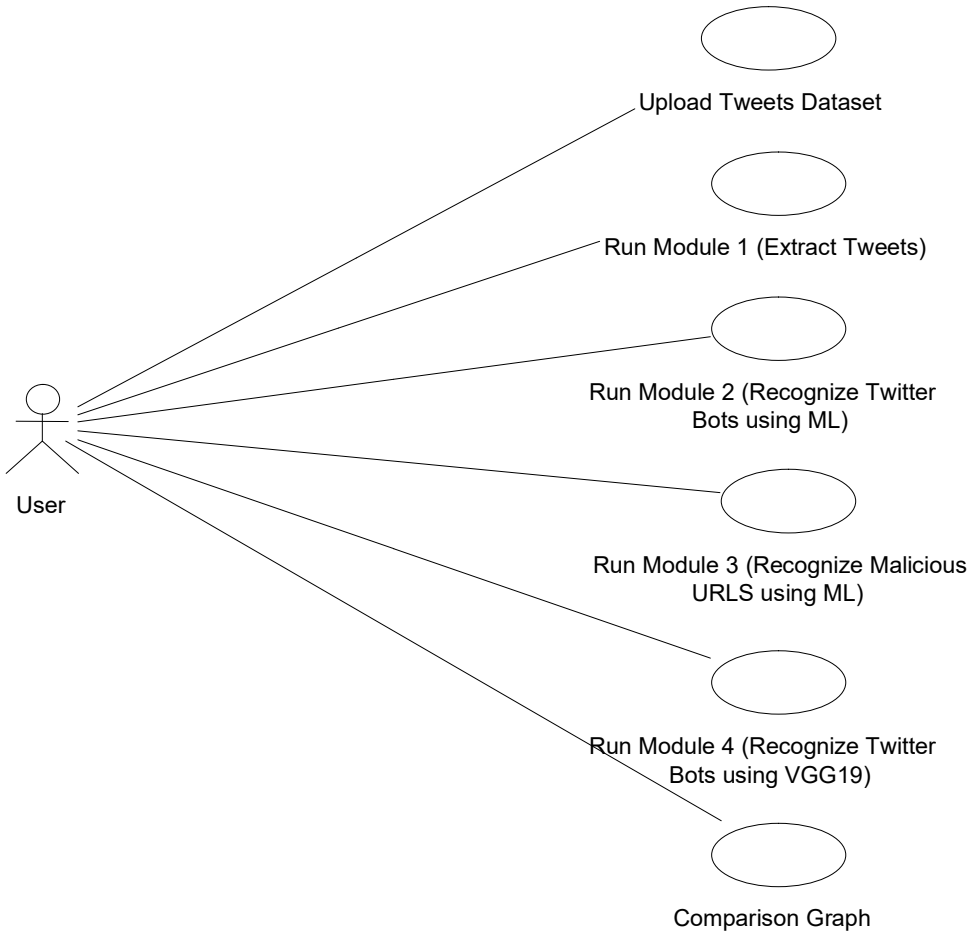


**Fig 3.3.1.2: Use case Diagram**

**C. Sequence Diagram:**

A **sequence diagram** is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams**, **event scenarios**, and timing diagrams.

**Sequence diagram:**

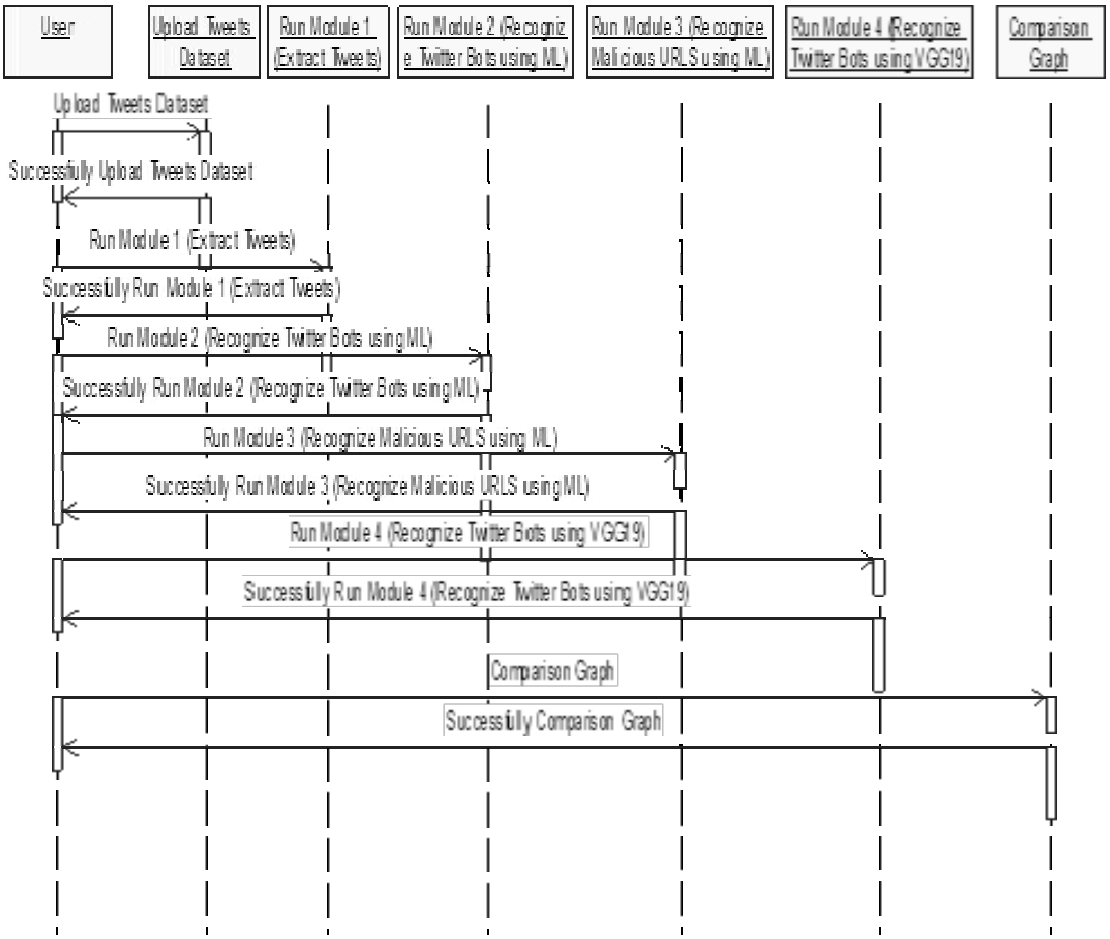


**Fig 3.3.1.3: Sequence Diagram**

**D. Collaboration diagram**

A collaboration diagram describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behavior of a system.

