**Detecting Malicious Twitter Bots Using Machine Learning**

1. **INTRODUCTION:**

Twitter is one of the fastest-growing social media platforms. It enables users to exchange news, express themselves, and debate current events. Users may follow individuals who share their interests or have similar viewpoints. Users may send tweets to their followers right away. Re-tweeting allows the content to reach a wider audience. During live events such as sports or award ceremonies, the number of tweets spikes. Smartphones and PCs can both access Twitter. Paid promotions may result in significant income creation as well as an increase in product sales. Students may use Twitter to learn more about the subjects that are covered in class. The message that is shared with followers is referred to as a tweet. The tweet should be short and to the point, with a maximum of 140 characters. The hashtag (#) is used to locate and follow a certain subject. When a hashtag gets popular, it is referred to as a trending topic. Twitter connections are bidirectional, meaning that a person may have both followers and followers. If you follow someone on Twitter, you will be able to view all of their tweets if the account is public; but this does not imply that he or she will be able to see your tweets. If you follow someone back, they will be able to view your tweets.

* 1. **Objective of the project:**

Undoubtedly, social media, such as Facebook and Twitter, constitute a major part of our everyday life due to the incredible possibilities they offer to their users. However, Twitter and generally [online social networks](https://www.sciencedirect.com/topics/computer-science/online-social-networks) (OSNs) are increasingly used by automated accounts, widely known as bots, due to their immense popularity across a wide range of user categories. Their main purpose is the dissemination of fake news, the promotion of specific ideas and products, the manipulation of the stock market and even the diffusion of sexually explicit material. Therefore, the early detection of bots in social media is quite essential. In this paper, two methods are introduced targeting this that are mainly based on [Natural Language Processing](https://www.sciencedirect.com/topics/engineering/natural-language-processing) (NLP) to distinguish legitimate users from bots. In the first method, a feature extraction approach is proposed for identifying accounts posting automated messages. After applying feature selection techniques and dealing with imbalanced datasets, the subset of features selected is fed in [machine learning algorithms](https://www.sciencedirect.com/topics/engineering/machine-learning-algorithm). In the second method, a deep learning architecture is proposed to identify whether tweets have been posted by real users or generated by bots. To the best of the authors’ knowledge, there is no prior work on using an attention mechanism for identifying bots. The introduced approaches have been evaluated over a series of experiments using two large real [Twitter datasets](https://www.sciencedirect.com/topics/computer-science/twitter-dataset) and demonstrate valuable advantages over other existing techniques targeting the identification of malicious users in social media.

**2.LITERATURE SURVEY:**

**“Using machine learning to detect fake identities: bots vs humans.”**

T There is 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.

**“Real-time detection of content polluters in partially observable Twitter networks.”**

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

**“A Machine Learning Approach. International Journal of Machine Learning and Computing”**

International Journal of Machine Learning and Computing (IJMLC***)*** is an international academic open access journal which gains a foothold in Singapore, Asia and opens to the world. It aims to promote the integration of machine learning and computing. The focus is to publish papers on state-of-the-art machine learning and computing. Submitted papers will be reviewed by technical committees of the Journal and Association. The audience includes researchers, managers and operators for machine learning and computing as well as designers and developers.

All submitted articles should report original, previously unpublished research results, experimental or theoretical, and will be peer-reviewed. Articles submitted to the journal should meet these criteria and must not be under consideration for publication elsewhere. Manuscripts should follow the style of the journal and are subject to both review and editing.

“**I Spot a Bot: Building a binary classifier to detect bots on Twitter.”**

It has been estimated that up to 50% of the activity on Twitter comes from bots [1]: algorithmically-automated accounts created to advertize products, distribute spam, or sway public opinion. It is perhaps this last intent that is most alarming; studies have found that up to 20% of the Twitter activity related to the 2016 U.S. presidential election came from suspected bot accounts, and there has been evidence of bots used to spread false rumors about French presidential candidate Emmanuel Macron and to escalate a recent conflict in Qatar [1]. Detecting bots is necessary in order to identify bad actors in the “Twitterverse” and protect genuine users from misinformation and malicious intents. This has been an area of research for several years, but current algorithms still lag in performance relative to humans [2]. Given a Twitter user’s profile and tweet history, our project was to build a binary classifier that identifies a given user as “bot” or “human.” The end-user application for a classifier such as this one would be a web plug-in for the browser that can score a given account in real-time (See page 5 for mock-ups). All of the raw inputs required to classify a public Twitter account via our algorithm are available for download from the Twitter API; in fact, our check\_screenname.py program is a working prototype that uses the API to classify a given Twitter user handle within seconds. It is our opinion that a product like this is sorely needed for the average Twitter consumer**.**

**“A Review on Social Bot Detection Techniques and Research Directions. In Proc.”**

The rise of web services and popularity of online social networks (OSN) like Facebook, Twitter, LinkedIn etc. have led to the rise of unwelcome social bots as automated social actors. Those actors can play many malicious roles including infiltrators of human conversations, scammers, impersonators, misinformation disseminators, stock market manipulators, astroturfers, and any content polluter (spammers, malware spreaders) and so on. It is undeniable that social bots have major importance on social networks. Therefore, this paper reveals the potential hazards of malicious social bots, reviews the detection techniques within a methodological categorization and proposes avenues for future research.

**“ Identifying correlated bots in twitter. In International Conference on Social Informatics”**

We develop a technique to identify abnormally correlated user accounts in Twitter, which are very unlikely to be human operated. This new approach of bot detection considers cross-correlating user activities and requires no labeled data, as opposed to existing bot detection techniques that consider users independently, and require large amount of recently labeled data. Our system uses a lag-sensitive hashing technique and a warping-invariant correlation measure to quickly organize the user accounts in clusters of abnormally correlated accounts. Our method is 94 % precise and detects unique bots that other methods cannot detect. Our system produces daily reports on bots at a rate of several hundred bots per day. The reports are available online for further analysis.

**“Bot spammer detection in Twitter using tweet similarity and time interval entropy”**

The popularity of Twitter has attracted spammers to disseminate large amount of spam messages. Preliminary studies had shown that most spam messages were produced automatically by bot. Therefore bot spammer detection can reduce the number of spam messages in Twitter significantly. However, to the best of our knowledge, few researches have focused in detecting Twitter bot spammer. Thus, this paper proposes a novel approach to differentiate between bot spammer and legitimate user accounts using time interval entropy and tweet similarity. Timestamp collections are utilized to calculate the time interval entropy of each user. Uni-gram matching-based similarity will be used to calculate tweet similarity. Datasets are crawled from Twitter containing both normal and spammer accounts. Experimental results showed that legitimate user may exhibit regular behavior in posting tweet as bot spammer. Several legitimate users are also detected to post similar tweets. Therefore it is less optimal to detect bot spammer using one of those features only. However, combination of both features gives better classification result. Precision, recall, and f-measure of the proposed method reached 85,71%, 94,74% and 90% respectively. It outperforms precision, recall, and f-measure of method which only uses either time interval entropy or tweet similarity.

