

DISCERNMENT OF AUTONOMOUS PROFILES ON SOCIAL NETWORKING SERVICES (SNS)

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering
by

ANUSHKA SINGH (38110042)

ANHUSHREE M. (38110043)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAA

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600 119**

APRIL - 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Anushka Singh (38110042)**, **Anhushree M (38110043)** who carried out the project entitled as **Discernment of Autonomous Profiles on Social Networking Services (SNS)** under my supervision from November 2021 to April 2022.

Internal Guide

Mr. VIGNESH. R, M.E., (Ph.D.,)

Head of the Department

Dr. S. VIGNESHWARI, M.E., Ph.D.,

Dr. L. LAKSHMANAN, M.E., Ph.D.,

Submitted for Viva voce Examination held on_____

Internal Examiner

External Examiner

DECLARATION

I **Anushka Singh (38110042), Anhushree M (38110043)** hereby declare that the Project Report entitled “**Discernment of Autonomous Profiles on Social Networking Services (SNS)**” done by us under the guidance of **Mr. Vignesh. R, M.E., (Ph.D.,)** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 09.04.2022

SIGNATURE OF THE CANDIDATE: Anushka Singh

PLACE: Chennai

ACKNOWLEDGEMENT

I am pleased to acknowledge our sincere thanks to **Board of Management of Sathyabama Institute of Science and Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, **Dr. S. Vigneshwari, M.E., Ph.D.**, and **Dr. L. Lakshmanan, M.E., Ph.D., Heads** of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Mr. Vignesh. R, M.E., (Ph.D.,)** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of our project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

At present social network sites are part of the life for most of the people. Every day several people are creating their profiles on the social network platforms and they are interacting with others independent of the user's location and time. The social network sites not only providing advantages to the users and also provide security issues to the users as well their information. Spam Messages. Malicious comments, Cyber bullying activities are very much common in today's generation. All these challenging activities are carried out in social platforms with the use of fake profiles, this has imposed numerous challenges to our privacy policies. To analyze who are encouraging threats in social network we need to classify the social networks profiles of the users. In this project, we shall provide a new method that is highly proficient in revealing fake account profiles in social media. We are proposing a machine learning classification model that helps us classify users into genuine users and bot accounts. Therefore, this project report highlights the review of Fake Profile Detection on Social Networking Services by using Machine Learning.

TABLE OF CONTENT

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	V
	LIST OF FIGURES	IX
	LIST OF ABBREVIATION	X
1.	INTRODUCTION	1
	1.1 MOTIVATION	1
	1.2 MODEL IDLE	2
	1.3 OBJECTIVES	2
	1.4 OVERVIEW OF THE PROJECT	2
2.	LITERATURE SURVEY	4
	SCOPE AND REQUIREMENT ANALYSIS OF	7
3.	PRESENT INVESTIGATION	
	3.1 EXISTING SYSTEM	7
	3.2 PROPOSED SYSTEM	7
	3.3 TECHNOLOGY USED	7
	3.4 REQUIREMENT ANALYSIS	8
	3.4.1 HARDWARE.ENVIRONMENT	8
	3.4.2 SOFTWARE ENVIRONMENT	8
4.	METHODOLOGY AND ALGORITHMS USED	10
	4.1 INTRODUCTION TO MACHINE LEARNING	10
	4.2 TRAINING THE DATA	11
	4.2.1 SUPERVISED LEARNING	11

4.2.2 UNSUPERVISED LEARNING	11
4.3 METHODS IN SUPERVISED LEARNING	12
4.3.1 CLASSIFICATION	12
4.3.2 REGRESSION	13
4.4 APPROACHES IN CLASSIFICATION	13
4.4.1 DECISION TREE	14
4.4.2 XGBOOSTING ALGORITHM	15
4.5 SYSTEM ARCHITECTURE	16
4.6 IMPLEMENTATION OF ARCHITECTURE	19
4.6.1 DATASET & MODEL FEATURES	19
4.6.2 CORRELATIONS & DISTRIBUTION PLOTS	20
4.6.3 MODEL EVALUATION & SCORING	24
4.6.4 TRAIN THE MODEL	24
5. RESULTS AND DISCUSSION	25
5.1 RESULT	25
5.1.1 PRE-MODEL DATA VISUALISATION	25
5.1.2 MODEL INFORMATION VISUALISATION	29
5.1.3 POST-MODEL DATA VISUALIZATION	33
6. CONCLUSION AND FUTURE WORK	35
6.1 CONCLUSION	
6.2 FUTURE WORK	
REFERENCES	36
APPENDIX	39

A. SOURCE CODE	39
B. SCREENSHOTS	64
C. PUBLICATION WITH PLAGIARISM REPORT	66

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
4.1	CLASSIFICATION V/S REGRESSION	12
4.2	RANDOM FOREST SIMPLIFIED	15
4.3	SYSTEM ARCHITECTURE DIAGRAM	17
4.4	BLOCK DIAGRAM	18
4.5	WORKFLOW DIAGRAM	19
5.1	CORRELATIONS OF POTENTIAL FEATURES: ALL DATA	26
5.2	CORRELATIONS OF POTENTIAL FEATURES: BOTS ONLY	27
5.3	CORRELATIONS OF POTENTIAL FEATURES: HUMANS ONLY	27
5.4	DISTRIBUTION OF FOLLOWERS & FOLLOWING	28
5.5	DISTRIBUTION OF AGE & NETWORK SIZE	28
5.6	TWEET-FOLLOWER METRIC BY VERIFICATION STATUS	29
5.7	NETWORK SIZE BY VERIFICATION STATUS	29
5.8	CLASSIFICATION REPORT	30
5.9	CONFUSION METRICES	33
7.1	USER INTERFACE OF OUR PROJECT	64
7.2	PROBABILITY PREDICTION OF USER PROFILE	65

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
SNS	Social Networking Services
ML	Machine Learning
IDE	Integrated Development Environment
UI	User Interface
API	Application Programming Interface
NLP	Natural Language Processing
SVM	Support Vector Machine
ANN	Artificial Neural Network
NLTK	Natural Language Toolkit
ROC	Receiver Operating Characteristic curve
AUC	Area Under the Curve
SQL	Structured Query Language

CHAPTER 1

INTRODUCTION

Every day several people are creating their profiles on the social network platforms and they are interacting with others independent of the user's location and time. The social network sites not only providing advantages to the users and also provide security issues to the users as well their information. Spam Messages. Malicious comments, Cyber bullying activities are very much common in today's generation. All these challenging activities are carried out in social platforms with the use of fake profiles, this has imposed numerous challenges to our privacy policies. To analyze who are encouraging threats in social network, we need to classify the social networks profiles of the users.

1.1 MOTIVATION

At present social network sites are part of the life for most of the people. More than half of the entire population use several social media platforms like Twitter, Facebook, Instagram, Kakao chats etc. This has evolved to be a part of their life as well. Every day several people are creating their profiles on the social network platforms and they are interacting with others independent of the user's location and time. There arises a thought of why social media platforms have become important? But the main reason is connections, online friends, this online world has let many of us to have comfortable, socializing relationship that is purely online. Through such interfaces one can easily chat via text, audio, video, posting comments etc. The social network sites not only providing advantages to the users and also provide security issues and threats from outside communities to the users as well their information. To avoid all these problems a lot of research and experiments were carried out. Most of it were completely carried out in the field of Supervised machine classifiers and Unsupervised machine classifiers.

1.2 MODEL IDE

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The IDE was installed in Anaconda, an open-source distribution for the languages Python and R used to perform Data Science and Machine Learning. The IDE's UI in which the model was developed is given in Fig 1.1.

1.3 OBJECTIVES

The main aim of this project is to ensure earlier, faster and a more reliable detection for identifying bot or human accounts. The most fundamental challenge for this application is the security issues and threats from outside communities to the users as well their information. This project provides many features like

- The objective is to provide a probability percentage of the given user account.
- To increase the security and privacy measures.
- Helps to detect accounts through different ML algorithms.

1.4 OVERVIEW OF THE PROJECT

Chapter 1, deals with the introduction to the project where the existing system is being discussed.

Chapter 2, a detailed description of the literature survey of the papers which are referred during the course of the project was summarized.

Chapter 3, gives a brief explanation on the aim and scope of the project. Here the proposed system has been compared with the existing system. The software and hardware requirements are provided along the system design, architecture overall project.

Chapter 4, the Results and Discussion along with the screenshots of each module has been depicted.

Chapter 5, deals with the summary and conclusion of the project. It also includes the future scope of the project.

CHAPTER 2

LITERATURE SURVEY

P. Srinivas Rao, Dr. Jayadev Gyani and Dr. G. Narsimha approach Machine Learning and NLP method to increase the accuracy rate of fake accounts detection. Through support vector and Naïve bays, many problems are removed on social media, the problem is like privacy issues, cyber bullies, and troll, etc. Most of the time seen these types of malicious activities are done by false accounts present on social sites. This research takes the Facebook dataset for the recognition of false accounts. Here Machine Learning Algorithm along with NLP pre-processing method applies for analyzing the dataset and for classification of the profile use support vector machine classifier that helps to detect the profile is fake or genuine.

As investigated by **Rubin V.** (2016), satirical cues in language can be used to detect certain language patterns to identify intentionally created misleading headlines. Instead of using content matching with original source they have followed NLP together with machine learning to detect language patterns specific to satire and irony. Fake news is also a form of news that intentionally altered or contain false information to deliberately manipulate readers. Usually these stories may cause confusion, will push political propaganda by influencing views of the people. When evaluating content online, there are multiple types of fake news that one should be aware of.

According to **Hwang**, modern bots can even be involved in more complex types of interactions, such as having conversations with others, commenting on their posts, and answering their questions. Some bots specifically aim to achieve greater influence. They can search social networks for popular and influential people and follow them or draw their attention by sending them questions, hoping to be noticed.

Chavoshi also applied unsupervised techniques in their bot detection system called DeBot that can detect bots based on their synchronicity because this system constantly collects data from Twitter.

Li have been stated that supervised technique algorithms are impractical because we need a large amount of dataset to train. However, since social media like Twitter have a very large number of users, it is impossible to label an abundance of training data. Meanwhile, unsupervised Authorized licensed use limited to techniques are not robust compared with supervised techniques and have a high false-positive rate. So, to overcome the difficulty, they used semi-supervised learning, which takes advantage of both supervised and unsupervised learning.

According to **Kudugunta and Ferrara**, using a supervised technique for detecting malicious accounts shows promising results in their experiment.

Nasteski also concluded that supervised technique is more successful than the unsupervised technique because the ability of the labeled training data gives us clearer criteria for optimization models. Moreover, this method contains a large number of algorithms which are increasing every time by data scientists.

Yeh-Cheng Chen, Shyhtsun Felix Wu has proposed that some previous work has studied the structure and composition of fake account that distribute spam messages, however the fast improvement of fake account creation made these previous works outdated or getting ineffective. They then target on new model of fake account that may not only automatically post or comment, but also spam with advertisement or spread false information. In this research, they proposed an innovative method to detect fake account in OSNs. This method based on user activity pattern through Facebook activities, and we leverage machine learning to predict if an account is controlled by fake user. They also provide an analysis of those account activity from the prediction result. In their paper, they have first collected account and then train the models by using Random Forest along with

Adaptive Boosting with decision stump. A given set of manually checked and labelled fake accounts are used to create our evaluation metrics. Supervised machine learning algorithm uses a decision tree classifier having the highest accuracy that is 98% with a 2% error rate. By using AdaBoost, it gave 97.55% accuracy. This research may bring up to a whole new level in fake accounts detection.

In addition to that, **Latifah Alhaura and Indra Budi** has proposed in which they divided the classification level into three: account level classification, tweet level classification, and combination of account and tweet level classification. For getting the classification results, they applied some known machine learning algorithms such as Random Forest, Decision Tree, AdaBoost Classifier, Neural Network and Logistic Regression to each classification level. The results show that Random Forest achieved high classification accuracy (AUC > 80%) in each classification level using our proposed feature set.