**Collaboration diagram:**

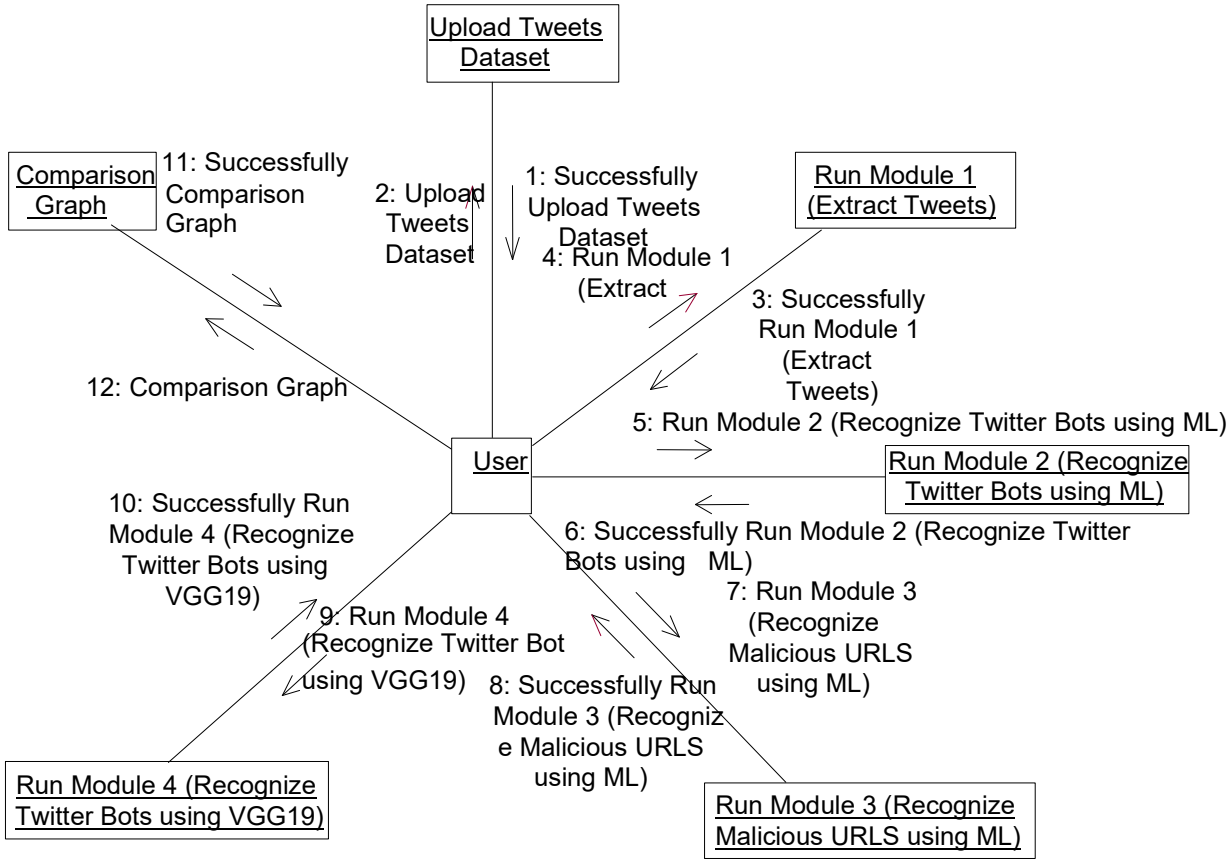


**Fig 3.3.1.4: Collaboration Diagram**

## E. Component Diagram

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems

Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. This illustrates the service consumer - service provider relationship between the two components.

**Component diagram:**

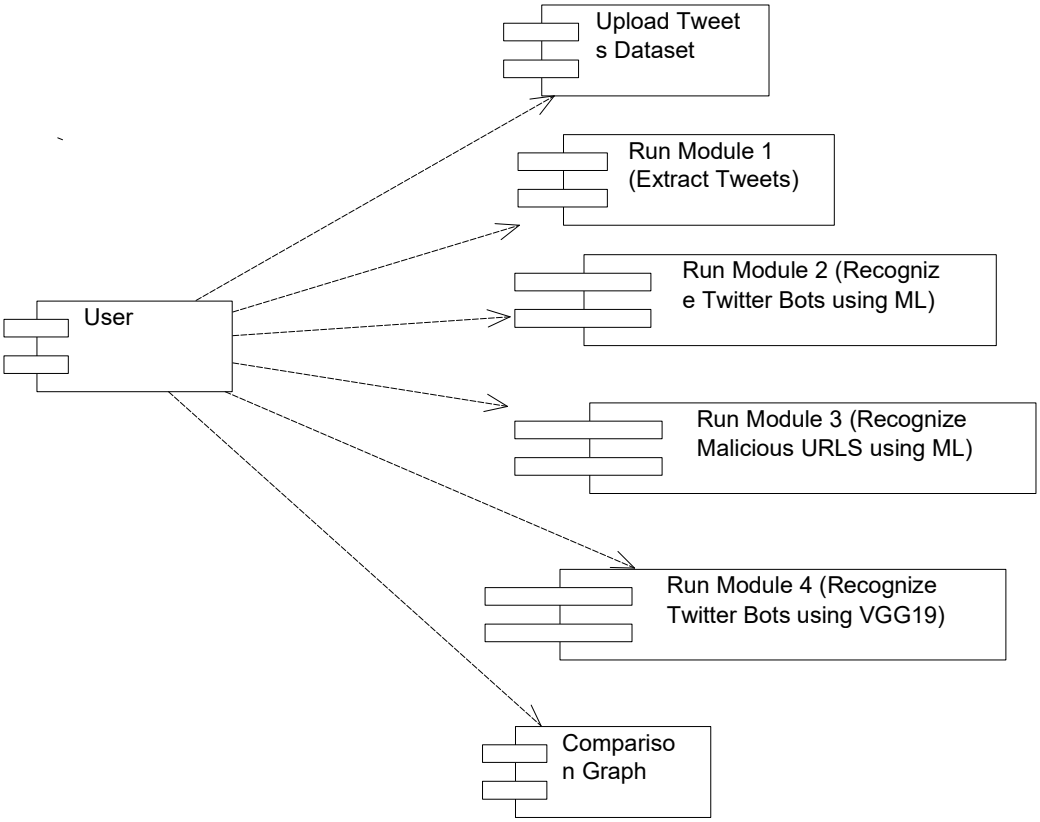


**Fig 3.3.1.5: Component Diagram**

## F. Deployment Diagram

A **deployment diagram** in the Unified Modeling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