**“Tweets as impact indicators: Examining the implications of automated bot accounts on T witter. Journal of the Association for Information Science and Technology”**

This brief communication presents preliminary findings on automated Twitter accounts distributing links to scientific articles deposited on the preprint repository arXiv. It discusses the implication of the presence of such bots from the perspective of social media metrics altmetrics, where mentions of scholarly documents on Twitter have been suggested as a means of measuring impact that is both broader and timelier than citations. Our results show that automated Twitter accounts create a considerable amount of tweets to scientific articles and that they behave differently than common social bots, which has critical implications for the use of raw tweet counts in research evaluation and assessment. We discuss some definitions of Twitter cyborgs and bots in scholarly communication and propose distinguishing between different levels of engagement-that is, differentiating between tweeting only bibliographic information to discussing or commenting on the content of a scientific work.

**3.SYSTEM ANALYSIS**

**Existing system:**

Due to the vast array of opportunities they present, social media platforms like Facebook and Twitter have become deeply embedded in our daily lives. However, due to the widespread appeal of Twitter and other OSNs, automated accounts commonly referred to as bots are rapidly making use of them.The train data has many attributes. The required features are extracted using Spearman correlation method. Random Forest.  **Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset*.***

**Disadvantages**:

* Low security

**Proposal system:**

There are numerous characteristics to the train data. The necessary functionality is retrieved using the Spear-man correlation technique. Logistic Regression algorithm by Machine learning.Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. Real-time data as displayed are used as the optimum learning model. Data are preprocessed and zero values are eliminated using pandas (tool for pre-processing). The dataset is trained and the test data set is the actual Twitter data.The output is in shape 0 or1. We developed an algorithm in our research that identifies Twitter bots. indicate Malicious URL and in above screen we can see URL prediction accuracy as 73%. we got ML accuracy of Logistic Regression is 74% Thus, word algorithms were used to real-time data and the Twitter bots have been detected effectively

**Advantages:**

* High security
* Hi accuracy
* High efficiency

**Modules Information:**

In this paper author is using machine learning algorithm such as Logistic Regression to detect BOTS from tweets and to implement this paper author has given 3 modules

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

**3.3. PROCESS MODEL USED WITH JUSTIFICATION**

**SDLC (Umbrella Model):**

**Umbrella Activity**

**Umbrella Activity**

**Umbrella Activity**

1. Feasibility Study
2. TEAM FORMATION
3. Project Specification PREPARATION

Business Requirement Documentation

ANALYSIS & DESIGN

CODE

UNIT TEST

DOCUMENT CONTROL

ASSESSMENT

TRAINING

INTEGRATION & SYSTEM TESTING

DELIVERY/INSTALLATION

ACCEPTANCE TEST

Requirements Gathering

SDLC is nothing but Software Development Life Cycle. It is a standard which is used by software industry to develop good software.

**Stages in SDLC:**

* Requirement Gathering
* Analysis
* Designing
* Coding
* Testing
* Maintenance

**Requirements Gathering stage:**

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.



These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are *not* included in the requirements document.

The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term requirements traceability.

The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

* Feasibility study is all about identification of problems in a project.
* No. of staff required to handle a project is represented as Team Formation, in this case only modules are individual tasks will be assigned to employees who are working for that project.
* Project Specifications are all about representing of various possible inputs submitting to the server and corresponding outputs along with reports maintained by administrator.

**Analysis Stage:**

The planning stage establishes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches.



The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included. The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage, and high level estimates of effort for the out stages.

**Designing Stage:**

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.

  
When the design document is finalized and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement. The outputs of the design stage are the design document, an updated RTM, and an updated project plan.

**Development (Coding) Stage:**

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, and data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software.



The RTM will be updated to show that each developed artifact is linked to a specific design element, and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration. The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software, an implementation map that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

**Integration & Test Stage:**

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user list are compiled into the Production Initiation Plan.



The outputs of the integration and test stage include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated project plan.

* **Installation & Acceptance Test:**

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loaded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer.

After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.



The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR enters the last of the actual labor data into the project schedule and locks the project as a permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.

**Maintenance:**

Outer rectangle represents maintenance of a project, Maintenance team will start with requirement study, understanding of documentation later employees will be assigned work and they will undergo training on that particular assigned category. For this life cycle there is no end, it will be continued so on like an umbrella (no ending point to umbrella sticks).

**3.4. Software Requirement Specification**

**3.4.1. Overall Description**

A Software Requirements Specification (SRS) – a [requirements specification](http://en.wikipedia.org/wiki/Requirements_specification) for a [software system](http://en.wikipedia.org/wiki/Software_system) is a complete description of the behavior of a system to be developed. It includes a set of [use cases](http://en.wikipedia.org/wiki/Use_case) that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. [Nonfunctional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements) are requirements which impose constraints on the design or implementation (such as [performance engineering](http://en.wikipedia.org/wiki/Performance_engineering) requirements, [quality](http://en.wikipedia.org/wiki/Quality_%28business%29) standards, or design constraints).

System requirements specification: A structured collection of information that embodies the requirements of a system. A [business analyst](http://en.wikipedia.org/wiki/Business_analyst), sometimes titled [system analyst](http://en.wikipedia.org/wiki/System_analyst), is responsible for analyzing the business needs of their clients and stakeholders to help identify business problems and propose solutions. Within the [systems development lifecycle](http://en.wikipedia.org/wiki/Systems_development_life_cycle) domain, the BA typically performs a liaison function between the business side of an enterprise and the information technology department or external service providers. Projects are subject to three sorts of requirements:

* [Business requirements](http://en.wikipedia.org/wiki/Business_requirements) describe in business terms what must be delivered or accomplished to provide value.
* Product requirements describe properties of a system or product (which could be one of several ways to accomplish a set of business requirements.)
* Process requirements describe activities performed by the developing organization. For instance, process requirements could specify .Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:
* **ECONOMIC FEASIBILITY**

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs. The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at NIC, There is nominal expenditure and economical feasibility for certain.

* **Operational Feasibility**

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization’s operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits. The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

* **TECHNICAL FEASIBILITY**

Earlier no system existed to cater to the needs of ‘Secure Infrastructure Implementation System’. The current system developed is technically feasible. It is a web based user interface for audit workflow at NIC-CSD. Thus it provides an easy access to .the users. The database’s purpose is to create, establish and maintain a workflow among various entities in order to facilitate all concerned users in their various capacities or roles. Permission to the users would be granted based on the roles specified. Therefore, it provides the technical guarantee of accuracy, reliability and security.

**3.4.2. External Interface Requirements**

**User Interface**

The user interface of this system is a user friendly python Graphical User Interface.

**Hardware Interfaces**

The interaction between the user and the console is achieved through python capabilities.

**Software Interfaces**

The required software is python.