According to **Indra Budi**, in this research, they build a classification model using only account tweets. We also build further classification distinguishing fake followers and spambots from genuine accounts. In this research, maximum accuracy has been reached at 95.55% in malicious vs genuine account detection using tf-idf features and XGBoost algorithm and 95.2% in all three types of accounts using Word2Vec features and XGBoost algorithm.

CHAPTER 3

SCOPE AND REQUIREMENT ANALYSIS OF PRESENT INVESTIGATION

In this chapter, the proposed system has been compared with the existing system. The software and hardware requirements are provided.

3.1 EXISTING SYSTEM

In existing system, the social network sites not only providing advantages to the users and also provide security issues and threats from outside communities to the users as well their information. To avoid all these problems a lot of research and experiments were carried out. Most of it were completely carried out in the field of Supervised machine classifiers and Unsupervised machine classifiers. The Supervised Machine Learning Algorithm uses classifiers like Naive Bayes, Decision tree, SVM, ANNs, and NLP. With the help of these the detection of Autonomous profiles can be carried out easily.

3.2 PROPOSED SYSTEM

The project would include a full-fledged Deep Learning model that would be trained on predicting if a user profile on a social media is a real human or fake (bot). We would use the XGBoosting Classification Algorithm for classifying each profile, with the dataset being sourced online using Kaggle. This model, after training, can be tested by supplying tweets from a bot and tweets from a real human and checking the results.

3.3 TECHNOLOGY USED

To implement the project, Jupyter Notebook has been used as the software development environment, some technologies have been used such as Machine

Learning, JavaScript, HTML, CSS, AJAX. Supervised machine learning classification model using Python like the sklearn library, and XGBoost has been used and then built a simple app using Flask.

3.4 REQUIREMENT ANALYSIS

Requirement analysis determines the requirements of a new system. This project analyses on product and resource requirement, which is required for this successful system. The product requirement includes input and output requirements it gives the wants in term of input to produce the required output. The resource requirements give in brief about the software and hardware that are needed to achieve the required functionality.

3.4.1 Hardware Environment

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the systems do and not how it should be implemented.

Hard Disk	:	80GB and Above
RAM	:	8GB and Above
Processor	:	P IV and Above (CPU with 1.5GHz)
Active internet connection		

3.4.2 Software Environment

The software requirements are the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost,

planning team activities, performing tasks and tracking the team's and tracking the team's progress throughout the development activity.

- Platform : Windows 7 and above
- Front End : HTML, CSS, Bootstrap.
- Back End : Flask / Django
- Language : Python 3.8
- Software : Jupyter Notebook / Google Notebook
- Modules : Pandas, Numpy, Scikit-Learn, Matplotlib Web
- Environment : Flask
- Cloud Computing : Heroku

CHAPTER 4

METHODOLOGY AND ALGORITHMS USED

In this chapter, architecture, workflow diagram with implementation and algorithms used has been discussed in detail.

4.1 INTRODUCTION TO MACHINE LEARNING

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) with data, without being explicitly programmed.

The name machine learning was coined in 1959 by Arthur Samuel. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or unfeasible; example applications include email filtering, detection of network intruders, and computer vision.

Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter sub-field focuses more on exploratory data analysis and is known as unsupervised learning.

Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data.

4.2 TRAINING THE DATA

There are basically two widely-used types of training that can be done to create a model:

- i. Supervised Learning
- ii. Un-supervised Learning

4.2.1 Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

4.2.2 Unsupervised Learning

Unsupervised machine learning is the machine learning task of inferring a function that describes the structure of "unlabeled" data (i.e., data that has not been classified or categorized). Since the examples given to the learning algorithm are unlabeled, there is no straightforward way to evaluate the accuracy of the structure that is

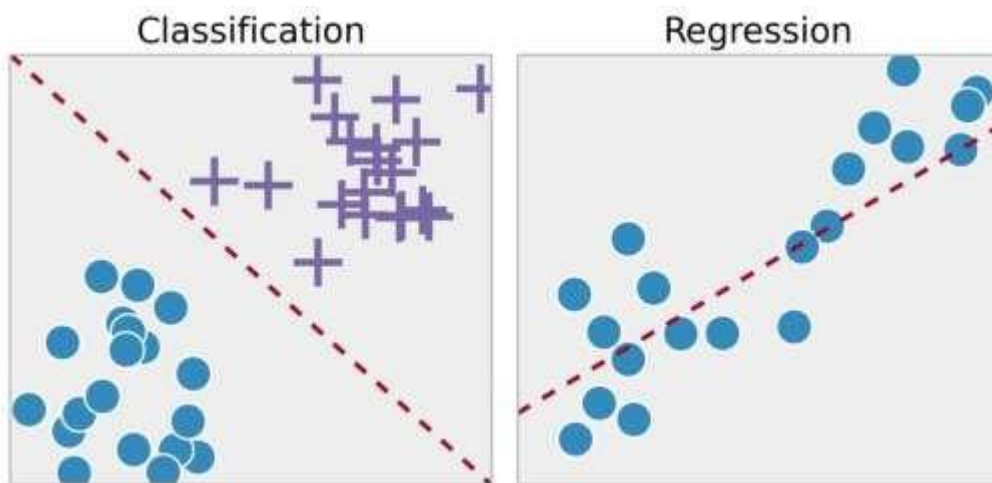
produced by the algorithm—one feature that distinguishes unsupervised learning from supervised learning and reinforcement learning.

The type of training used in this model is ***SUPERVISED LEARNING***.

4.3 METHODS IN SUPERVISED LEARNING

Supervised Learning mainly consists of two methods,

- Classification



- Regression

Fig 4.1 Classification vs Regression

4.3.1 Classification

In machine learning, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Examples are assigning a given email to the "spam" or "non-spam" class, and assigning a diagnosis to a given patient based on observed

characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.

4.3.2 Regression

Regression analysis estimates the conditional expectation of the dependent variable given the independent variables – that is, the average value of the dependent variable when the independent variables are fixed. Less commonly, the focus is on a quantile, or other location parameter of the conditional distribution of the dependent variable given the independent variables. In all cases, a function of the independent variables called the regression function is to be estimated.

Regression analysis is widely used for prediction and forecasting. It is also used to understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships. In restricted circumstances, regression analysis can be used to infer causal relationships between the independent and dependent variables. However, this can lead to illusions or false relationships, so caution is advisable; for example, correlation does not prove causation.

The type used in this model is **CLASSIFICATION** and so, more focus will be given on it.

4.4 APPROACHES IN CLASSIFICATION

There are different approaches when it comes to the Classification method, they are:

1. Linear classifiers

- a) Fisher's linear discriminant
 - b) Logistic regression
 - c) Naive Bayes classifier
 - d) Perceptron
- 2. Support vector machines
 - a) Least squares support vector machines
- 3. Quadratic classifiers
- 4. Kernel estimation
 - a) k-nearest neighbor
- 5. Boosting (XGBoosting algorithm)
- 6. Decision trees
 - a) Random forests
- 7. Neural networks
- 8. Learning vector quantization

The most common approaches used are **decision trees** and **XGBoosting algorithm**.

4.4.1 Decision Tree

Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data (but the resulting classification

tree can be an input for decision making).

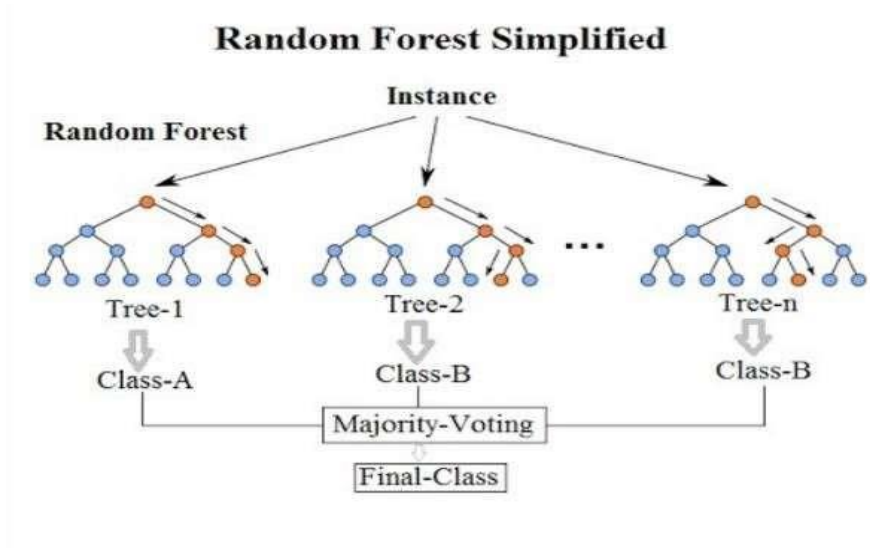


Fig 4.2 Random Forest Simplified

The most common and versatile application of decision trees come in the Random Forests approach/ algorithm. **Random forests** or **random decision forests** are an ensemble learning method for classification, that operates by constructing a multitude of decision trees at training time and outputting the class that is the most commonly appearing of the classes of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

4.4.2 XGBoosting Algorithm

XgBoost stands for Extreme Gradient Boosting, which was proposed by the researchers at the University of Washington. It is a library written in C++ which optimizes the training for Gradient Boosting.

XGBoost is an implementation of Gradient Boosted decision trees. XGBoost models majorly dominate in many Kaggle Competitions. In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the

decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems. It can be installed as a standalone library and XGBoost model can be developed using the scikit-learn API.

The approach used in this model is the ***XGBOOSTING ALGORITHM*** which has already been discussed.

4.5 SYSTEM ARCHITECTURE

First clean the data, getting it ready for training and testing, with Numpy, Pandas. NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

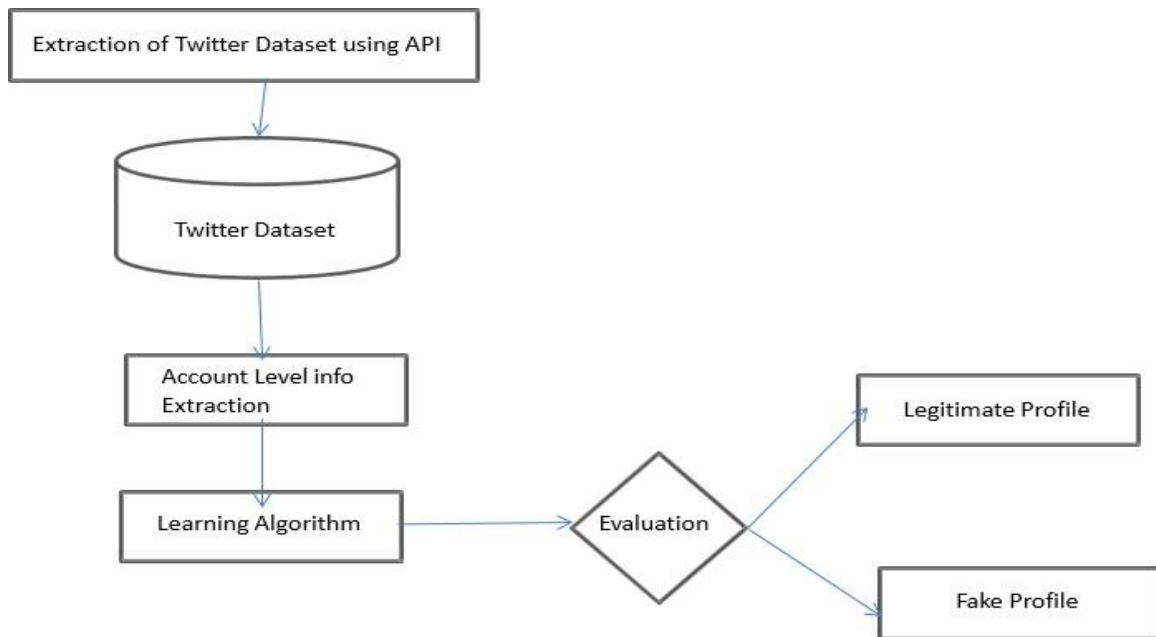


Fig.4.3 System architecture to check the probability for account to be fake

We visualize the data using Matplotlib, Seaborn. Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

We tokenize the tweets using NLTK for getting it ready for analysis. The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

We use WordCloud to visualize the most used words of tweets and removing it in case it would be polarizing.