**Deployment diagram:**

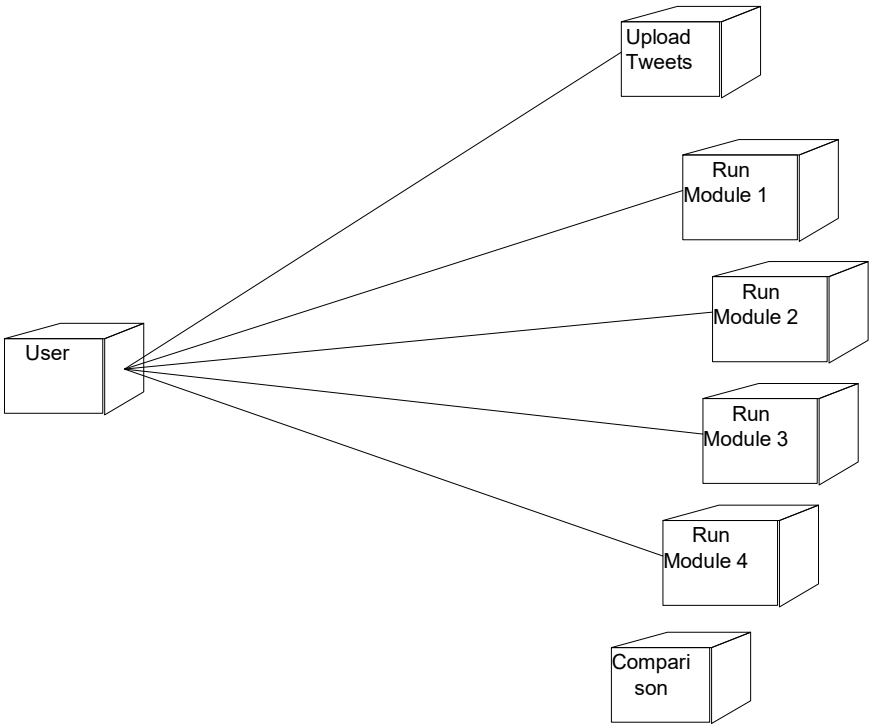


**Fig 3.3.1.6: Deployment Diagram**

G. **Activity diagram:**

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. It is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system.

So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent.

**Activity diagram:**

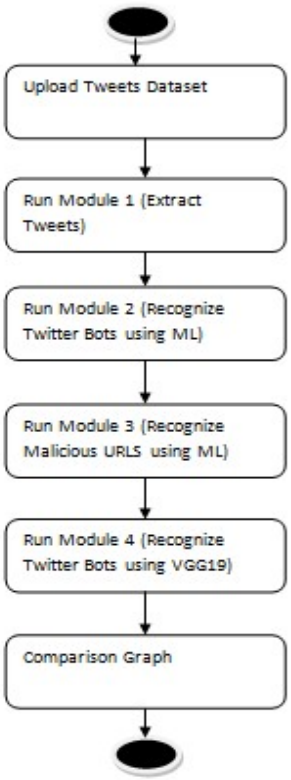


**Fig 3.3.1.7: Activity Diagram**

H. **Data Flow Diagram:**

Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out in precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.

As the name suggests, Data Flow Diagram (DFD) is an illustration that explicates the passage of information in a process. A DFD can be easily drawn using simple symbols. Additionally, complicated processes can be easily automated by creating DFDs using

easy-to-use, free downloadable diagramming tools. A DFD is a model for constructing and analyzing information processes. DFD illustrates the flow of information in a process depending upon the inputs and outputs. A DFD can also be referred to as a Process Model. A DFD demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.

**Dataflow Diagram**:


**Fig 3.3.1.8: Data Flow Diagram**

## 3.4 STEPWISE IMPLEMENTATION AND TESTING

**Implementation and Testing:**

Implementation is one of the most important tasks in project is the phase in which one has to be cautions because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

**Implementation**

The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may be requiring extensive user training. The initial parameters of the system should be modifies as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or dot matrix printer, which is available at the disposal of the user. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

**Testing**

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data should be chosen such that it passed through all possible condition. Actually testing is the state of implementation which aimed at ensuring that the system works accurately and efficiently before the actual operation commence. The following is the description of the testing strategies, which were carried out during the testing period.

### System Testing

Testing has become an integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to user to use the software must be tested whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data are repeated. Thus the code was exhaustively checked for all possible correct data and the outcomes were also checked.

### Module Testing

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example the job classification module is tested separately. This module is tested with different job and its approximate execution time and the result of the test is compared with the results that are prepared manually. The comparison shows that the results proposed system works efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

### Integration Testing

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus the mapping of jobs with resources is done correctly by the system.

**Acceptance Testing**

When that user fined no major problems with its accuracy, the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

**Modules Information:**

- **Module 1(Tweet Extraction):** Using this module we will extract tweets from online or offline and every time internet will not be available so we are using offline KAGGLE tweets dataset. By using this module we will read or extract all tweets from dataset. If we are downloading tweets online then we need WOEID from twitter but we are using dataset so WOEID not require.
- **Module 2 (Recognize Twitter Bots using ML):** In this module we are extracting features from tweets such as Activity, Anonymity and Amplification. Activity refers to finding tweet frequency and Anonymity refers to account information and Amplification refers to retweet count. By using above 3 concepts author is checking whether account is normal or bot.
- **Module 3 (Recognize Malicious URLS using ML):** Using this module we analyse all tweets and check if tweet contains more number of URLs then it will consider as malicious URLS and below code with screen shots showing method3 checking for malicious URL.
- **Module 4 (Recognize Twitter Bots using VGG19) :** Using this module we analyse all tweets and check Recognize Twitter Bots using VGG19
- **Comparison Graph:** Using this module comparision graph between Recognize Twitter Bots using ML, Recognize Malicious URLS using ML and Recognize Twitter Bots using VGG19

| Test Case Id | Test Case Name | Test Case Desc. | Test Steps | | | Test Case Status | Test Priority |
|---|---|---|---|---|---|---|---|
| | | | Step | Expected | Actual | | |
| O1 | Upload tweet bot dataset | Verify Dataset updated or not | If dataset is May not be Uploaded | we can't do further operations | we can do further operations | High | High |
| 02 | Run module 1(extract tweets) | Verify tweets are extracted or not | If tweets are not extracted | we cannot do any further operations | we can do further operations | High | High |
| 03 | Run module 2(Recognize Twitter Bots using ML) | Verify (Recognize Twitter Bots using ML)or not | If recognisation is not done | We cannot run operation | We can Run the Operation | High | High |
| 04 | Run module 3 (recognize URL using ML) | Verify the URLs are recognized or not using ML | If UMLs are not recognized | We cannot run operation | We can Run the Operation | High | High |
| 05 | Run Module 4 (Recognize Twitter Bots using VGG19) | Verify Run Module 4 (Recognize Twitter Bots using VGG19)or not | If Run Module 4 Twitter Bots using VGG19 May not recognized | We cannot run operation | We can Run the Operation | High | High |
| 06 | Comparison Graph | Verify Comparison Graph or not | If Comparison Graph May not Plot | We cannot run operation | We can Run the Operation | High | High |