**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

**3.SYSTEM DESIGN**

**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:**



**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:**



**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:**

****

**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:**

****

**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:**



**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:**

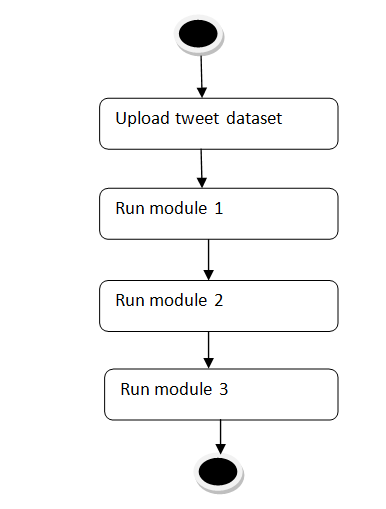


**Activity diagram:**

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:**

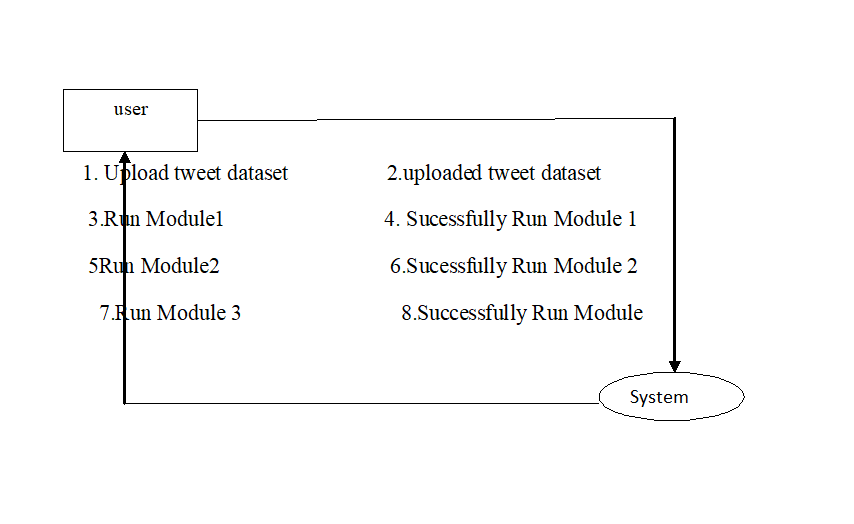
****

**Data Flow Diagram:**

[Data flow diagrams](http://www.edrawsoft.com/Data-Flow-Diagrams.php) 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.

**DataFlowDiagram**:



**5.IMPLEMETATION**

**5.1 Python**

Python is a general-purpose language. It has wide range of applications from Web development (like: Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D). The syntax of the language is clean and length of the code is relatively short. It's fun to work in Python because it allows you to think about the problem rather than focusing on the syntax.

**History of Python:**

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.

**Why Python was created?**

In late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group. He wanted to use an interpreted language like ABC (ABC has simple easy-to-understand syntax) that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to design of a new language which was later named Python.

**Why the name Python?**

No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

**Features of Python:**

**A simple language which is easier to learn**

Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#. Python makes programming fun and allows you to focus on the solution rather than syntax.

If you are a newbie, it's a great choice to start your journey with Python.

**Free and open-source**

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software’s written in it, you can even make changes to the Python's source code.

Python has a large community constantly improving it in each iteration.

**Portability**

You can move Python programs from one platform to another, and run it without any changes.

It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

**Extensible and Embeddable**

Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code.

This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

**A high-level, interpreted language**

Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on.

Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

**Large standard libraries to solve common tasks**

Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself. For example: Need to connect MySQL database on a Web server? You can use MySQLdb library using import MySQLdb .

Standard libraries in Python are well tested and used by hundreds of people. So you can be sure that it won't break your application.

**Object-oriented**

Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively.

With OOP, you are able to divide these complex problems into smaller sets by creating objects.

**Applications of Python:**

**1. Simple Elegant Syntax**

Programming in Python is fun. It's easier to understand and write Python code. Why? The syntax feels natural. Take this source code for an example:

a = 2

b = 3

sum = a + b

print(sum)

**2. Not overly strict**

You don't need to define the type of a variable in Python. Also, it's not necessary to add semicolon at the end of the statement.

Python enforces you to follow good practices (like proper indentation). These small things can make learning much easier for beginners.

**3. Expressiveness of the language**

Python allows you to write programs having greater functionality with fewer lines of code. Here's a link to the source code of Tic-tac-toe game with a graphical interface and a smart computer opponent in less than 500 lines of code. This is just an example. You will be amazed how much you can do with Python once you learn the basics.

**4. Great Community and Support**

Python has a large supporting community. There are numerous active forums online which can be handy if you are stuck.

**5.2 Sample 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

main = tkinter.Tk()

main.title("Detecting Malicious Twitter Bots Using Machine Learning") #designing main screen

main.geometry("1300x1200")

global filename

global dataset

words = ['bot','cannabis','tweetme','mishear','followme','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():

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)

logreg = LogisticRegression().fit(X\_train, y\_train)#logistic regression object

actual = y\_test

pred = logreg.predict(X\_test)

accuracy = accuracy\_score(actual, pred) \* 100

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']]

X = train\_attr

Y = np.asarray(train\_label)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)

logreg = LogisticRegression().fit(X\_train, y\_train) #logistic regression object

actual = y\_test

pred = logreg.predict(X\_test)

accuracy = accuracy\_score(actual, pred) \* 100

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()

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=450,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 2 (Recognize Malicious URLS using ML)", command=runModule3, bg='#ffb3fe')

module3Button.place(x=450,y=600)

module3Button.config(font=font1)

main.config(bg='SpringGreen2')

main.mainloop()

**6. 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.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 cannot do any 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 tweetbos using ML) | Verify the tweets are recognized or not using ML | If recognisation is not done | We cannot run  operation | We can Run the Operation | | High | High |
| 04 | Run module 3  (recognize tweet bots URL using ML) | Verify the tweets URLs are recognized or not using ML | If UMLs are not recognized | We cannot run  operation | We can Run the Operation | | High | High |