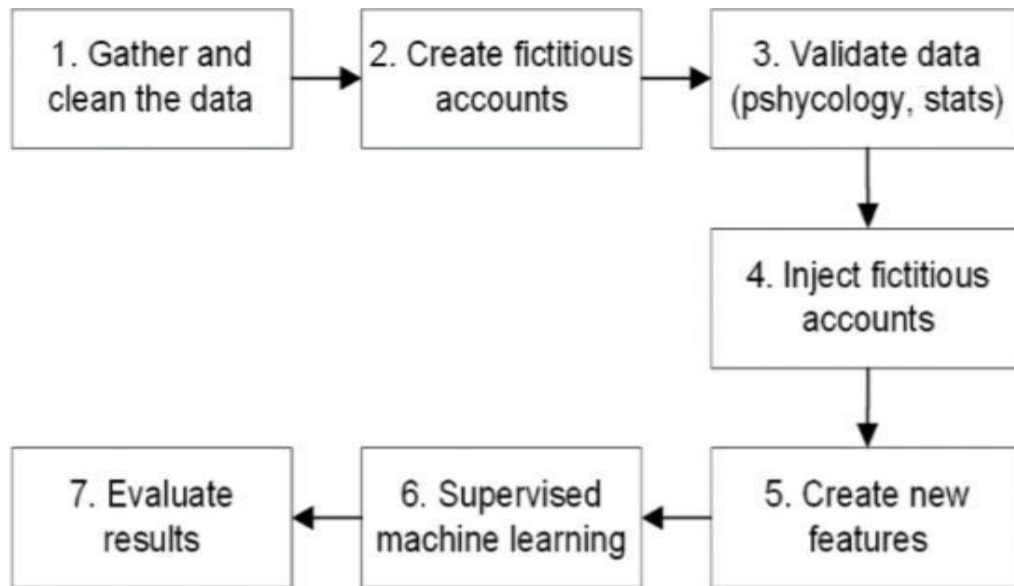


Fig.4.4 Block diagram of the proposed architecture

We use WordToVec to find similarities between tweets. Word2vec is a technique for natural language processing published in 2013. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence.

We train a Random Forest Classifier in order to classify between bots and humans. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.

We run the test dataset through this classifier to check the accuracy of the model. Here is a simple flow diagram to represent the above process.

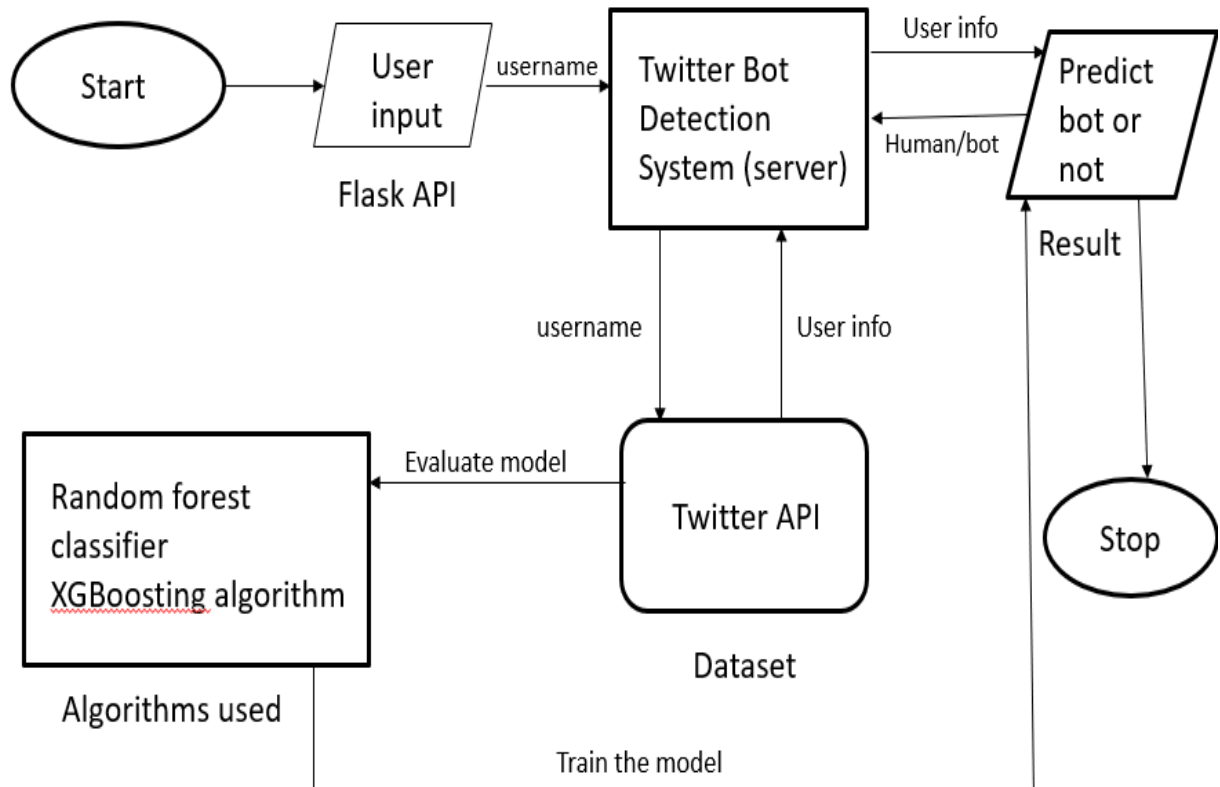


Fig.4.5 Workflow diagram to represent the above process

4.6 IMPLEMENTATION OF ARCHITECTURE

We have covered the modules that are set as the main aspect for detection of fake, bot profiles, The following ones are very much required to carry on the model.

1. Dataset & Model Features
2. Correlations and Distribution plots
3. Model Scoring & Evaluation
4. Train the model

4.6.1 Dataset & Model Features

The dataset is comprised of approximately 37,000 Twitter users, labeled bot or human, with account-level information like:

1. number of favorites/likes
2. number of tweets
3. number of followers
4. number of friends (accounts their following)
5. whether or not the profile is still in default mode and more.

The target label is account type: bot or human. any authorized.

4.6.2 Correlations and Distribution plots

Once the transformations and interactions are set up, we will go through the correlation heatmaps of the different features like potential features for all data, for human and for bots as well. For that purpose, we need to import and install all packages, libraries and modules required in the project.

Required libraries and modules:

Numpy:

NumPy is a library for Python, adding support for large, multidimensional arrays and matrices, along with a large collection of highlevel mathematical functions to operate on these arrays. NumPy is an open-source software and has many contributors.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as

most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality.

Written in: Python and C.

Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open-source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

The two primary data structures of pandas, Series (1- dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R's dataframe provides and much more. pandas are built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Written in: Python, Cython and C.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells,

the Jupyter notebook, web application servers, and four graphical user interface toolkits.

It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt or GTK+.

matplotlib.pyplot provides a MATLAB-like plotting framework.

pylab combines pyplot with numpy into a single namespace. This is convenient for interactive work, but for programming it is recommended that the namespaces be kept separate.

Written in: Python.

Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of our plots mean, rather than on the details of how to draw them.

Written in: Python.

PostgreSQL

PostgreSQL is a powerful, open-source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. The origins of PostgreSQL date back to 1986 as part of the POSTGRES project at the University of California at Berkeley and has more than 30 years of active development on the core platform.

PostgreSQL has earned a strong reputation for its proven architecture, reliability, data integrity, robust feature set, extensibility, and the dedication of the open-source community behind the software to consistently deliver performant and innovative solutions. PostgreSQL runs on all major operating systems, has been ACID-compliant since 2001, and has powerful add-ons such as the popular PostGIS geospatial database extender.

Written in: Python, Perl, Tcl, PostgreSQL, C.

Scikit-learn

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines(svm), random forest, gradient boosting, kmeans and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

It has been built on top of NumPy, SciPy and matplotlib.

Written in: Python, Cython, C and C++.

XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Written in: Python, C++.

Tweepy

Tweepy is a Python library for accessing the Twitter API. It is great for simple automation and creating twitter bots.

Written in: Python

Now, we can set up some distribution plots to understand how they behave in a different way. When exploring the dataset, we can say that bot and human accounts behave differently — for example, humans' tweet more frequently than their robot counterparts. Their networks also look differently as well — humans have bigger networks, both in terms of accounts they follow and that follow them.

4.6.3 Model Scoring & Evaluation

For the evaluation and scoring of the model, we have imported and tried a number of classification models from the sklearn library, like - Logistic Regression, K-Nearest Neighbors, Naive Bayes, Decision Tree, Random Forest, and

XGBoost. We have chosen features and model parameters that would optimize for a balance between precision and recall and a high ROC AUC (area under the curve) score. Essentially, we wanted the model to accurately label bots as such, but not by simply labelling everything as a bot.

4.6.4 Train the models

Consider class-weight balancing (data is ~70/30 split human/bot). Refine feature selection and tune candidate model parameters. We can create confusion matrices of each model i.e., Random Forest and XGBoost for a single train/test split. Now, we will ensemble Random Forest and XGBoost to create a super model by using Voting Classifier model. Pick best model i.e., XGBoost algorithm and perform full train and test. Train best model on full data set and pickle to be used in Flask app.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 RESULT

The result of the classification model prepared using Python may be divided into three parts in itself:

1. Pre-model Data Visualization
2. Model Information Visualization
3. Post-model Data Visualization

5.1.1 Pre-model Data Visualization

The pre-model data visualization section consists of all the data visualizations or example visualizations made with the pure data set to understand the features and the spread of data in the data set.

Data set Name – twitter human bots account

Dimensions – 644 kb

Number of user accounts – approx. 37,000+

Attributes – number of tweets

number of likes/favourites

number of followers

number of followings, etc.