**Table 3.4.1: Module Execution**

## 3.5 MODEL ARCHITECTURE:

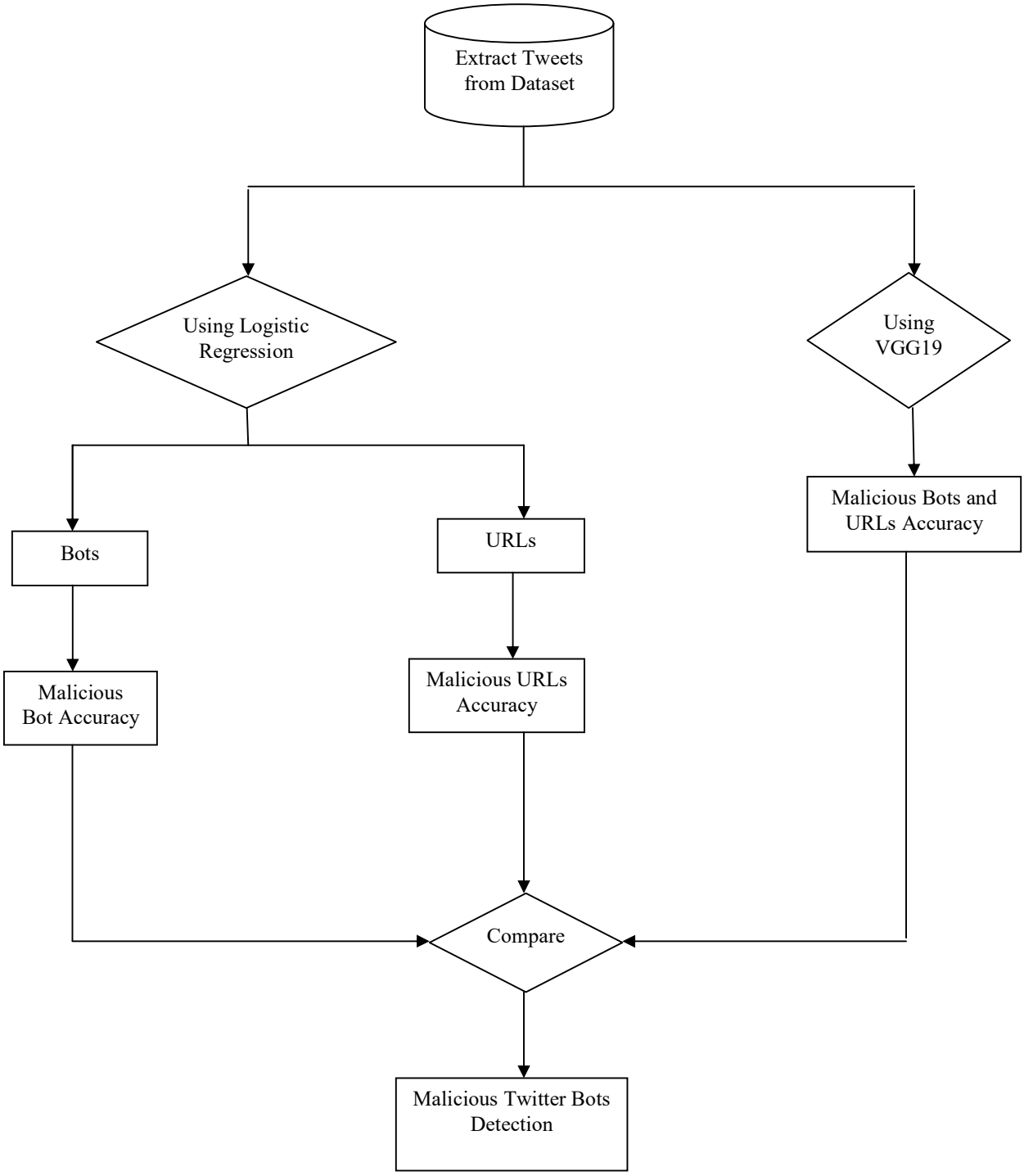


**Fig 3.5.1: Model Architecture**

## 3.6 SYSTEM REQUIREMENT:

### HARDWARE REQUIREMENTS:

- Processor - Intel i3(min)
- Speed - 1.1 Ghz
- RAM - 4GB(min)
- Hard Disk - 500 GB
- Key Board - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse
- Monitor - SVGA

### SOFTWARE REQUIREMENTS:

- Operating System - Windows10(min)
- Programming Language - Python 3.7

# CHAPTER 4
# RESULTS AND DISCUSSION

# CHAPTER 4
# RESULTS AND DISCUSSION

## 4.1 OUTPUT SCREENS:

To run the modules for execution and get output screen a bat file is created. When we click on it the execution starts and we get the below screen

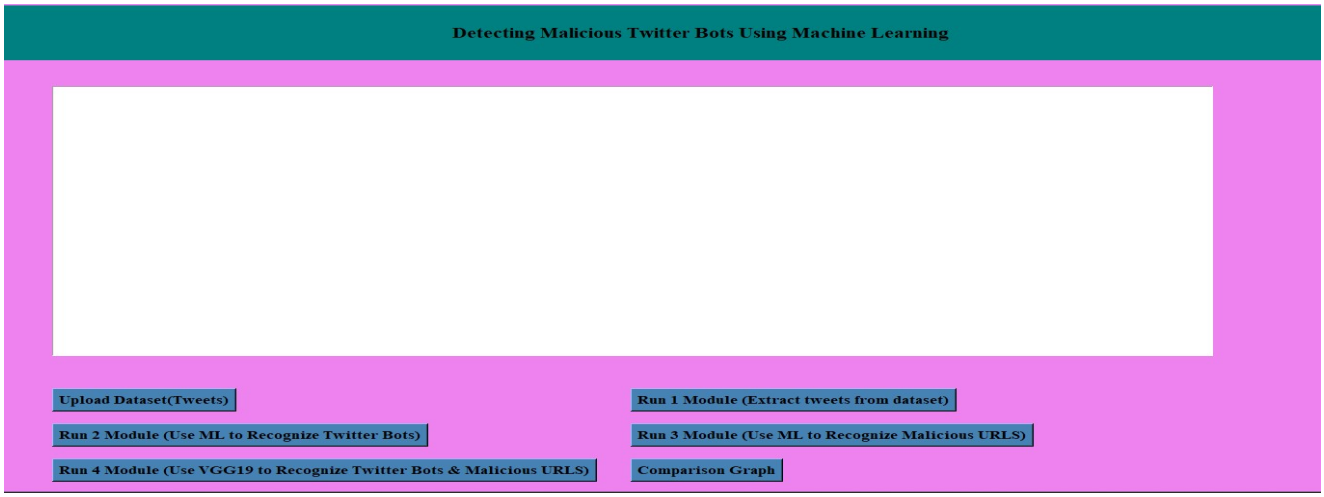


**Fig 4.1.1: It represents the output screen**

To upload the dataset we click the button named Upload Dataset (Tweets). The dataset is uploaded successfully.