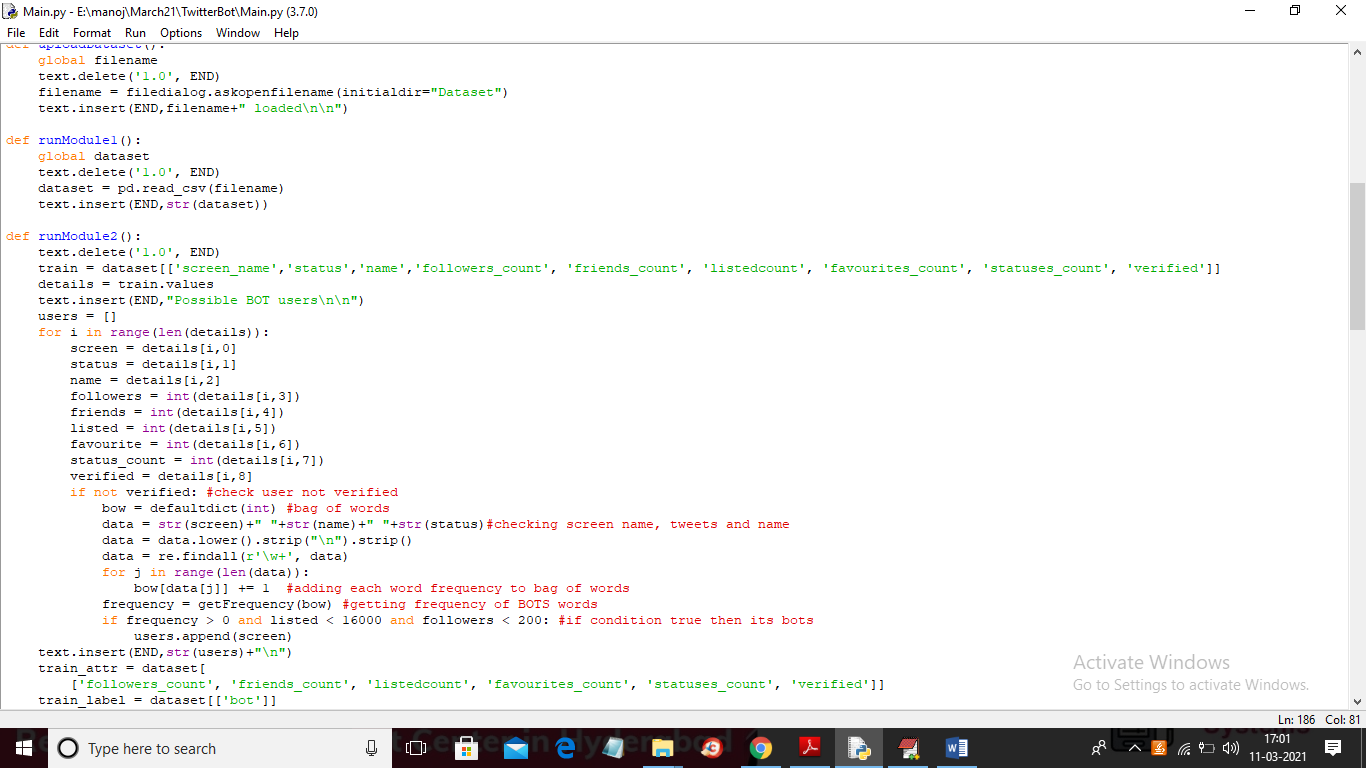
**7. SCREENSHOTS:**

In this paper author is using machine learning algorithm such as Logistic Regression to detect BOTS from tweets and to implement this paper author has given 3 modules

Checking all tweets for BOTS words and then finding its frequency

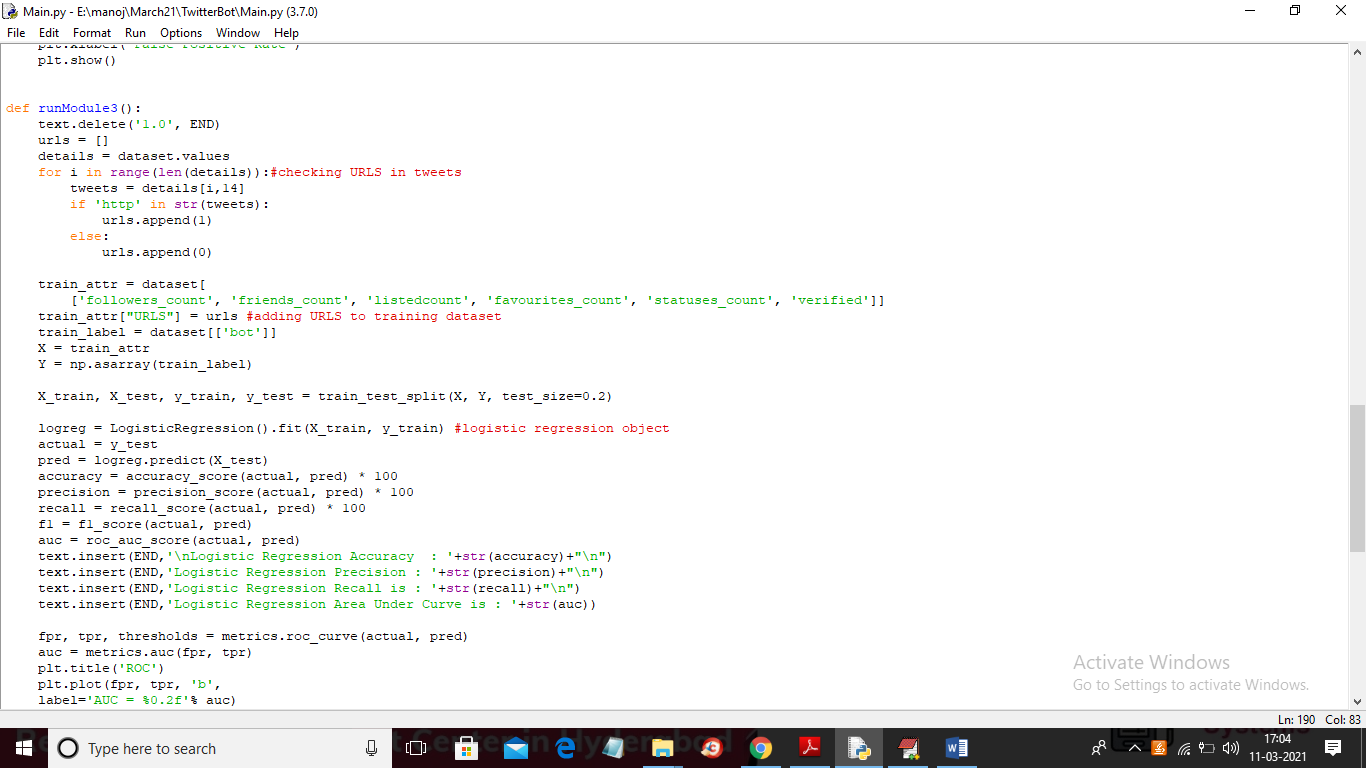
If account not verified and If listed count < 16000 and followers < 200 and retweet\_count> 10000 then account will be consider as BOT

By using above finding we will train Logistic regression and calculate bot prediction accuracy. Below screen shots showing code with comments about this methods