Visualizations of Data

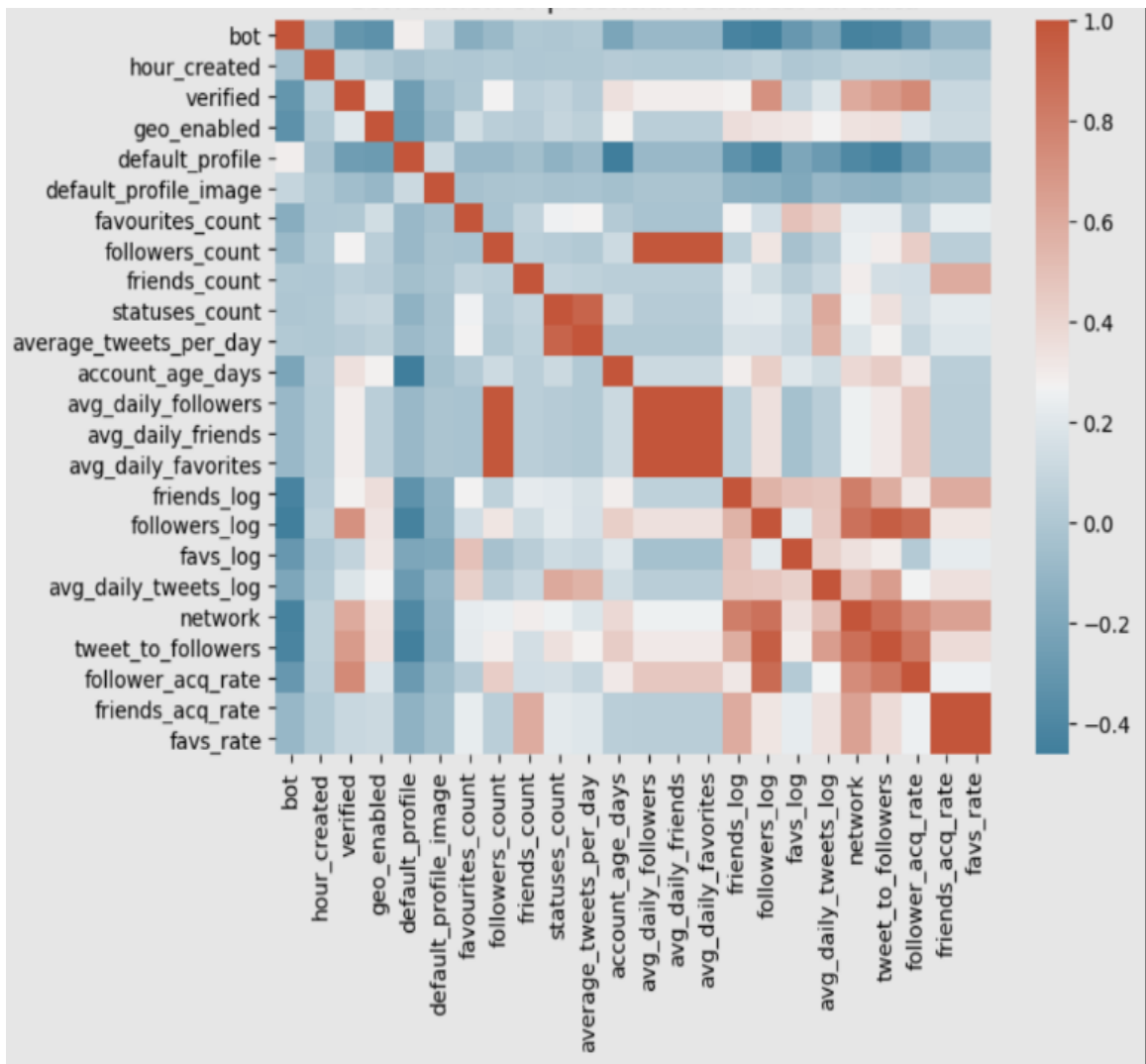


Fig. 5.1 Correlation of potential features: all data

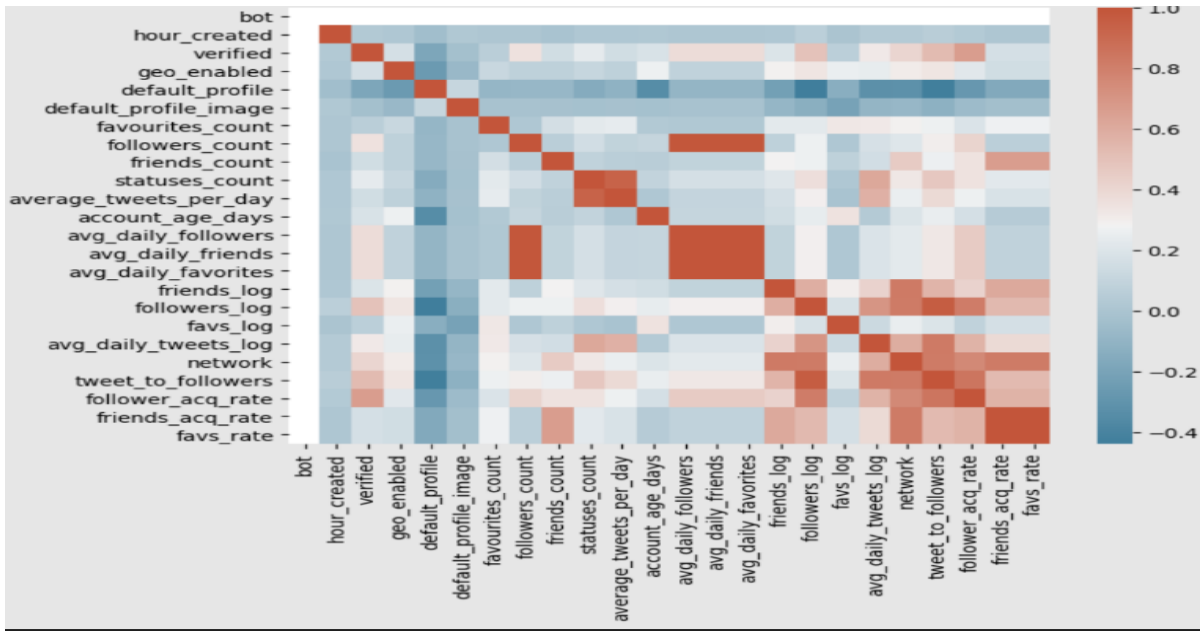


Fig. 5.2 Correlation of potential features: Bots only

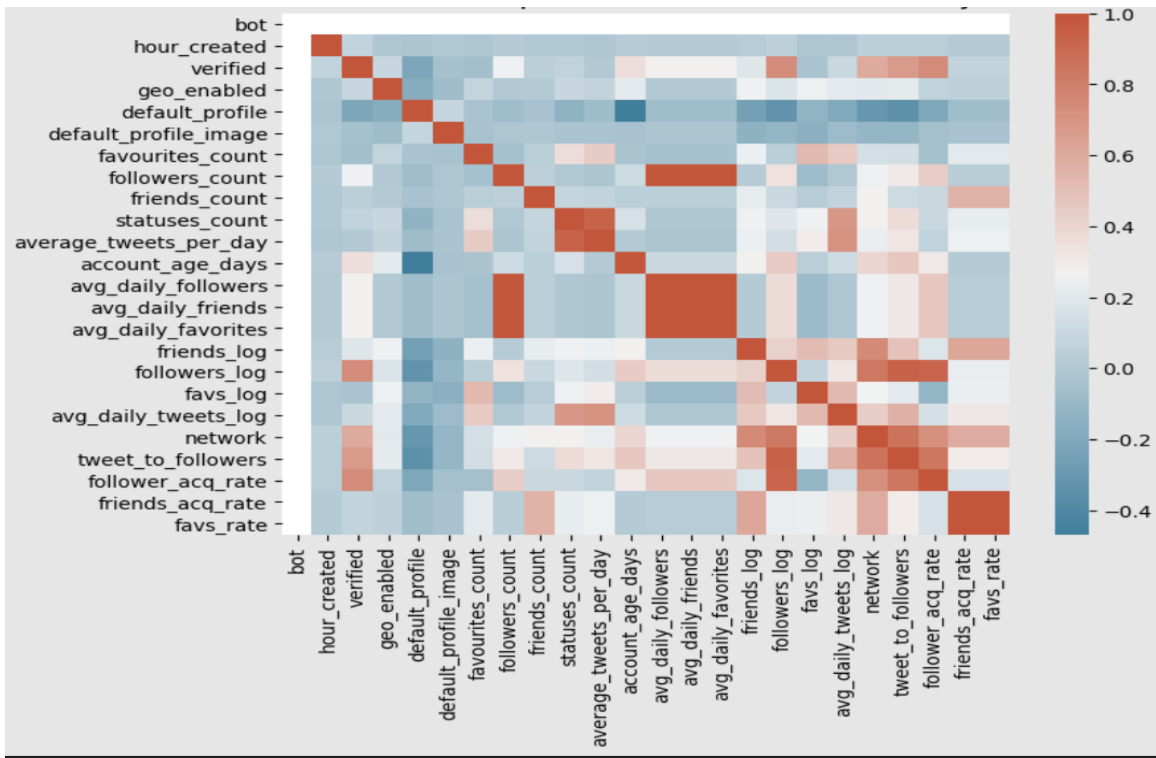


Fig. 5.3 Correlation of potential features: Humans only

Distribution plots to better understand how some of the features differ between bot and human accounts.