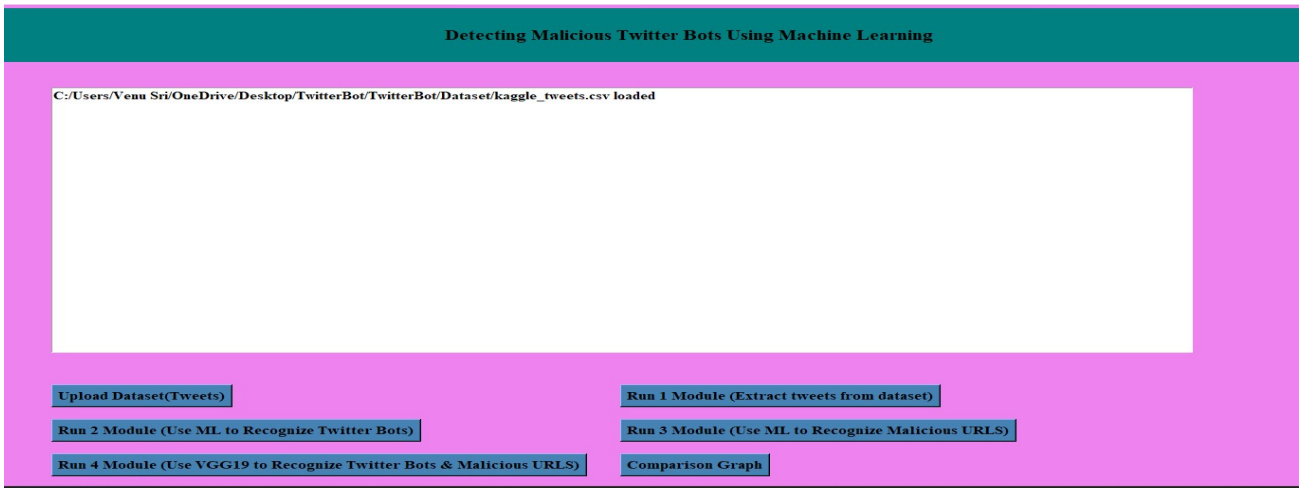


**Fig 4.1.2: It represents the image when dataset is uploaded**

When the dataset is uploaded successfully, we run first module to extract tweets from the dataset and read them.

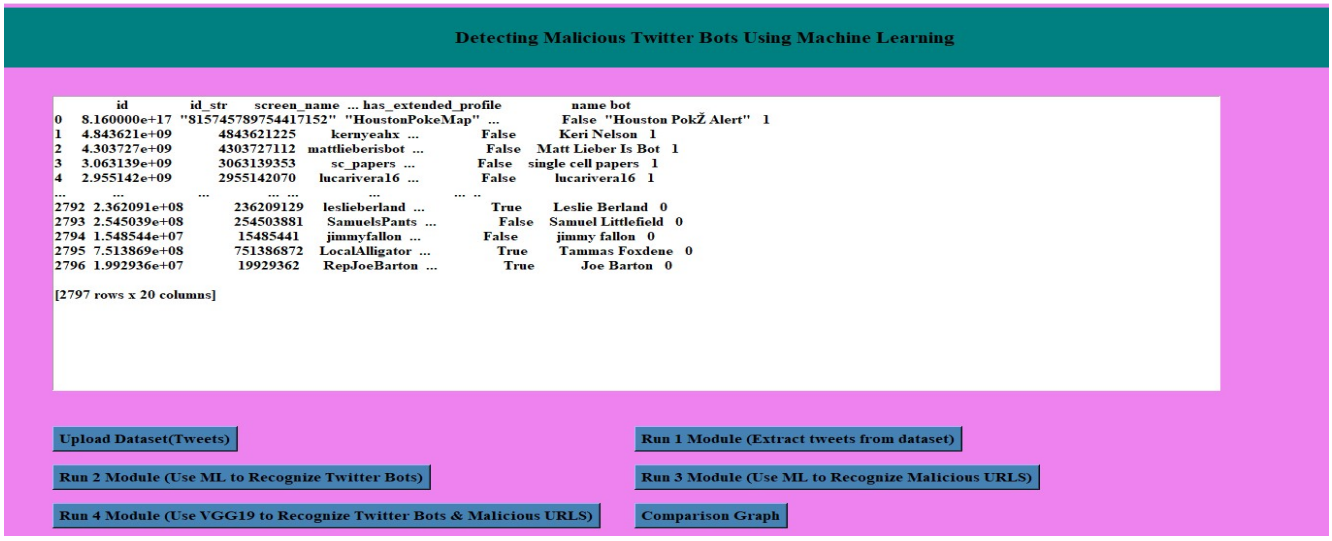


**Fig 4.1.3: It represents the extraction of tweets from dataset**

"Run 2 Module" to recognize the bots from real users using machine learning algorithm i.e; logistic regression. Using this, we got 71% accuracy.

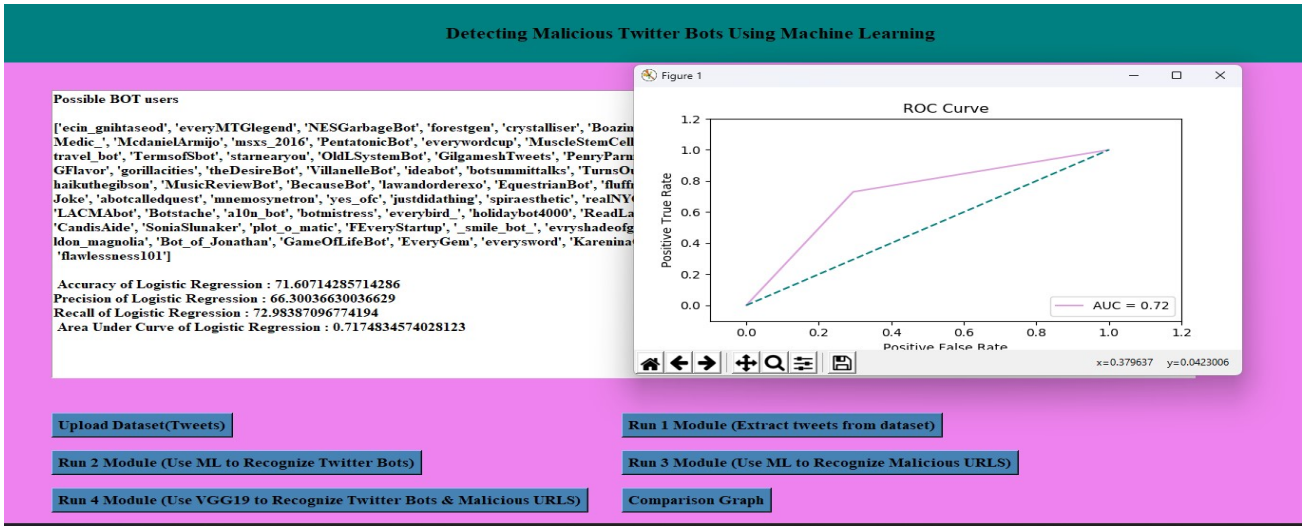


**Fig 4.1.4: It presents the ROC for recognition of twitter bots**

"Run 3 Module" to recognize malicious URLS using logistic machine learning algorithm. Using this we got accuracy of 73%.