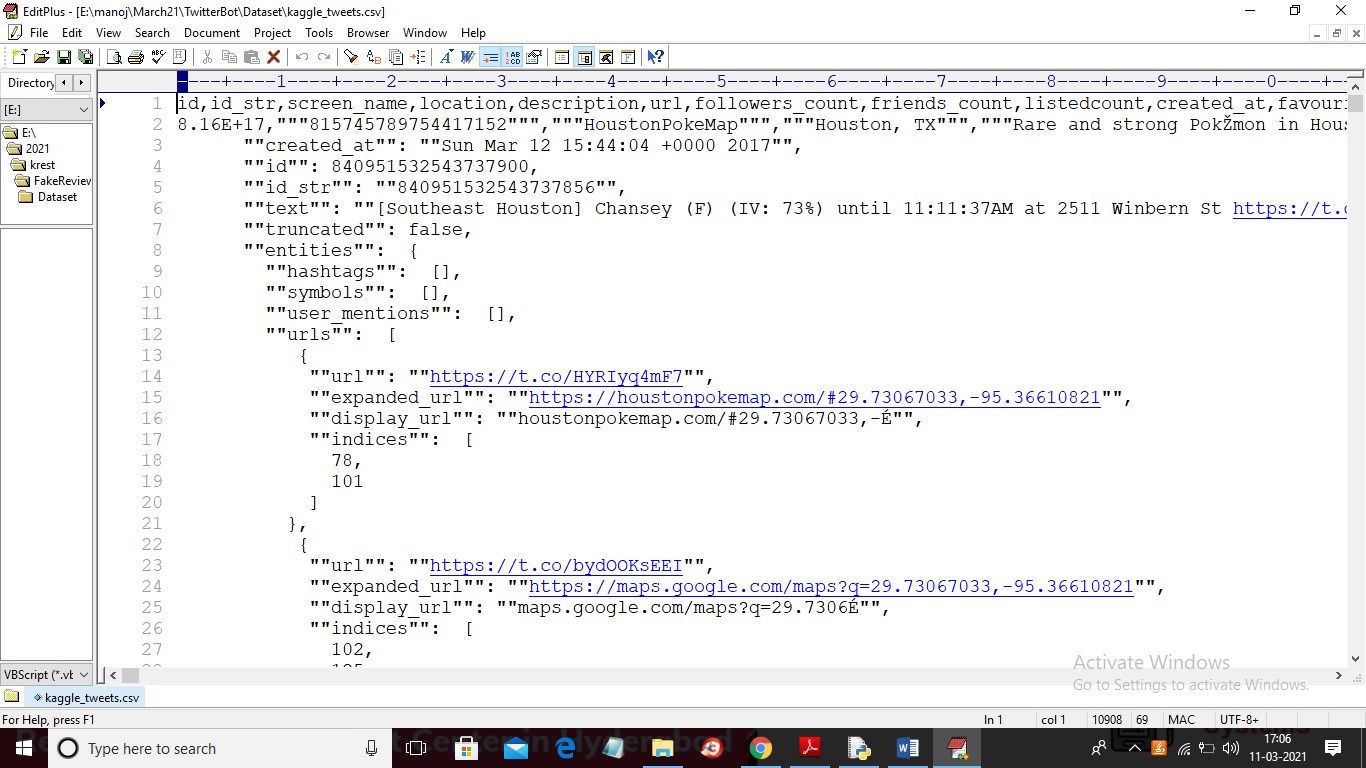


In above screen read red colour line comments to understand concept

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



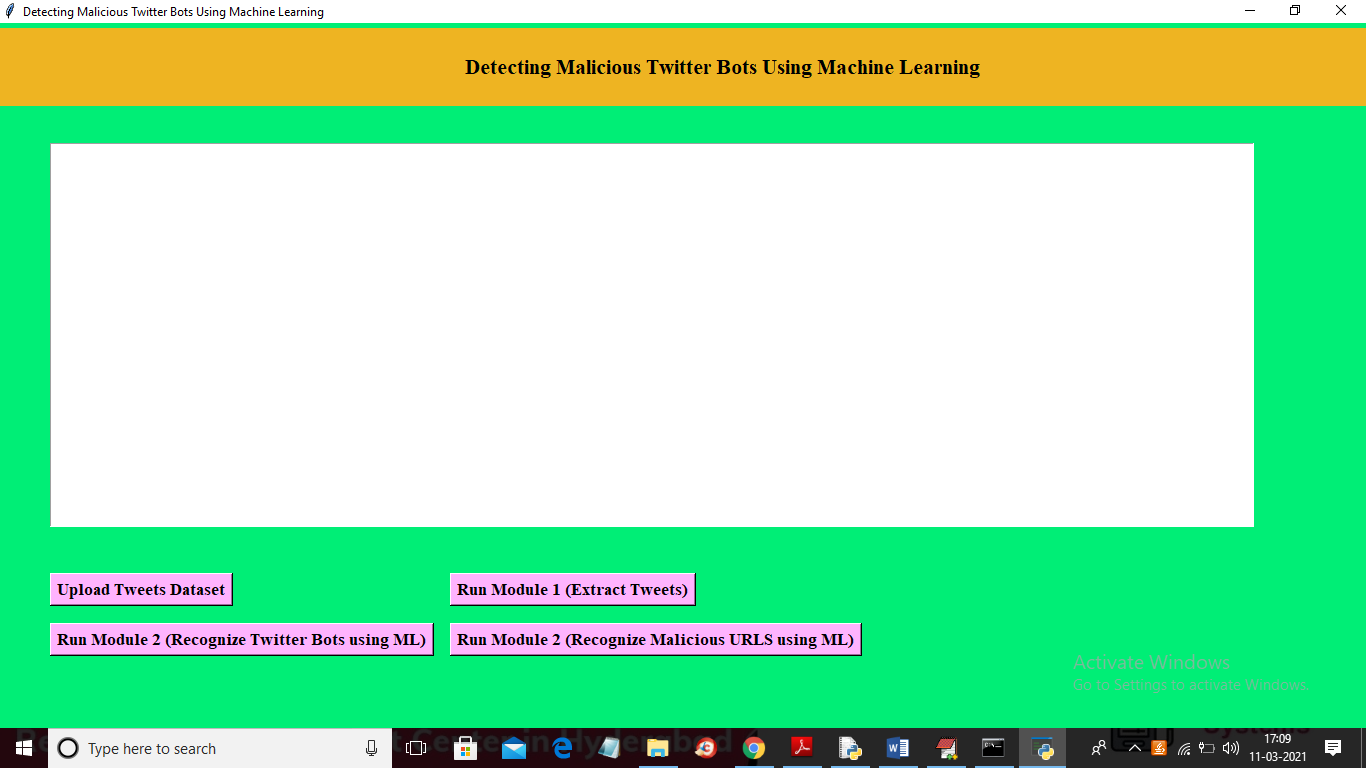
Dataset used in this project saved inside ‘Dataset’ folder and below screen shots showing all dataset details



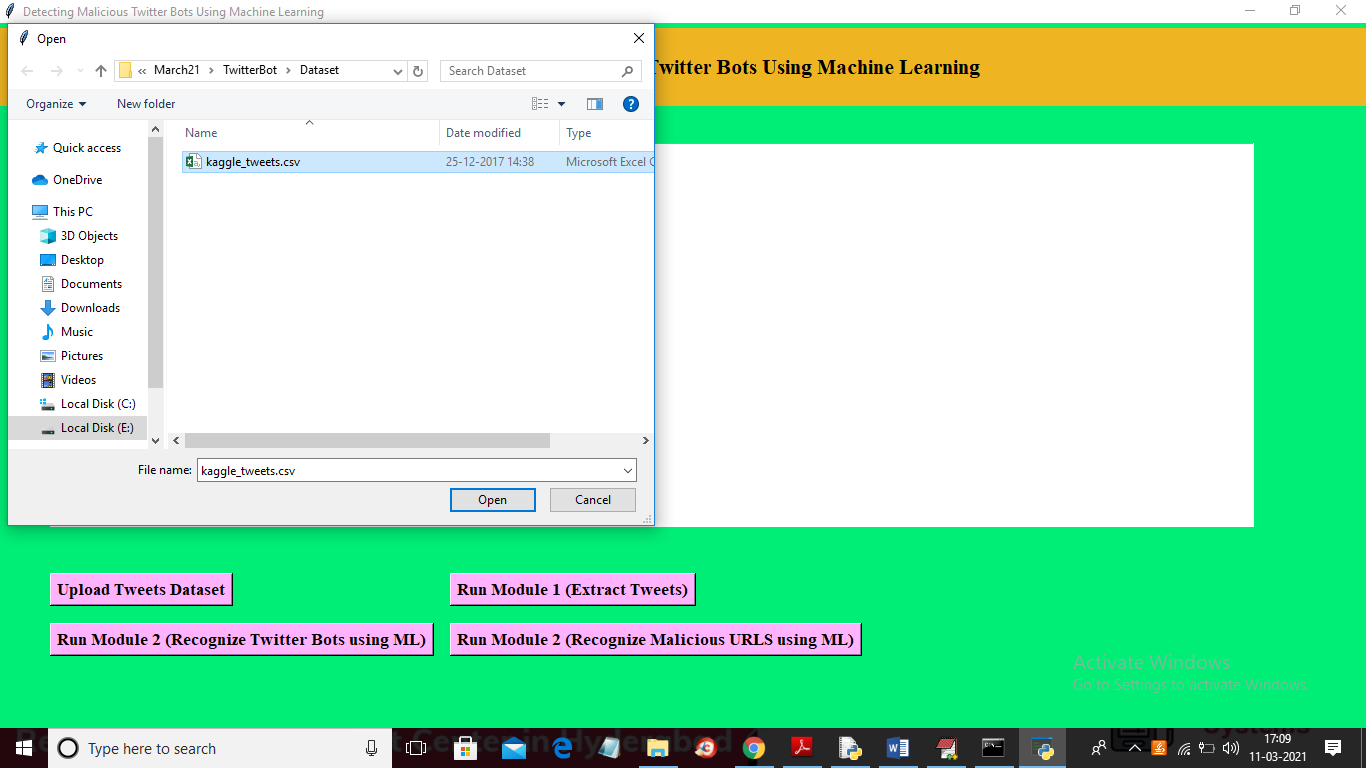
In above dataset screen in first line we can see all dataset column names and from second lines we can see dataset values.