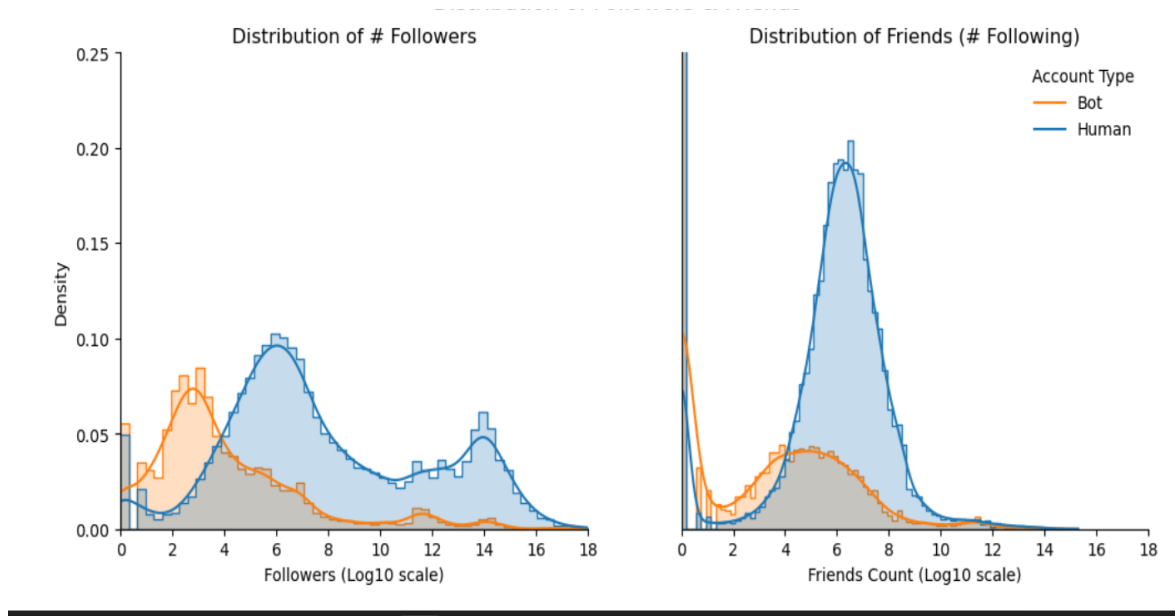


Fig 5.4 Distribution of Followers and Following

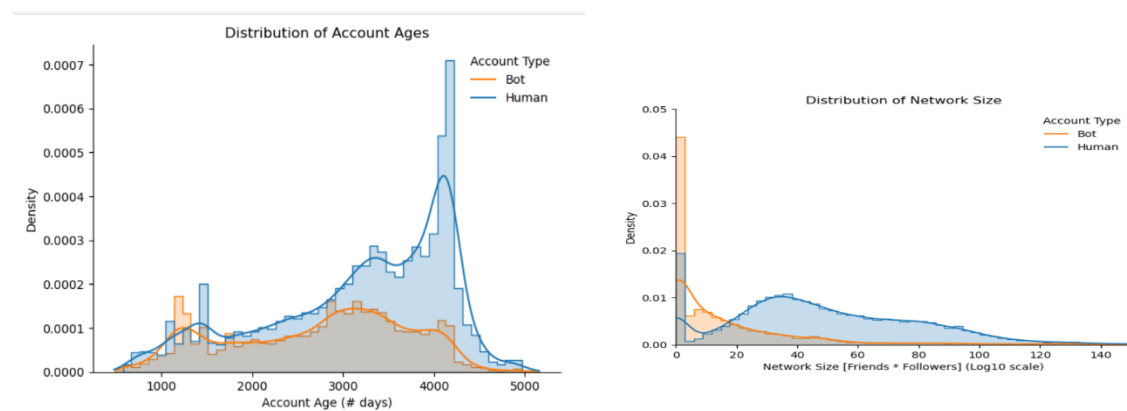


Fig.5.5 Distribution of account age and network size

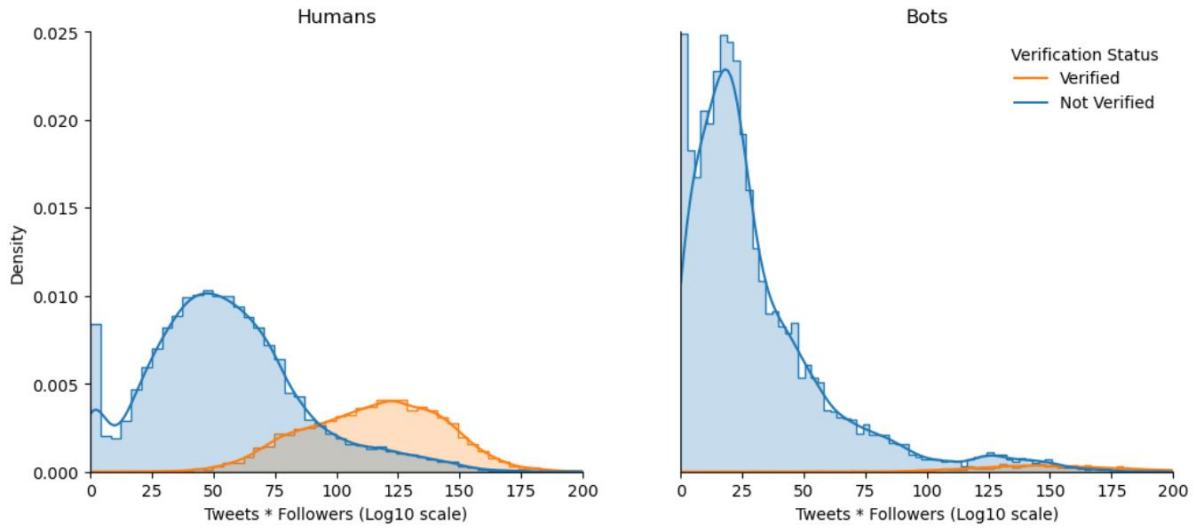


Fig.5.6 Tweets-followers metric by verification status

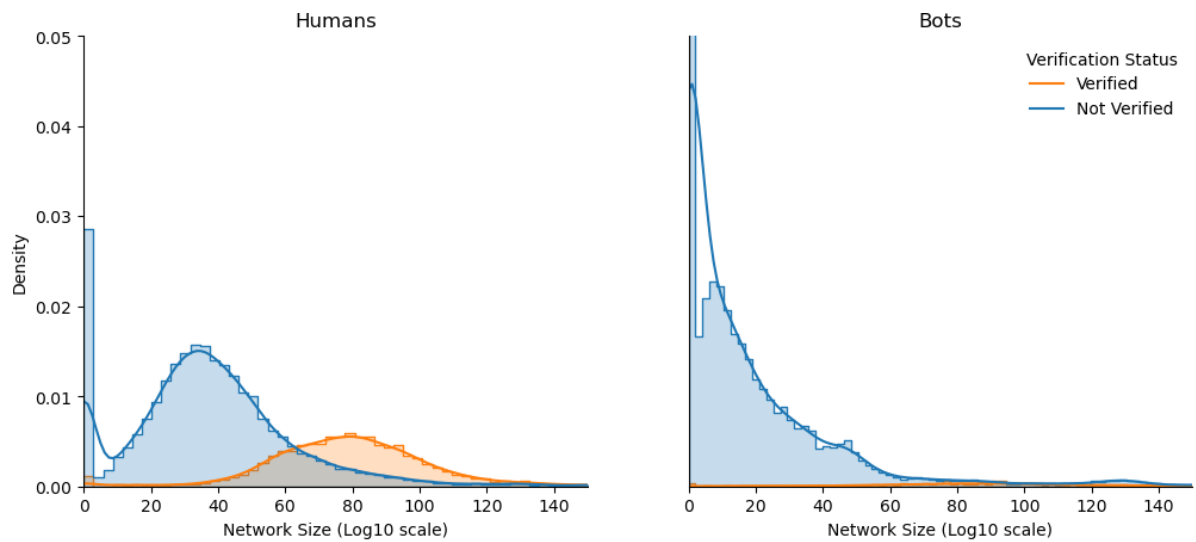


Fig.5.7 Network size by verification status

5.1.2 Model Information Visualization

Model has been evaluated by using different algorithms like KNeighbors Classifier, Logistic Regression, Naïve Bayes (Gaussian, Bernoulli, Multinomial), Decision Tree, Random Forest, XGBoost algorithms.

For this purpose, we need formula to be calculated.

```

Model: KNeighborsClassifier(n=1
-----
Accuracy: 0.83767 +- 0.004042
Precision: 0.80131 +- 0.006912
Recall: 0.68362 +- 0.013989
F1 Score: 0.73768 +- 0.006909
ROC AUC: 0.89569 +- 0.003649

Model: LogisticRegression()
-----
Accuracy: 0.80314 +- 0.002075
Precision: 0.71114 +- 0.006977
Recall: 0.69157 +- 0.010010
F1 Score: 0.70113 +- 0.003798
ROC AUC: 0.85181 +- 0.002741

Model: GaussianNB()
-----
Accuracy: 0.69503 +- 0.059328
Precision: 0.55385 +- 0.090251
Recall: 0.82332 +- 0.138664
F1 Score: 0.64431 +- 0.011958
ROC AUC: 0.82068 +- 0.002617

Model: BernoulliNB()
-----
Accuracy: 0.74124 +- 0.003394
Precision: 0.69436 +- 0.006968
Recall: 0.40249 +- 0.010015
F1 Score: 0.50949 +- 0.007802
ROC AUC: 0.80215 +- 0.000312

Model: MultinomialNB()
-----
Accuracy: 0.46963 +- 0.003039
Precision: 0.38474 +- 0.002703
Recall: 0.98151 +- 0.003564
F1 Score: 0.55279 +- 0.002419
ROC AUC: 0.59444 +- 0.000558

Model: DecisionTreeClassifier()
-----
Accuracy: 0.80951 +- 0.002601
Precision: 0.71052 +- 0.002434
Recall: 0.72499 +- 0.005602
F1 Score: 0.71768 +- 0.003801
ROC AUC: 0.78842 +- 0.003250

Model: RandomForestClassifier()
-----
Accuracy: 0.86835 +- 0.002722
Precision: 0.84612 +- 0.007291
Recall: 0.74069 +- 0.010617
F1 Score: 0.78981 +- 0.004186
ROC AUC: 0.92495 +- 0.003060

Model: XGBClassifier()
-----
Accuracy: 0.86808 +- 0.004606
Precision: 0.83845 +- 0.012897
Recall: 0.74970 +- 0.006979
F1 Score: 0.79152 +- 0.006060
ROC AUC: 0.92547 +- 0.002796

```

Fig.5.8 Classification Report

The above figure represents all the necessary factors taken to determine whether a model is good or bad. They are technically known as **Model Evaluation Metrics**.

- TP = True Positive = Model identifies it as a bot, and it is a bot in real life
- FP = False Positive = Model identifies it as a bot, and it is NOT a bot in real life
- FN = False Negative = Model identifies it as NOT a bot, and it IS a bot in real life
- TP + FP = The number of accounts the model identifies as a bot
- TP + FN = The number of accounts that are bots in real life

Precision

Precision is defined as the fraction of the examples which are actually positive among all the examples which we predicted positive. It is also defined as the number of true positives divided by the number of true positives plus the number of false positives.

$$\begin{aligned}\text{Precision} &= \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \\ &= \frac{\text{True Positive}}{\text{Total Predicted Positive}}\end{aligned}$$

Recall

Recall is defined as number of accounts that Model identifies as a bot, and it is a bot in real life divided by the number of accounts that are bots in real life.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$= \frac{\text{True Positive}}{\text{Total Actual Positive}}$$

The precise definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives. True positives are data point classified as positive by the model that actually are positive (meaning they are correct), and false negatives are data points the model identifies as negative that actually are positive (incorrect).

The metric our intuition tells us we should maximize is known in statistics as recall, or the ability of a model to find all the relevant cases within a dataset.

F1 Score

Traditionally, F1 Score is defined to be the harmonic mean of precision and recall. It is also called the F Score or the F Measure. Put another way, the F1 score conveys the balance between the precision and the recall.

It is considered to be a better measure than Precision and Recall individually as the

$$F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

trade-off between the two is hard to achieve.

ROC-AUC

ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

RandomForest and **XGBoost** seem most promising in terms of balance between Precision and Recall metrics and high ROC AUC scores.

5.1.3 Post-model Data Visualization

Confusion Matrices: - Let's take a look at the confusion matrices of each model for a single train/test split.