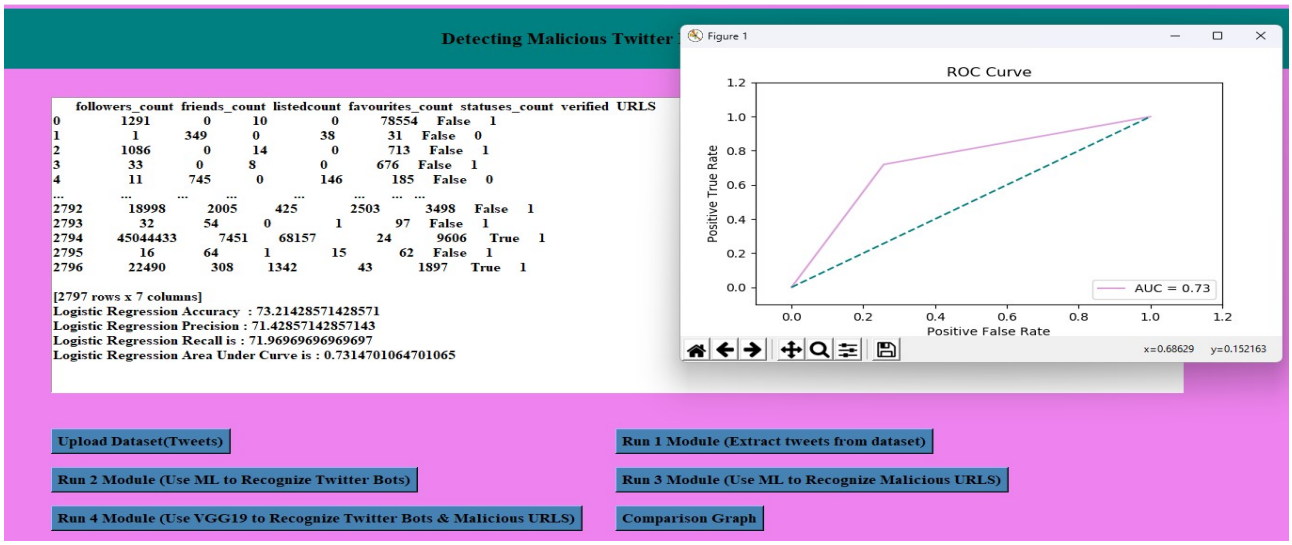


**Fig 4.1.5: It represents the ROC of recognition of malicious URLS**

"Run 4 Module" to detect bots and URLS at a time using deep learning algorithm VGG19.Using this we got accuracy of 87.5%.

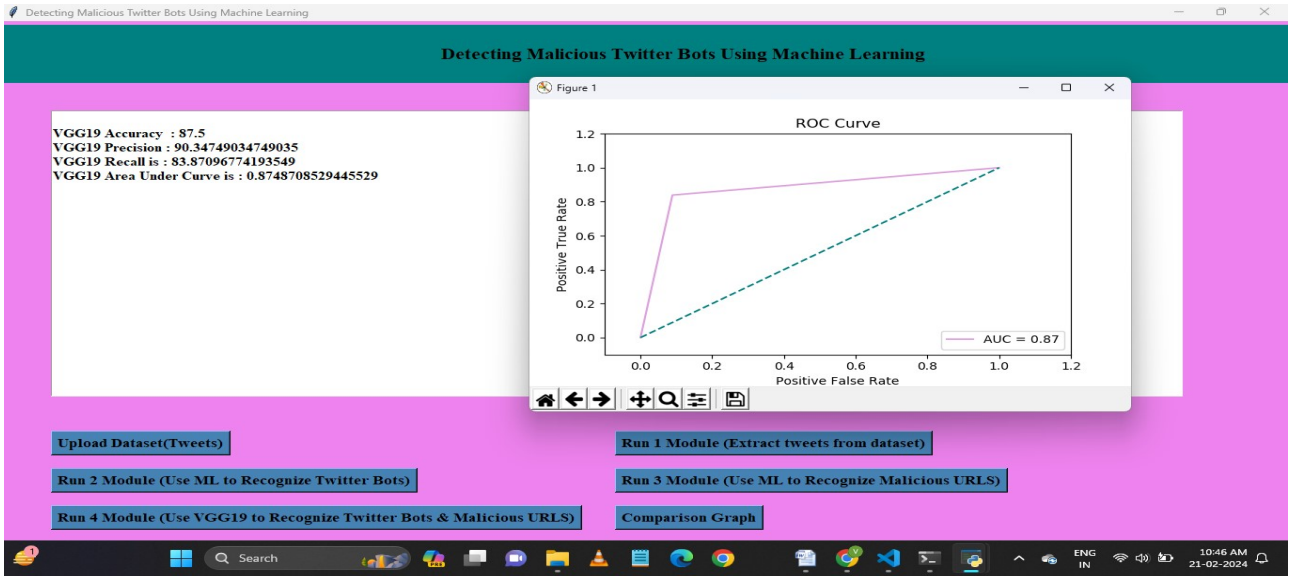


**Fig 4.1.6: It represents the ROC for recognition of both malicious twitter bots and malicious URLS**

To compare the results obtained by each module, we use “Comparison Graph” button and check which algorithm is best suited for detecting malicious twitter bots. By comparing it in a graph we got VGG19 with 87.5%.

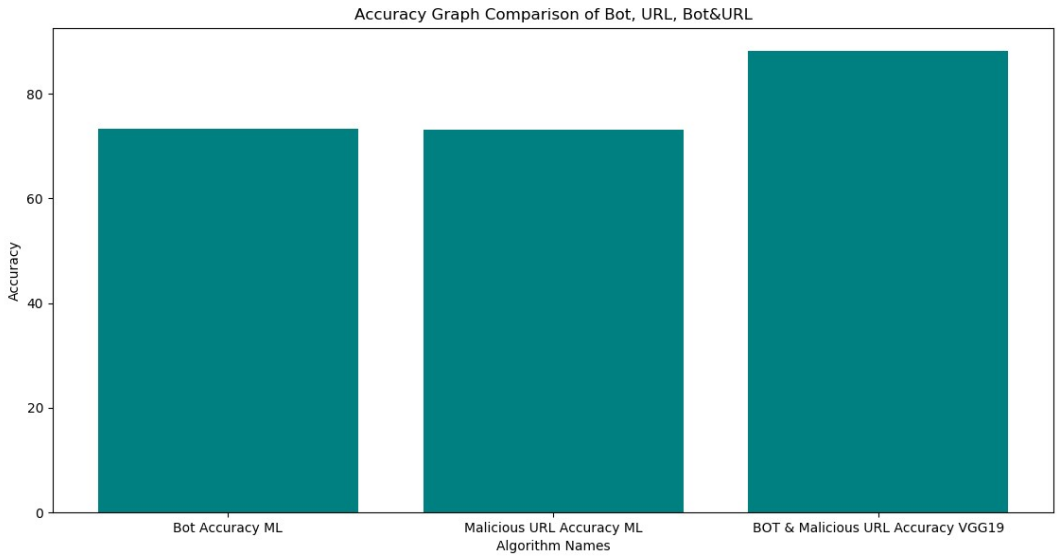


**Fig 4.1.7: It represents the Accuracy comparison graphs**

## 4.2 PERFORMANCE METRICS:

| **Algorithms** | **Bot Using ML** | **URL Using ML** | **Bot and URL using VGG19** |
|---|---|---|---|
| Accuracy | 71.60 | 73.21 | 87.5 |
| Precision | 66.30 | 71.42 | 90.34 |
| Recall | 72.98 | 71.96 | 83.87 |
| AUC | 0.71 | 0.73 | 0.83 |

**Table 4.2.1 shows the graphical representation of the accuracy, precision, recall, AUC of the algorithms.**

# CHAPTER 5
# CONCLUSION

# CHAPTER 5
# CONCLUSION

## 5.1 CONCLUSION AND FUTURE ENHANCEMENT:

- Developing a malicious Twitter bots detection system using machine learning is vital for preserving the integrity of online platforms. It addresses critical issues such as the spread of fake news, cyber-bullying, and threats to digital democracy, making online spaces safer and more reliable.
- We created a calculation in our investigate that distinguishes Twitter bots. Pack of words strategy for prepare information was the leading show VGG19 having tall precision compared to calculated relapse. Hence, word calculations were utilized to real-time information and the Twitter bots have been identified successfully. By using VGG19 along with convolutional neural network architecture we got a high accuracy when compared to Logistic regression machine learning algorithm.
- VGG19, a convolutional neural network architecture, for feature extraction from Twitter bot images followed by Logistic Regression for classification can be a best approach.
- So, by using VGG19 architecture we can identify and classify any number of images and any type of images and can detect malicious bots .