SCREEN SHOTS

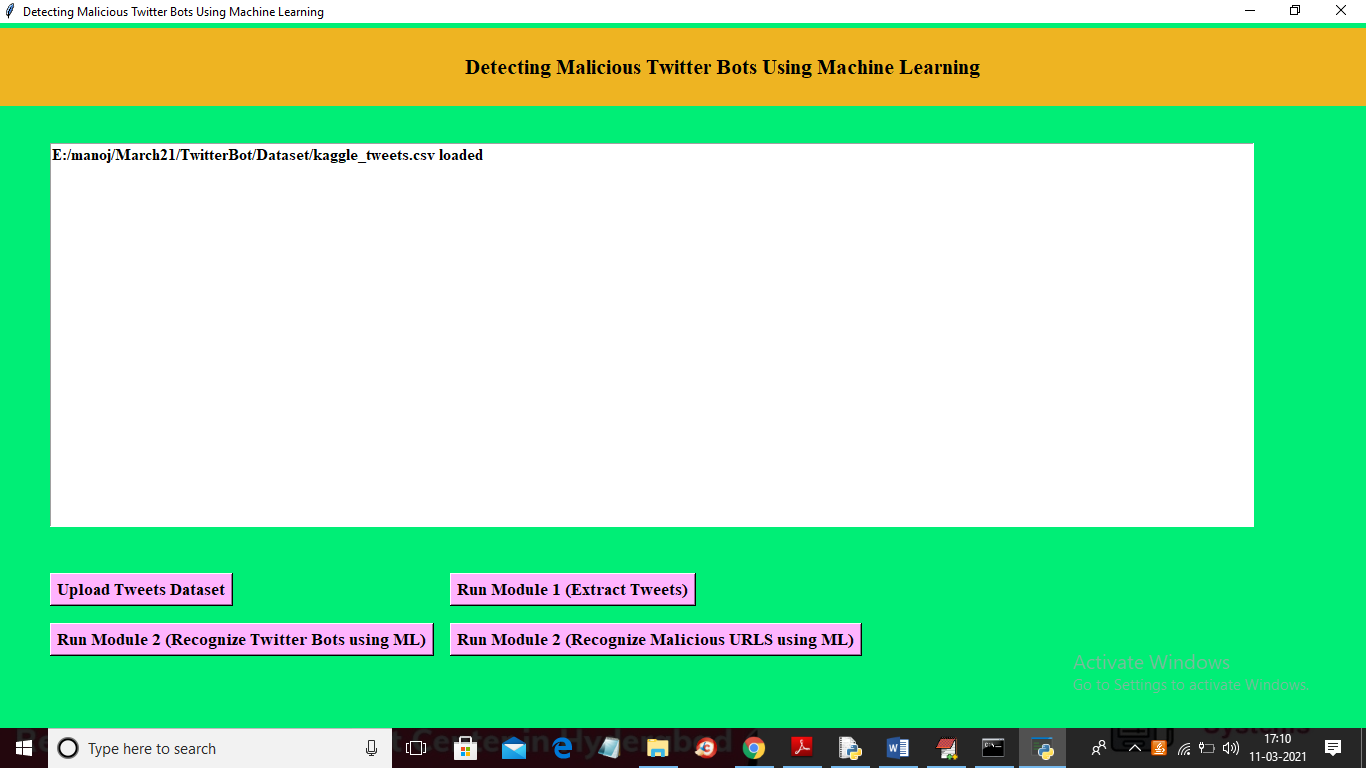
To run project double click on ‘run.bat’ file to get below screen



In above screen click on ‘Upload Tweets Dataset’ button and then upload dataset



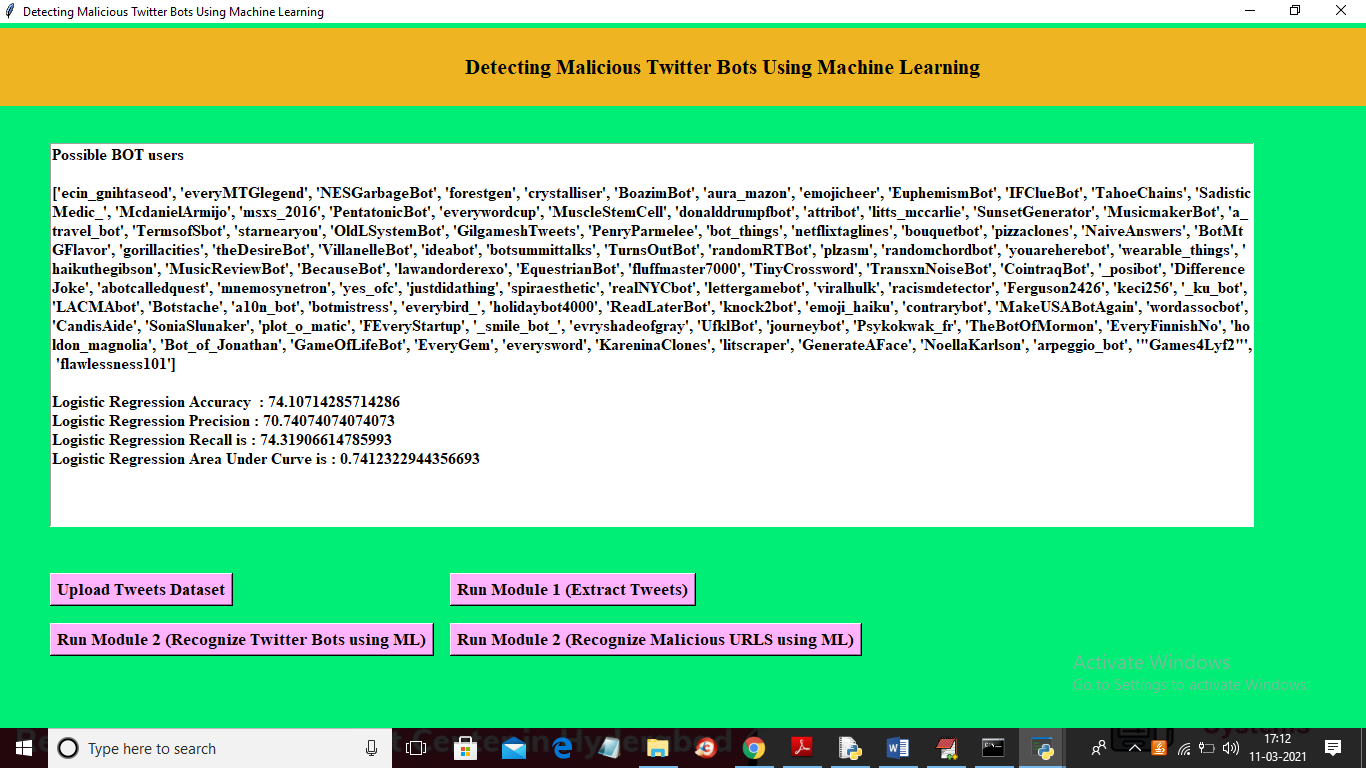
In above screen selecting and uploading ‘kaggle\_tweets.csv’ file and then click on ‘Open’ button to load dataset and to get below screen