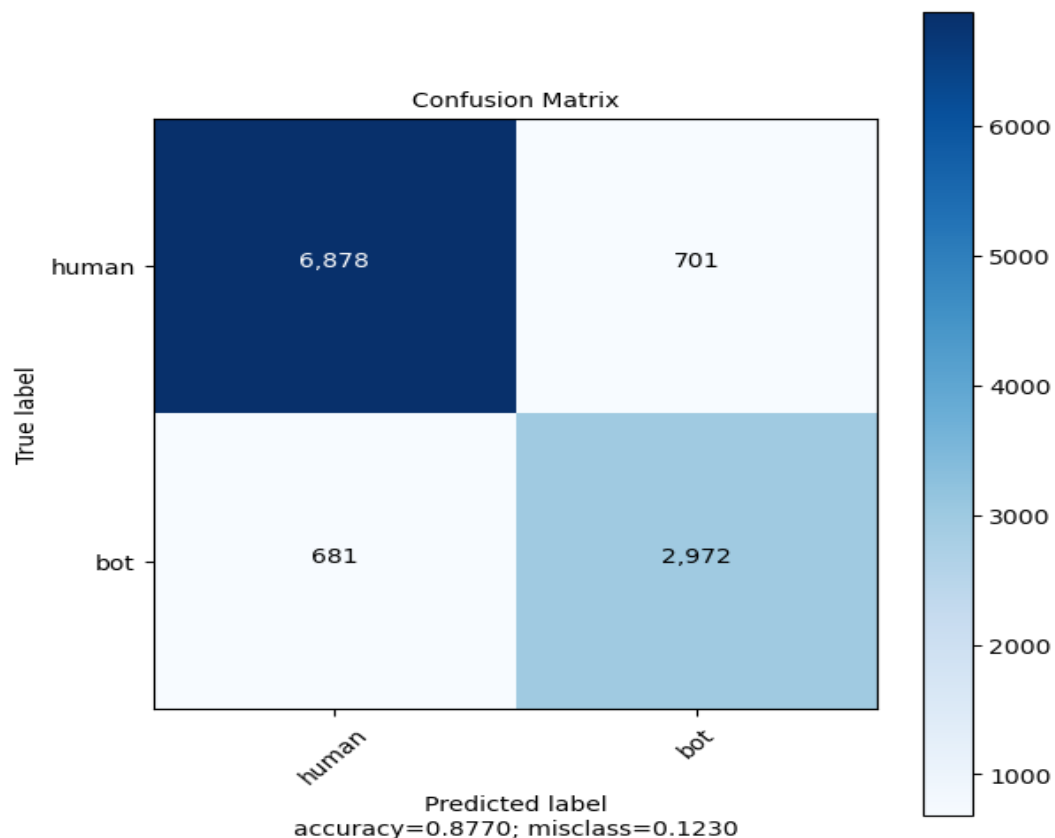


Fig.5.9 Confusion matrices for XGBoosting Algorithm

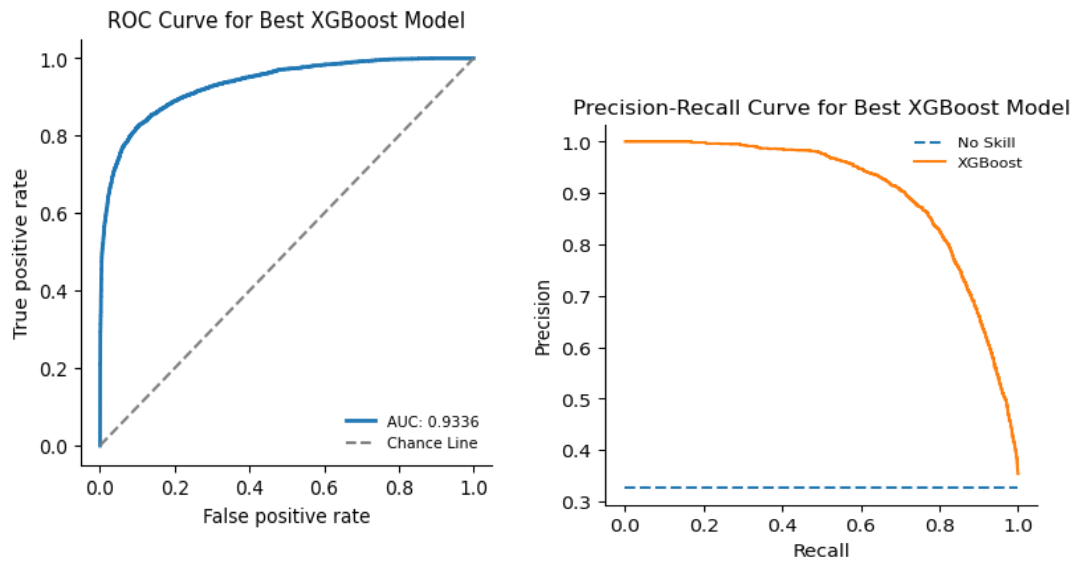


Fig.5.10 Classification report for XGBoost Model

Final result for XGBoost Model has been given

Metric	CV Score	Test Score
-----+-----+-----		
Accuracy:	0.8687	0.8770
Precision:	0.8035	0.8136
Recall:	0.8031	0.8091
F1 Score:	0.8033	0.8144
ROC AUC:	0.9310	0.9336

Accuracy: 87.7%, Precision: 81.3%, Recall: 80.9% and ROC AUC: 93.4%

Also, we have given user interface by using flask API to check the probability of a given user. This interface will also tell whether it's a bot account, human account or else likely to be a bot or human.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

According to our project, we have concluded an innovative and effective method to detect fake profile by leveraging machine learning algorithms. We can get rid of the spam that bots create on social media networks by running each tweet through this bot filter, in case they are not authorized to use automation to create content. This will further help get rid of artificial echo chambers echoing fake news amplified by automatically creating new posts on social media.

6.2 FUTURE WORK

The future works of the research may include some improvements in this research. For instance, future work may include research on some other domain languages other than English such as German, Korean or any other languages.

It can also include other social accounts like Facebook or Instagram under the same web API framework by integrating it. Future works may also use different kinds of features that don't rely on tweet-based features that undermine some of the spambots and fake followers' account data which never or rarely tweets or uses pf non-English language used in this research.

REFERENCES

- [1] Nambouri Sravya, Chavana Sai praneetha, S. Saraswathi," Identify the Human or Bots Twitter Data using Machine Learning Algorithms", International Research Journal of Engineering and Technology(IRJET), Volume: 06 Issue: 03 | Mar 2019 www.irjet.net, e-ISSN: 2395-0056, p- ISSN: 2395-0072.
- [2] M. Smruthi, N. Harini," A Hybrid Scheme for Detecting Fake Accounts in Facebook", International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-7, Issue-5S3, February 2019.
- [3] O. Varol, E. Ferrara, C. A. Davis, F. Menczer, A. Flammini, —Online Human-Bot Interactions: Detection, Estimation, and CharacterizationII, Proceedings of the Eleventh International AAAI Conference on Web and Social Media(ICWSM2017).
- [4] V. D. WALT and J. ELOFF, —Using Machine Learning to Detect Fake Identities: Bots vs HumansII, 2169-3536, 2018 IEEE, VOLUME 6, 2018.
- [5] S. Cresci, R. D. Pietro, M. Petrocchi, A. Spognardi, M. Tesconi, —The paradigm-shift of social spambotsII, Web Science 2017 ACM, Perth, Australia, ISBN 123-4567-24-567/08/06.
- [6] Kietzmann, J.H., Hermkens, K., McCarthy, I.P., Silvestre, B.S., 2011. Social media? Get serious! Understanding the functional building blocks of social media. Bus. Horiz., SPECIAL ISSUE: SOCIAL MEDIA 54, 241– 251. doi:10.1016/j.bushor.2011.01.005.
- [7] Kontaxis, G., Polakis, I., Ioannidis, S., Markatos, E.P., 2011. Detecting social network profile cloning, in: 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops). Presented at the 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 295–300. doi:10.1109/PERCOMW.2011.5766886.

- [8] Wang, J., Yu, L.C., Lai, K.R. and Zhang, X., 2016. Community-based weighted graph model for valence-arousal prediction of affective words. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(11), pp.1957- 1968.
- [9] De Choudhury, M., Counts, S., & Gamon, M. (2012, May). Not all moods are created equal! exploring human emotional states in social media. In *Sixth international AAAI conference on weblogs and social media* [28] Russell, J. A. (1980). A circumplex model of affect. *Journal of personality and social psychology*, 39(6), 1161. [29] Gao, H., Tang, J., Hu, X., & Liu, H. (2015, January). Content-Aware Point of Interest Recommendation on Location-Based Social Networks. In *AAAI* (pp. 1721-1727).
- [10] Tripathy, A., Agrawal, A., & Rath, S. K. (2016). Classification of sentiment reviews using n-gram machine learning approach. *Expert Systems with Applications*, 57, 117-126. [31] Bird, S., & Loper, E. (2004, July). NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions* (p. 31). Association for Computational Linguistics.
- [11] Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3), 264-323.
- [12] Y. Roth, and D. Harvey, "How Twitter is fighting spam and malicious automation", *Blog.twitter.com*, 2018. [Online]. Available:https://blog.twitter.com/official/en_us/topics/company/2018/howtwitter-is-fighting-spam-and-malicious-automation.html. [Accessed: 21- Apr- 2019].
- [13] P. Bourgonje, J. M. Schneider, & G. Rehm, (2017, September). From clickbait to fake news detection: an approach based on detecting the stance of headlines to articles. In *Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism* (pp. 84- 89).

- [14] A. Reyes, P. Rosso, & D. Buscaldi, (2012). From humor recognition to irony detection: The figurative language of social media. *Data & Knowledge Engineering*, 74, 1-12.
- [15] Z. Chu, S. Gianvecchio, H. Wang, & S. Jajodia, (2012). Detecting automation of twitter accounts: Are you a human, bot, or cyborg?. *IEEE Transactions on Dependable and Secure Computing*, 9(6), 811- 824.
- [16] X. Zhou, R. Zafarani, (2018). Fake news: A survey of research, detection methods, and opportunities. *arXiv preprint arXiv:1812.00315*.
- [17] V. Rubin, N. Conroy, Y. Chen, & S. Cornwell, (2016, June). Fake news or truth? using satirical cues to detect potentially misleading news. In *Proceedings of the second workshop on computational approaches to deception detection* (pp. 7-17).
- [18] Y. Chen, N. J. Conroy, & V. L. Rubin, (2015, November). Misleading online content: Recognizing clickbait as false news. In *Proceedings of the 2015 ACM on Workshop on Multimodal Deception Detection* (pp. 15-19). ACM.
- [19] Y. Chen, & V. L. Rubin, (2017). Perceptions of Clickbait: A QMethodology Approach. In *Proceedings of the 45th Annual Conference of The Canadian Association for Information Science/L'Association canadienne des sciences de l'information (CAIS/ACSI2017)*, Ryerson University, Toronto, May 31-June 2, 2017.
- [20] "Explained: What is Fake news? | Social Media and Filter Bubbles", *webwise.ie*, 2019. [Online]. Available: <https://www.webwise.ie/teachers/what-is-fake-news/>. [Accessed: 29- Aug- 2019].

APPENDIX

A) SOURCE CODE

Twitter Bot Detection: Exploratory Data Analysis

Basics

```
import pandas as pd
```

```
import psycopg2 as pg
```

```
import numpy as np
```

Visuals

```
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

```
import seaborn as sns
```

Postgres info to connect

```
connection_args = {
```

```
    'host': 'localhost',          # Connecting to _local_ version of psq
```

```
    'dbname': 'twitter_accounts', # DB with Twitter info
```

```
    'port': 5432                  # port opened on AWS
```

```
}
```

```

connection = pg.connect(**connection_args)

raw_df = pd.read_sql('SELECT * FROM human_bots', connection)

raw_df.head()

raw_df.info()

q = ""

SELECT account_type, COUNT(*) as total

FROM human_bots

GROUP BY account_type

""

account_counts = pd.read_sql(q, connection)

account_counts

num_humans = account_counts.total.values[1]

num_bots = account_counts.total.values[0]

types = ['Humans', 'Bots']

counts = [num_humans, num_bots]

plt.figure(figsize=(4, 4))

sns.barplot(x = types, y = counts)

plt.title("Number of Entries by Account Type", fontsize=11)

sns.despine();

q = ""

```

WITH ver as

```
(SELECT account_type, COUNT(*) as ver_true FROM human_bots WHERE
verified='true' GROUP BY account_type),
```

not_ver as

```
(SELECT account_type, COUNT(*) as ver_false FROM human_bots WHERE
verified='false' GROUP BY account_type)
```

```
SELECT v.account_type, v.ver_true, nv.ver_false FROM ver v JOIN not_ver nv
ON v.account_type = nv.account_type
```

'''

```
ver_status_by_type = pd.read_sql(q, connection)
```

```
ver_status_by_type
```

```
bots_verified = ver_status_by_type.ver_true.values[0]
```

```
bots_not_verified = ver_status_by_type.ver_false.values[0]
```

```
humans_verified = ver_status_by_type.ver_true.values[1]
```

```
humans_not_verified = ver_status_by_type.ver_false.values[1]
```

```
types = ['Not Verified', 'Verified']
```

```
bot_counts = [bots_not_verified, bots_verified]
```

```
human_counts = [humans_not_verified, humans_verified]
```

```
plt.figure(figsize=(8, 4), dpi=100)
```

```
plt.suptitle("Count by Verification Status", fontsize=12)
```

```
plt.subplot(1, 2, 1)
```

```

sns.barplot(x = types, y = human_counts)

plt.title("Humans", fontsize=11)

plt.ylabel("Count", fontsize=10)

sns.despine()

plt.subplot(1, 2, 2)

sns.barplot(x = types, y = bot_counts)

plt.title("Bots", fontsize=11)

sns.despine();

# drop extra index column

raw_df.drop(columns=['index'], inplace=True)

# Binary classifications for bots and boolean values

raw_df['bot'] = raw_df['account_type'].apply(lambda x: 1 if x == 'bot' else 0)

raw_df['default_profile'] = raw_df['default_profile'].astype(int)

raw_df['default_profile'] = raw_df['default_profile'].astype(int)

raw_df['default_profile_image'] = raw_df['default_profile_image'].astype(int)

raw_df['geo_enabled'] = raw_df['geo_enabled'].astype(int)

raw_df['verified'] = raw_df['verified'].astype(int)

# datetime conversion

raw_df['created_at'] = pd.to_datetime(raw_df['created_at'])

# hour created

```

```

raw_df['hour_created'] = pd.to_datetime(raw_df['created_at']).dt.hour

# usable df setup

df = raw_df[['bot', 'screen_name', 'created_at', 'hour_created', 'verified',
'acct_location', 'geo_enabled', 'lang', 'default_profile', 'default_profile_image',
'favourites_count', 'followers_count', 'friends_count', 'statuses_count',
'average_tweets_per_day', 'account_age_days']]

del raw_df

df.head()

df.info()

df.describe()