# REFERENCES

# REFERENCE

[1] Van Der Walt,E.,& Eloff,J.(2018).Using machine learning to detect fake identities:botsvshumans.*IEEEaccess*,*6*,6540-6549

[2] Nasim, M., Nguyen, A., Lothian, N., Cope, R., & Mitchell, L. (2018, April). Real-time detection of content polluters in partially observable Twitter networks. In *Companion Proceedings of the The Web Conference 2018* (pp. 1331-1339).

[3] Khalil, A., Hajjdiab, H., & Al-Qirim, N. (2017). Detecting fake followers in twitter: A machine learning approach. *International Journal of Machine Learning and Computing*, *7*(6), 198-202.

[4] Wetstone, J. H.,& Nayyar, S. R.(2017). I Spot a Bot: Building a binary classifier to detect bots on Twitter. *CS 229Final Project Report*.

[5] Karataş,A.,&Şahin,S. (2017).A review on social bot detection techniques and research directions.

[6] Chavoshi, N., Hamooni, H., & Mueen, A. (2016). Identifying correlated bots in twitter. In *Social Informatics: 8th International Conference, SocInfo 2016, Bellevue, WA, USA, November 11-14, 2016, Proceedings, Part II 8* (pp. 14-21). Springer International Publishing.

[7] Perdana,R.S.,Muliawati,T.H.,&Alexandro,R.(2015).Bot spammer detection in Twitter using tweet similarity and time interval entropy. *JurnalIlmuKomputer dan Informasi*, *8*(1), 19-25.