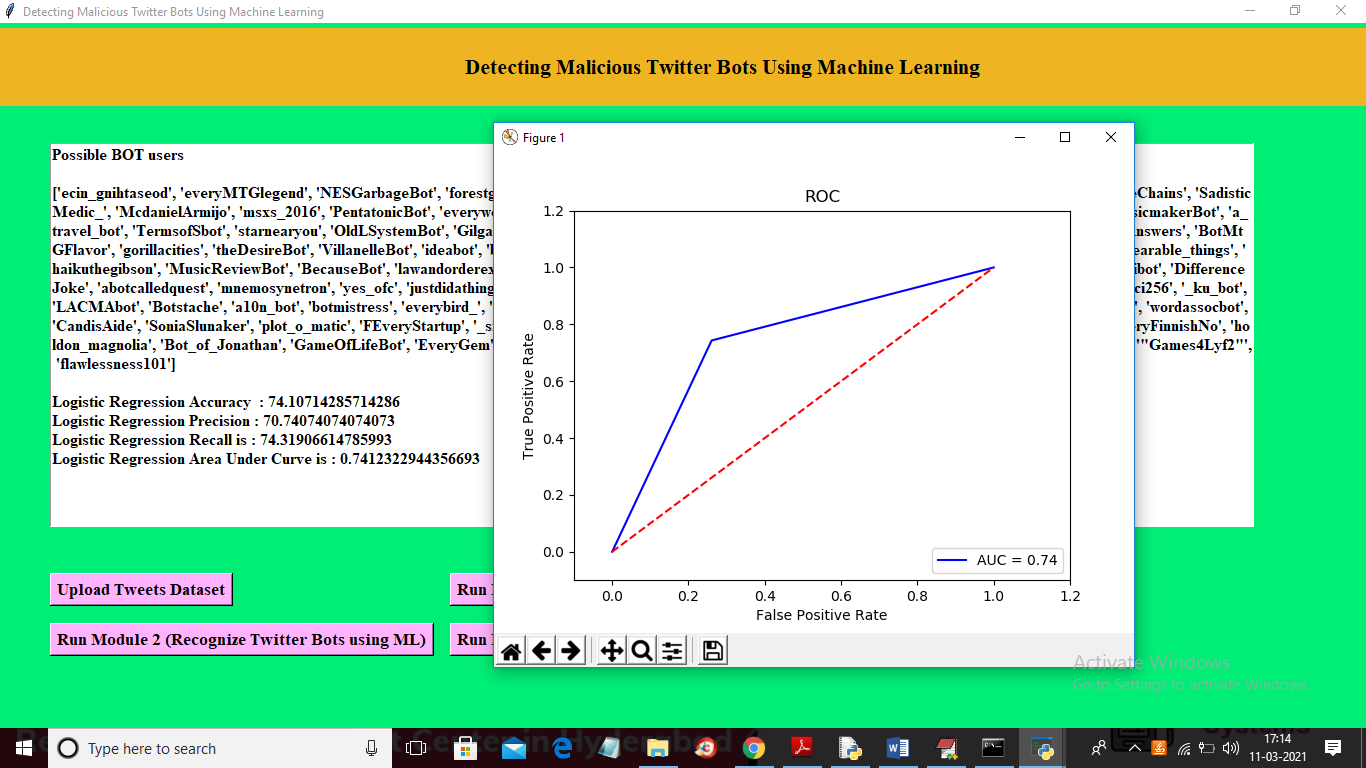
In above screen dataset loaded and now click on ‘Run Module 1 (Extract Tweets)’ button to read all tweets from dataset and to get below screen



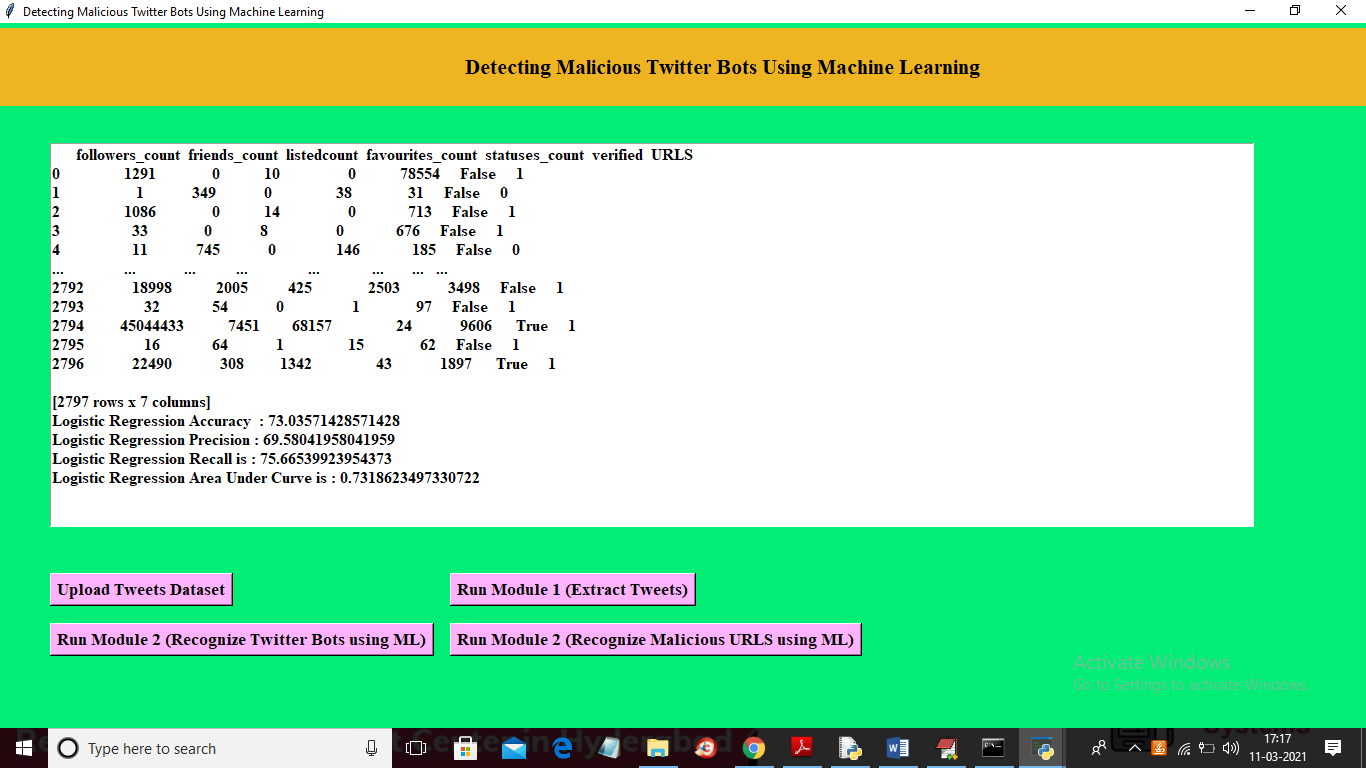
In above screen application read all tweets and displaying few tweets in above screen and now click on ‘Run Module 2 (Recognize Twitter Bots using ML)’ button to recognize BOTS user and then apply logistic regression ML to calculate BOT prediction accuracy