# Interesting features to look at:

df['avg_daily_followers'] = np.round(df['followers_count'] / df['account_age_days'])

df['avg_daily_friends'] = np.round(df['followers_count'] / df['account_age_days'])

df['avg_daily_favorites'] = np.round(df['followers_count'] / df['account_age_days'])

# Log transformations for highly skewed data

df['friends_log'] = np.round(np.log(1 + df['friends_count']), 3)

df['followers_log'] = np.round(np.log(1 + df['followers_count']), 3)

df['favs_log'] = np.round(np.log(1 + df['favourites_count']), 3)

df['avg_daily_tweets_log'] = np.round(np.log(1 + df['average_tweets_per_day']), 3)

# Possible interaction features

df['network'] = np.round(df['friends_log'] * df['followers_log'], 3)

```

```

df['tweet_to_followers'] = np.round(np.log( 1+ df['statuses_count']) * np.log(1+
df['followers_count']), 3)

# Log-transformed daily acquisition metrics for dist. plots

df['follower_acq_rate'] = np.round(np.log(1 + (df['followers_count'] /
df['account_age_days'])), 3)

df['friends_acq_rate'] = np.round(np.log(1 + (df['friends_count'] /
df['account_age_days'])), 3)

df['favs_rate'] = np.round(np.log(1 + (df['friends_count'] / df['account_age_days'])), 3)

plt.figure(figsize=(8,6), dpi=100)

cmap = sns.diverging_palette(230, 20, as_cmap=True)

sns.heatmap(df.corr(), cmap=cmap, annot=False)

plt.title('Correlation of potential features: all data', fontsize=14);

bots = df[df['bot'] == 1]

humans = df[df['bot'] == 0]

plt.figure(figsize=(8,6), dpi=100)

cmap = sns.diverging_palette(230, 20, as_cmap=True)

sns.heatmap(bots.corr(), cmap=cmap, annot=False)

plt.title('Correlation of potential features: bots only', fontsize=14);

plt.figure(figsize=(8,6), dpi=100)

cmap = sns.diverging_palette(230, 20, as_cmap=True)

sns.heatmap(humans.corr(), cmap=cmap, annot=False)

```

```

plt.title('Correlation of potential features: humans only', fontsize=14);

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='account_age_days', data=df, hue='bot', alpha=.25, kde=True,
stat='density', common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

plt.title('Distribution of Account Ages', fontsize=12)

plt.xlabel('Account Age (# days)', fontsize=10)

plt.ylabel('Density', fontsize=10)

sns.despine();

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='followers_log', data=df, hue='bot', alpha=.25, kde=True, stat='density',
common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

plt.title('Distribution of # Followers', fontsize=12)

plt.xlabel('Followers (Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 18)

plt.ylim(0, 0.11)

sns.despine();

```

```

plt.savefig('imgs/dist_followers.png');

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='friends_log', data=df, hue='bot', alpha=.25, kde=True, stat='density',
common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

plt.title('Distribution of Friends (# Following)', fontsize=12)

plt.xlabel('Friends Count (Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 16)

plt.ylim(0, 0.65)

sns.despine();

plt.figure(figsize=(12,5), dpi=100)

plt.suptitle('Distribution of Followers & Friends', fontsize=14)

plt.subplot(1, 2, 1)

sns.histplot(x='followers_log', data=df, hue='bot', alpha=.25, kde=True, stat='density',
common_bins=True, element='step', legend=False)

plt.title('Distribution of # Followers', fontsize=12)

plt.xlabel('Followers (Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 18)

```

```

plt.ylim(0, 0.25)

sns.despine()

plt.subplot(1, 2, 2)

sns.histplot(x='friends_log', data=df, hue='bot', alpha=.25, kde=True, stat='density',
common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

plt.title('Distribution of Friends (# Following)', fontsize=12)

plt.xlabel('Friends Count (Log10 scale)', fontsize=10)

plt.ylabel("")

plt.yticks([])

plt.xlim(0, 18)

plt.ylim(0, 0.25)

sns.despine();

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='network', data=df, hue='bot', alpha=.25, kde=True, stat='density',
common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

plt.title('Distribution of Network Size', fontsize=12)

plt.xlabel('Network Size [Friends * Followers] (Log10 scale)', fontsize=10)

```

```

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 150)

plt.ylim(0, 0.05)

sns.despine();

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='favs_log', data=df, hue='bot', alpha=.25, kde=True, stat='density',
common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

plt.title("Distribution of Favorites Count", fontsize=12)

plt.xlabel('Favorites Count(Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 14)

plt.ylim(0, 0.3)

sns.despine();

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='avg_daily_tweets_log', data=df, hue='bot', alpha=.25, kde=True,
stat='density', common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

plt.title("Distribution of Tweets per Day", fontsize=12)

```

```

plt.xlabel('Average Tweets per Day (Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 5)

plt.ylim(0, 0.7)

sns.despine();

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='follower_acq_rate', data=df, hue='bot', alpha=.25, kde=True,
stat='density', common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

plt.title("Distribution of Follower Acquisition Rate", fontsize=12)

plt.xlabel('Followers per Day (Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 10)

plt.ylim(0, 2.5)

sns.despine();

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='friends_acq_rate', data=df, hue='bot', alpha=.25, kde=True,
stat='density', common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

```

```

plt.title("Distribution of Friend Acquisition Rate", fontsize=12)

plt.xlabel('Friends per Day (Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 1)

plt.ylim(0, 6)

sns.despine();

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='tweet_to_followers', data=df, hue='bot', alpha=.25, kde=True,
stat='density', common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='upper right', labels=['Bot', 'Human'],
frameon=False)

plt.title("Tweets to Followers Metric", fontsize=12)

plt.xlabel('Tweets * Followers (Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 200)

plt.ylim(0, 0.009)

sns.despine();

plt.figure(figsize=(7,5), dpi=100)

sns.histplot(x='hour_created', data=df, hue='bot', alpha=.25, kde=True, stat='density',
common_bins=True, element='step', legend=True)

plt.legend(title='Account Type', loc='best', labels=['Bot', 'Human'])

```

```

plt.title("Time of Day Account was Created", fontsize=12)

plt.xlabel('Time of Day (24h)', fontsize=10)

plt.ylabel('Density', fontsize=10)

plt.xlim(0, 24)

plt.ylim(0, 0.05)

sns.despine();

plt.figure(figsize=(12,5), dpi=100)

plt.suptitle('Tweets-Followers Metric by Verification Status', fontsize=14)

plt.subplot(1, 2, 1)

sns.histplot(x='tweet_to_followers', data=humans, hue='verified', alpha=.25,
kde=True, stat='density', common_bins=True, element='step', legend=False)

plt.title("Humans", fontsize=12)

plt.xlabel('Tweets * Followers (Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

sns.despine(bottom = True, left = True)

plt.xlim(0, 200)

plt.ylim(0, 0.025)

plt.subplot(1, 2, 2)

sns.histplot(x='tweet_to_followers', data=bots, hue='verified', alpha=.25, kde=True,
stat='density', common_bins=True, element='step', legend=True)

```

```

plt.legend(title='Verification Status', loc='upper right', labels=['Verified', 'Not Verified'],
frameon=False)

plt.title("Bots", fontsize=12)

plt.xlabel('Tweets * Followers (Log10 scale)', fontsize=10)

plt.ylabel("")

plt.yticks([])

sns.despine()

plt.xlim(0, 200)

plt.ylim(0, 0.025);

plt.figure(figsize=(12,5), dpi=100)

plt.suptitle('Network Size by Verification Status', fontsize=14)

plt.subplot(1, 2, 1)

sns.histplot(x='network', data=humans, hue='verified', alpha=.25, kde=True,
stat='density', common_bins=True, element='step', legend=False)

plt.xlabel('Network Size (Friends * Followers)')

plt.title("Humans", fontsize=12)

plt.xlabel('Network Size (Log10 scale)', fontsize=10)

plt.ylabel('Density', fontsize=10)

sns.despine(bottom = True, left = True)

plt.xlim(0, 150)

```

```

plt.ylim(0, 0.05)

plt.subplot(1, 2, 2)

sns.histplot(x='network', data=bots, hue='verified', alpha=.25, kde=True,
stat='density', common_bins=True, element='step', legend=True)

plt.xlabel('Network Size (Friends * Followers)')

plt.legend(title='Verification Status', loc='upper right', labels=['Verified', 'Not Verified'],
frameon=False)

plt.title("Bots", fontsize=12)

plt.xlabel('Network Size (Log10 scale)', fontsize=10)

plt.ylabel("")

plt.yticks([])

sns.despine()

plt.xlim(0, 150)

plt.ylim(0, 0.05);

```

Twitter Bot Detection: Classification Modeling

Basics

```

import pandas as pd

import psycopg2 as pg

import numpy as np

import pickle

```

```

# Visuals

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

from model_evaluation import *

# Models

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from sklearn.svm import SVC

from xgboost import XGBClassifier

# Model support

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import (accuracy_score, classification_report, confusion_matrix,
f1_score, auc, plot_confusion_matrix, precision_score, recall_score, roc_auc_score,
roc_curve, precision_recall_curve)

# Postgres info to connect

```

```

connection_args = {

    'host': 'localhost',          # Connecting to _local_ version of psql

    'dbname': 'twitter_accounts', # DB with Twitter info

    'port': 5432                  # port we opened on AWS

}

connection = pg.connect(**connection_args)

raw_df = pd.read_sql('SELECT * FROM human_bots', connection)

# drop funny index column

raw_df.drop(columns=['index'], inplace=True)

# Binary classifications for bots and boolean values

raw_df['bot'] = raw_df['account_type'].apply(lambda x: 1 if x == 'bot' else 0)

raw_df['default_profile'] = raw_df['default_profile'].astype(int)

raw_df['default_profile'] = raw_df['default_profile'].astype(int)

raw_df['default_profile_image'] = raw_df['default_profile_image'].astype(int)

raw_df['geo_enabled'] = raw_df['geo_enabled'].astype(int)

raw_df['verified'] = raw_df['verified'].astype(int)

# datetime conversion

raw_df['created_at'] = pd.to_datetime(raw_df['created_at'])

# hour created

raw_df['hour_created'] = pd.to_datetime(raw_df['created_at']).dt.hour

```

```

# usable df setup

df = raw_df[['bot', 'screen_name', 'created_at', 'hour_created', 'verified',
'acct_location', 'geo_enabled', 'lang', 'default_profile', 'default_profile_image',
'favourites_count', 'followers_count', 'friends_count', 'statuses_count',
'average_tweets_per_day', 'account_age_days']]

del raw_df

# Interesting features to look at:

df['avg_daily_followers'] = np.round((df['followers_count'] / df['account_age_days']),
3)

df['avg_daily_friends'] = np.round((df['followers_count'] / df['account_age_days']), 3)

df['avg_daily_favorites'] = np.round((df['followers_count'] / df['account_age_days']), 3)

# Log transformations for highly skewed data

df['friends_log'] = np.round(np.log(1 + df['friends_count']), 3)

df['followers_log'] = np.round(np.log(1 + df['followers_count']), 3)

df['favs_log'] = np.round(np.log(1 + df['favourites_count']), 3)

df['avg_daily_tweets_log'] = np.round(np.log(1+ df['average_tweets_per_day']), 3)

# Possible interactive features

df['network'] = np.round(df['friends_log'] * df['followers_log'], 3)

df['tweet_to_followers'] = np.round(np.log( 1 + df['statuses_count']) * np.log(1+
df['followers_count']), 3)