**CODE:**

```
from tkinter import messagebox
from tkinter import *
from tkinter import simpledialog
import tkinter
from tkinter import filedialog
from tkinter.filedialog import askopenfilename
import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd
import re
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from collections import defaultdict
from sklearn import metrics
from keras.applications import VGG19
from keras.utils.np_utils import to_categorical
from keras.layers import  MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D
from keras.models import Sequential, load_model
from keras.callbacks import ModelCheckpoint
```

```
main = tkinter.Tk()
main.title("Detecting Malicious Twitter Bots Using Machine Learning") #designing main screen
main.geometry("1300x1200")
global filename
global dataset, X, Y, X_train, X_test, y_train, y_test
global graph
words = ['bot','cannabis','tweet me','mishear','follow me','updates','every','gorilla','forget']
def getFrequency(bow):
    count = 0
    for i in range(len(words)):
        if words[i] in bow:
            count = count + bow.get(words[i])
    return count
def uploadDataset():
    global filename
    text.delete('1.0', END)
    filename = filedialog.askopenfilename(initialdir="Dataset")
    text.insert(END,filename+" loaded\n\n")
def runModule1():
    global dataset
    text.delete('1.0', END)
    dataset = pd.read_csv(filename)
    text.insert(END,str(dataset))
def runModule2():
    global graph
    global X, Y, X_train, X_test, y_train, y_test
    graph = []
    text.delete('1.0', END)
    train = dataset[['screen_name','status','name','followers_count', 'friends_count', 'listedcount',
'favourites_count', 'statuses_count', 'verified']]
    details = train.values
```

```
    text.insert(END,"Possible BOT users\n\n")
    users = []
    for i in range(len(details)):
        screen = details[i,0]
        status = details[i,1]
        name = details[i,2]
        followers = int(details[i,3])
        friends = int(details[i,4])
        listed = int(details[i,5])
        favourite = int(details[i,6])
        status_count = int(details[i,7])
        verified = details[i,8]
        if not verified: #check user not verified
            bow = defaultdict(int) #bag of words
            data = str(screen)+" "+str(name)+" "+str(status)#checking screen name, tweets and name
            data = data.lower().strip("\n").strip()
            data = re.findall(r'\w+', data)
            for j in range(len(data)):
                bow[data[j]] += 1  #adding each word frequency to bag of words
            frequency = getFrequency(bow) #getting frequency of BOTS words
            if frequency > 0 and listed < 16000 and followers < 200: #if condition true then its bots
                users.append(screen)
    text.insert(END,str(users)+"\n")
    train_attr = dataset[
        ['followers_count',    'friends_count',    'listedcount',    'favourites_count',    'statuses_count',
'verified']]
    train_label = dataset[['bot']]
    X = train_attr
    Y = train_label.as_matrix()
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
    print(X.shape)
```

```
    print(X_train.shape)
    logreg = LogisticRegression().fit(X_train, y_train)#logistic regression object
    actual = y_test
    pred = logreg.predict(X_test)
    accuracy = accuracy_score(actual, pred) * 100
    graph.append(accuracy)
    precision = precision_score(actual, pred) * 100
    recall = recall_score(actual, pred) * 100
    f1 = f1_score(actual, pred)
    auc = roc_auc_score(actual, pred)
    text.insert(END,'\nLogistic Regression Accuracy  : '+str(accuracy)+"\n")
    text.insert(END,'Logistic Regression Precision : '+str(precision)+"\n")
    text.insert(END,'Logistic Regression Recall is : '+str(recall)+"\n")
    text.insert(END,'Logistic Regression Area Under Curve is : '+str(auc))
    fpr, tpr, thresholds = metrics.roc_curve(actual, pred)
    auc = metrics.auc(fpr, tpr)
    plt.title('ROC')
    plt.plot(fpr, tpr, 'b',
    label='AUC = %0.2f'% auc)
    plt.legend(loc='lower right')
    plt.plot([0,1],[0,1],'r--')
    plt.xlim([-0.1,1.2])
    plt.ylim([-0.1,1.2])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
def runModule3():
    text.delete('1.0', END)
    urls = []
    details = dataset.values
    for i in range(len(details)):#checking URLS in tweets
```

```
        tweets = details[i,14]
        if 'http' in str(tweets):
            urls.append(1)
        else:
            urls.append(0)
    train_attr = dataset[
        ['followers_count', 'friends_count', 'listedcount', 'favourites_count', 'statuses_count',
'verified']]
    train_attr["URLS"] = urls #adding URLS to training dataset
    text.insert(END,str(train_attr))
    train_label = dataset[['bot']]
    X1 = train_attr
    Y1 = np.asarray(train_label)
    X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, Y1, test_size=0.2)
    logreg = LogisticRegression().fit(X_train1, y_train1) #logistic regression object
    actual = y_test1
    pred = logreg.predict(X_test1)
    accuracy = accuracy_score(actual, pred) * 100
    graph.append(accuracy)
    precision = precision_score(actual, pred) * 100
    recall = recall_score(actual, pred) * 100
    f1 = f1_score(actual, pred)
    auc = roc_auc_score(actual, pred)
    text.insert(END,'\nLogistic Regression Accuracy  : '+str(accuracy)+"\n")
    text.insert(END,'Logistic Regression Precision : '+str(precision)+"\n")
    text.insert(END,'Logistic Regression Recall is : '+str(recall)+"\n")
    text.insert(END,'Logistic Regression Area Under Curve is : '+str(auc))
    fpr, tpr, thresholds = metrics.roc_curve(actual, pred)
    auc = metrics.auc(fpr, tpr)
    plt.title('ROC')
    plt.plot(fpr, tpr, 'b',
```

```
    label='AUC = %0.2f'% auc)
    plt.legend(loc='lower right')
    plt.plot([0,1],[0,1],'r--')
    plt.xlim([-0.1,1.2])
    plt.ylim([-0.1,1.2])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
def runModule4():
    global X, Y, X_train, X_test, y_train, y_test
    text.delete('1.0', END)
    X_train = X_train.values
    X_test = X_test.values
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1, 1))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1, 1))
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)
    if os.path.exists("model/vgg19_weights.hdf5") == True:
        vgg_model = load_model("model/vgg19_weights.hdf5")
    else:
        #now training VGG19 on Eyepacs dataset
        vgg = VGG19(include_top=False, weights="imagenet", input_shape=(X_train.shape[1],
X_train.shape[2], X_train.shape[3]))
        for layer in vgg.layers:
            layer.trainable = False
        vgg_model = Sequential()
        vgg_model.add(vgg)#adding VGG as transfer learning
        #now adding new layers to VGG for classifying eyepacs dataset
        vgg_model.add(Convolution2D(32, (1, 1), input_shape = (X_train.shape[1],
X_train.shape[2], X_train.shape[3]), activation = 'relu'))
        vgg_model.add(MaxPooling2D(pool_size = (1, 1)))
```

```
        vgg_model.add(Convolution2D(32, (1, 1), activation = 'relu'))
        vgg_model.add(MaxPooling2D(pool_size = (1, 1)))
        vgg_model.add(Flatten())
        vgg_model.add(Dense(units = 256, activation = 'relu'))
        vgg_model.add(Dense(units = y_train.shape[1], activation = 'softmax'))
        vgg_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        model_check_point = ModelCheckpoint(filepath='model/vgg19_weights.hdf5', verbose = 1, save_best_only = True)
        hist = vgg_model.fit(X_train, y_train, batch_size=16, epochs=40, shuffle=True, callbacks=[model_check_point], verbose=1, validation_data=(X_test, y_test))
    predict = vgg_model.predict(X_test) #now perform prediction on test data
    predict = np.argmax(predict, axis=1)
    y_test1 = np.argmax(y_test, axis=1)
    accuracy = accuracy_score(y_test1, predict) * 100
    graph.append(accuracy)
    precision = precision_score(y_test1, predict) * 100
    recall = recall_score(y_test1, predict) * 100
    f1 = f1_score(y_test1, predict)
    auc = roc_auc_score(y_test1, predict)
    text.insert(END,'\nVGG19 Accuracy  : '+str(accuracy)+"\n")
    text.insert(END,'VGG19 Precision : '+str(precision)+"\n")
    text.insert(END,'VGG19 Recall is : '+str(recall)+"\n")
    text.insert(END,'VGG19 Area Under Curve is : '+str(auc))
    fpr, tpr, thresholds = metrics.roc_curve(y_test1, predict)
    auc = metrics.auc(fpr, tpr)
    plt.title('ROC')
    plt.plot(fpr, tpr, 'b',
    label='AUC = %0.2f'% auc)
    plt.legend(loc='lower right')
    plt.plot([0,1],[0,1],'r--')
```

```
    plt.xlim([-0.1,1.2])
    plt.ylim([-0.1,1.2])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
def plotGraph():
    global graph
    height = graph
    bars = ('ML Bot Accuracy','ML URL Accuracy','VGG19 BOT & URL Accuracy')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.xlabel("Algorithm Names")
    plt.ylabel("Accuracy")
    plt.title("Accuracy Comparison Graph")
    plt.show()
font = ('times', 16, 'bold')
title = Label(main, text='Detecting Malicious Twitter Bots Using Machine Learning')
title.config(bg='goldenrod2', fg='black')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0,y=5)
font1 = ('times', 12, 'bold')
text=Text(main,height=20,width=150)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=50,y=120)
text.config(font=font1)
font1 = ('times', 13, 'bold')
uploadButton = Button(main, text="Upload Tweets Dataset", command=uploadDataset,
bg='#ffb3fe')
```

```
uploadButton.place(x=50,y=550)
uploadButton.config(font=font1)
module1Button = Button(main, text="Run Module 1 (Extract Tweets)", command=runModule1, bg='#ffb3fe')
module1Button.place(x=520,y=550)
module1Button.config(font=font1)
module2Button = Button(main, text="Run Module 2 (Recognize Twitter Bots using ML)", command=runModule2, bg='#ffb3fe')
module2Button.place(x=50,y=600)
module2Button.config(font=font1)
module3Button = Button(main, text="Run Module 3 (Recognize Malicious URLS using ML)", command=runModule3, bg='#ffb3fe')
module3Button.place(x=520,y=600)
module3Button.config(font=font1)
module3Button = Button(main, text="Run Module 4 (Recognize Twitter Bots using VGG19)", command=runModule4, bg='#ffb3fe')
module3Button.place(x=50,y=650)
module3Button.config(font=font1)
module3Button = Button(main, text="Comparison Graph", command=plotGraph, bg='#ffb3fe')
module3Button.place(x=520,y=650)
module3Button.config(font=font1)
main.config(bg='SpringGreen2')
main.mainloop()
```

**PAPER PUBLICATION PROOF:**

Paper Published - IJRASET59085 Inbox

noreply@i... Yesterday

to me, ijraset

iJRASET
International Journal For Research in Applied Science and Engineering Technology

**PAPER PUBLISHED**

**About Us | Aim & Scope | Check Paper Status**

**Dear Author/Research Scholar,**

I am pleased to inform you that your manuscript **"Detecting Malicious Twitter Bots using Machine Learning"** has been successfully published in Volume 12 Issue III March 2024 .

**Paper Title :** Detecting Malicious Twitter Bots using Machine Learning
**Paper ID :** IJRASET59085
**Paper Status :** Paper Published and It is accessible online on our website.

You can track the status of your manuscript by navigating to https://www.ijraset.com/status.php . We thank you for your interest, and hope you choose to submit another article for review in the future.

Please submit your photograph on the given link: https://www.ijraset.com/add-paper-details.php . It will be displayed on our website along with your paper.

**With Warm Regards**
**Editor-In Chief**
**IJRASET Publications**
https://www.ijraset.com/, **Email id:** ijraset@gmail.com

**GITHUB LINK:**

https://github.com/sneha9989/Major-project-MaliciousTwitterBot