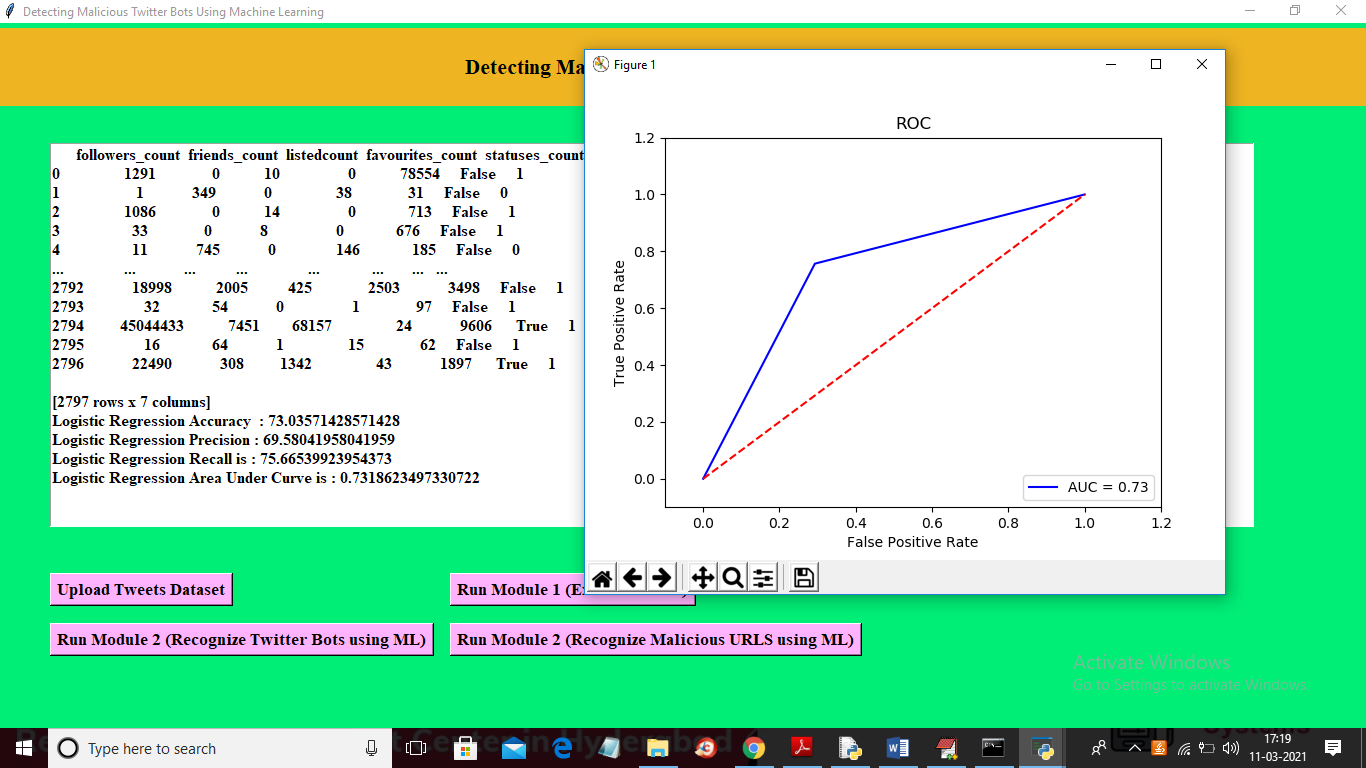
In above screen in square brackets we can see SCREEN NAME of all BOTS account and then we got ML accuracy of Logistic Regression is 74% and in below screen we can see ROC graph



In above graph x-axis represents False Positive Rate (wrong prediction) and y-axis represents True Positive Rate (Correct Prediction) and red line represents False Rate and blue line represents True prediction Rate and from above graph we can conclude that True rate is higher than false prediction rate. Now click on ‘Run Module 2 (Recognize Malicious URLS using ML)’ button to find malicious URLS and then calculate malicious URL prediction rate.



In above screen in last column we find URL as malicious or non-malicious and in above screen in last column 1 indicates Non-Malicious URL and 0 indicate Malicious URL and in above screen we can see URL prediction accuracy as 73% and below is URL prediction ROC graph



Above screen represents malicious URL ROC graph and blue line rate is higher than red colour false prediction rate

**8. CONCLUSION:**

We developed an algorithm in our research that identifies Twitter bots. Bag of words method for train data was the best model with an accuracy of 96.7 percent compared to Decision Tree, Naive Bayes and Random Forest 96.65 percent for test data. Thus, word algorithms were used to real-time data and the Twitter bots have been detected effectively

**9. REFERENCES:**

[1] Van Der Walt, Estée, and Jan Eloff. Using machine learning to detect fake identities: bots vs humans. IEEE Access 6 (2018): 6540-6549.

[2] Sever Nasim, Mehwish, Andrew Nguyen, Nick Lothian, Robert Cope, and Lewis Mitchell. Real-time detection of content polluters in partially observable Twitter networks. arXiv preprint arXiv:1804.01235 (2018).

[3] Khalil, Ashraf, Hassan Hajjdiab, and Nabeel Al-Qirim. Detecting Fake Followers in Twitter: A Machine Learning Approach. International Journal of Machine Learning and Computing 7, no.6(2017).

[4] Wetstone, Jessica and Sahil R. Nayyar. I Spot a Bot: Building a binary classifier to detect bots on Twitter. (2017).

[5] Karataş, Arzum, and SerapŞahin. A Review on Social Bot Detection Techniques and Research Directions. In Proc. Int. Security and Cryptology Conference Turkey, pp. 156-161. 2017.

[6] Chavoshi, Nikan, Hossein Hamooni, and Abdullah Mueen. Identifying correlated bots in twitter. In International Conference on Social Informatics, pp. 14- 21. Springer, Cham, 2016.

[7] Perdana, Rizal Setya, Tri HadiahMuliawati, and Reddy Alexandro. Bot spammer detection in Twitter using tweet similarity and time interval entropy. JurnalIlmuKomputer dan Informasi 8, no. 1 (2015): 19-25.

[8] Haustein, Stefanie, Timothy D. Bowman, Kim Holmberg, Andrew Tsou, Cassidy R. Sugimoto, and Vincent Larivière. Tweets as impact indicators: Examining the implications of automated bot accounts on T witter. Journal of the Association for Information Science and Technology 67, no. 1 (2016): 232-238.