# Log-transformed daily acquisition metrics for dist. plots

```

```

df['follower_acq_rate'] = np.round(np.log(1 + (df['followers_count'] /
df['account_age_days'])), 3)

df['friends_acq_rate'] = np.round(np.log(1 + (df['friends_count'] /
df['account_age_days'])), 3)

df['favs_rate'] = np.round(np.log(1 + (df['friends_count'] / df['account_age_days'])), 3)

df.head()

features = ['verified', 'geo_enabled', 'default_profile', 'default_profile_image',
'favourites_count', 'followers_count', 'friends_count', 'statuses_count',
'average_tweets_per_day', 'network', 'tweet_to_followers', 'follower_acq_rate',
'friends_acq_rate', 'favs_rate']

X = df[features]

y = df['bot']

X, X_test, y, y_test = train_test_split(X, y, test_size=.3, random_state=1234)

# Models that require scaling:

knn = KNeighborsClassifier(n_neighbors=10)

lr = LogisticRegression()

# Scaling

scalar = StandardScaler()

scalar.fit(X)

X_train_scaled = scalar.transform(X)

model_list = [knn, lr]

```

```

kf = KFold(n_splits=5, shuffle=True, random_state=33)

multi_model_eval(model_list, X_train_scaled, y, kf)

# Models that don't require scaling

gnb = GaussianNB()

bnb = BernoulliNB()

mnb = MultinomialNB()

tree = DecisionTreeClassifier()

forest = RandomForestClassifier()

xgb = XGBClassifier()

model_list = [gnb, bnb, mnb, tree, forest, xgb]

kf = KFold(n_splits=3, shuffle=True, random_state=33)

multi_model_eval(model_list, X, y, kf)

forest = RandomForestClassifier()

kf = KFold(n_splits=5, shuffle=True, random_state=33)

roc_curve_cv(forest, X, y, kf, model_alias='RandomForest')

precision_recall_cv(forest, X, y, kf, model_alias='RandomForest')

xgb = XGBClassifier()

kf = KFold(n_splits=5, shuffle=True, random_state=33)

roc_curve_cv(xgb, X, y, kf, model_alias='XGBoost')

precision_recall_cv(xgb, X, y, kf, model_alias='XGBoost')

```

```

num_bots = len(df[df['bot'] == 1])

num_humans = len(df[df['bot'] == 0])

print("Number of bots: ", num_bots)

print("Number of humans: ", num_humans)

print(f'Bots / Total %: {(num_bots / len(df))*100:.2f}')

types = ['Humans', 'Bots']

counts = [num_humans, num_bots]

plt.figure(figsize=(4, 4))

sns.barplot(x = types, y = counts)

plt.title("Number of Entries by Account Type", fontsize=11)

sns.despine();

# For XGBoost

estimate = num_humans/num_bots

estimate

forest = RandomForestClassifier(class_weight='balanced')

xgb = XGBClassifier(scale_pos_weight=estimate)

models = [forest, xgb]

kf = KFold(n_splits=5, shuffle=True, random_state=33)

multi_model_eval(models, X, y, kf)

roc_curve_cv(forest, X, y, kf, model_alias='RandomForest')

```

```

precision_recall_cv(forest, X, y, kf, model_alias='RandomForest')

roc_curve_cv(xgb, X, y, kf, model_alias='XGBoost')

precision_recall_cv(xgb, X, y, kf, model_alias='XGBoost')

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=.25,
random_state=3333)

forest = RandomForestClassifier(class_weight='balanced')

forest.fit(X_train, y_train)

preds = forest.predict(X_val)

metrics_report(preds, y_val)

xgb = XGBClassifier(scale_pos_weight=estimate)

xgb.fit(X_train, y_train)

preds = xgb.predict(X_val)

metrics_report(preds, y_val)

plot_feature_importance(forest, features, model_alias='RandomForest')

plot_feature_importance(xgb, features, model_alias='XGBoost')

df.head(3)

features = ['verified',b'geo_enabled', 'default_profile', 'default_profile_image',
'favourites_count', 'followers_count', 'friends_count', 'statuses_count',
'average_tweets_per_day', 'network', 'tweet_to_followers', 'follower_acq_rate',
'friends_acq_rate', 'favs_rate']

X = df[features]

```

```

y = df['bot']

X, X_test, y, y_test = train_test_split(X, y, test_size=.3, random_state=1234)

xgb = XGBClassifier(scale_pos_weight=1.8, tree_method='hist', learning_rate=0.1,
eta=0.01, max_depth=7, gamma=0.05, n_estimators=200, colsample_bytree=.8)

model_list = [xgb]

kf = KFold(n_splits=5, shuffle=True, random_state=33)

multi_model_eval(model_list, X, y, kf)

roc_curve_cv(xgb, X, y, kf, model_alias='XGBoost')

precision_recall_cv(xgb, X, y, kf, model_alias='XGBoost')

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=.3,
random_state=1234)

forest = RandomForestClassifier(class_weight='balanced')

forest = forest.fit(X_train, y_train)

xgb = XGBClassifier(scale_pos_weight=1.8, tree_method='hist', learning_rate=0.1,
eta=0.01,max_depth=7, gamma=0.05,n_estimators=200,colsample_bytree=.8)

xgb = xgb.fit(X_train, y_train)

models = [('forest', forest), ('xgb', xgb)]

voting_classifier = VotingClassifier(estimators=models, voting='soft', n_jobs=-1)

voting_classifier = voting_classifier.fit(X_train, y_train)

voting_classifier_prediction = voting_classifier.predict(X_val)

metrics_report(voting_classifier_prediction, y_val)

```

```

# Full train & test

best_model=XGBClassifier(scale_pos_weight=1.8,tree_method='hist',learning_rate=
0.1, eta=0.01,max_depth=7, gamma=0.05,n_estimators=200, colsample_bytree=.8 )

best_model.fit(X, y)

best_model_prediction = best_model.predict(X_test)

metrics_report(best_model_prediction, y_test)

fpr, tpr, thresholds = roc_curve(y_test, best_model.predict_proba(X_test)[:,:1])

model_auc = roc_auc_score(y_test, best_model.predict_proba(X_test)[:,:1])

plt.figure(figsize=(4, 4), dpi=100)

plt.plot(fpr, tpr,lw=2, label=f'AUC: {model_auc:.4f}')

plt.plot([0,1],[0,1],c='grey',ls='--', label='Chance Line')

plt.xlim([-0.05,1.05])

plt.ylim([-0.05,1.05])

plt.xlabel('False positive rate', fontsize=10)

plt.ylabel('True positive rate', fontsize=10)

plt.title('ROC Curve for Best XGBoost Model', fontsize=11)

plt.legend(loc='lower right', prop={'size': 8}, frameon=False)

sns.despine()

print(f'ROC AUC score: {model_auc:.4f}')

print("")

```

```

plt.show();

# Prec Recal Curve here

model_precision, model_recall, _ = precision_recall_curve(y_test,
best_model.predict_proba(X_test)[:,-1])

# plot the precision-recall curves

no_skill = len(y_test[y_test==1]) / len(y_test)

plt.figure(figsize=(4, 4), dpi=100)

plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')

plt.plot(model_recall, model_precision, marker=',', label='XGBoost')

# axis labels

plt.title('Precision-Recall Curve for Best XGBoost Model', fontsize=11)

plt.xlabel('Recall', fontsize=10)

plt.ylabel('Precision', fontsize=10)

# show the legend

plt.legend(loc='upper right', prop={'size': 8}, frameon=False)

# show the plot

sns.despine()

pred = best_model.predict(X_test)

prec_score = precision_score(y_test, pred)

rec_score = recall_score(y_test, pred)

```

```
print(f'Precision score: {prec_score:.4f}')

print(f'Recall score: {rec_score:.4f}')

print("")

plot_feature_importance(best_model, features, model_alias='XGBoost')

fully_trained_model = XGBClassifier(scale_pos_weight=1.8,tree_method='hist',
learning_rate=0.1,eta=0.01,max_depth=7,gamma=0.05,n_estimators=200,colsample
_bytree=.8 )

fully_trained_model.fit(X, y)
```

B) SCREENSHOTS

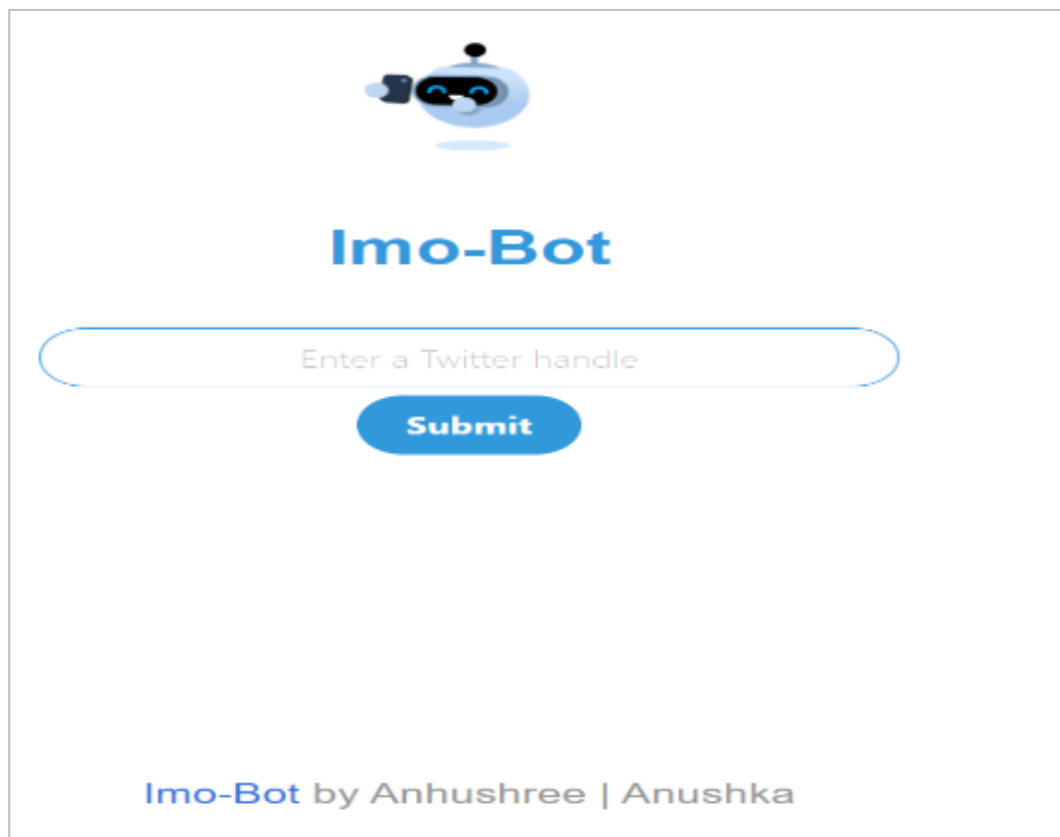


Fig:7.1 User Interface of our project



Imo-Bot

Enter a Twitter handle

Submit

Prediction for @Enhypen

Not a bot

Probability of being a bot: 4.4%



Imo-Bot

Enter a Twitter handle

Submit

Prediction for @harsh_casper

Not a bot

Probability of being a bot: 3.54%

Imo-Bot by Anhushree | Anushka

Fig. 7.2. Shows the probability percentage of the given account along with the prediction

C) PUBLICATION WITH PLAGIARISM REPORT



Document Information

Analyzed document	Anhushree_Anushka_Discernment of Autonomous Profiles on SNS.pdf (D125913574)
Submitted	2022-01-24T04:15:00.0000000
Submitted by	R. Vignesh
Submitter email	vignesh.cse@sathyabama.ac.in
Similarity	8%
Analysis address	vignesh.cse.sathya@analysis.urkund.com

Sources included in the report

	URL: https://www.irjet.net/archives/V7/i5/IRJET-V7I5312.pdf Fetched: 2020-09-02T13:20:33.8130000		1
	URL: https://www.iiste.org/Journals/index.php/NMMC/article/download/52605/54365		.