A

Mini Project Report on

UNVEILING THE POWER OF EXTREME LEARNING MACHINE: COMBATTING SPAM AND IDENTIFYING FAKE USERS ON TWITTER

Submitted for partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

DEPARTMENT OF INFORMATION TECHNOLOGY

CONTENTS **PAGE NUMBER**

ABSTRACT

Social networking sites engage millions of users around the world. The users' interactions with these social sites, such as Twitter and Facebook have a tremendous impact and occasionally undesirable repercussions for daily life. The prominent social networking sites have turned into a target platform for the spammers to disperse a huge amount of irrelevant and deleterious information. Twitter, for example, has become one of the most extravagantly used platforms of all times and therefore allows an unreasonable amount of spam. Fake users send undesired tweets to users to promote services or websites that not only affect legitimate users but also disrupt resource consumption. Moreover, the possibility of expanding invalid information to users through fake identities has increased that results in the unrolling of harmful content. Recently, the detection of spammers and identification of fake users on Twitter has become a common area of research in contemporary online social Networks (OSNs). This work proposes the detection of spammers and fake user identification on Twitter data using extreme learning machine (ELM) and compared the obtained results with various machine learning algorithms like random forest, naevi bayes and support vector machine. Moreover, a taxonomy of the Twitter spam detection approaches is presented that classifies the techniques based on their ability to detect: (i) fake content, (ii) spam based on URL, (iii) spam in trending topics, and (iv) fake users. The presented techniques are also compared based on various features, such as user features, content features, graph features, structure features, and time features.

LIST OF FIGURES

List of Acronyms and Definitions

S.No.	Acronym	Definition
01	OSN	Open Storage Network
02	URL	Uniform Resource Locator
03	ELM	Extreme Learning Machine
04	MSN	Microsoft Network
05	QR	Quick Response
06	APT	Advanced Persistent Threat
07	P2P	Peer to Peer
08	HPSD	High PU Spam Detection
09	SGD	Stochastic Gradient Descent
10	ML	Machine Learning
11	RF	Random Forest
12	SVM	Support Vector Machine
13	ReLU	Rectified Linear Unit
14	MSE	Mean Squared Error
15	JDBC	Java Database Connectivity
16	ODBC	Open Database Connectivity

CHAPTER 1

INTRODUCTION

1.1 Overview

In recent years, MSNs such as Twitter, Facebook and Sina Weibo have become important platforms for people to obtain information, spread information and make friends. Twitter's monthly active users were 200 million in 2012, and the figure rose to 328 million in 2017, with 20 million tweets being posted every hour. While MSNs enrich people's lives, some security issues have emerged. Attackers spread attacks through MSNs, such as phishing, drive-by download, malicious code injection and so on. According to new research, up to 15 percent of Twitter accounts are in fact bots rather than people. Malicious URLs are one of the most common methods used by attackers to initiate cyberattacks. Attackers trick users into clicking malicious URLs, clicking pictures containing malicious URLs, scanning QR codes with malicious URLs, and so on by disguising themselves as well-known accounts, advertisements of discounted merchandise, or by using mutual trust between friends. In these ways, attackers lure victims to a phishing website for phishing attacks or embed malicious software in the victim's computer to control the target host or perform an APT attack, which will cause huge losses to individuals, businesses, governments and organizations. Many MSNs use blacklist techniques to filter URLs sent by users, such as Google Safe Browsing, Phishing Tank, and so on. However, there is often a delay in blacklist technology, and research shows that 90 percent of victims click on malicious URLs before they are blacklisted. In order to provide users with a secure MSN environment, researchers have proposed many strategies to deal with online social network attacks. Existing detection methods are mainly divided into two categories. The first category includes detection algorithms based on the relation graph of social networks. Many kinds of relations exist in social networks, so researchers use relations in social networks to build a social graph. By analysing the characteristics of the user’s location in the graph, a detection algorithm can be designed based on the graph to identify suspicious messages or users. The second category includes detection methods based on machine learning algorithms.

1.2 Research Motivation

Researchers extract features from social network data such as users' personal information, social behaviours, relationships with friends, message content and so on, then use machine learning algorithms to train classifiers to identify malicious messages or users. Spam has been known as a major problem since long, but its impact on the global network infrastructure has now reached epidemic proportions. In the earliest days of spam, users could simply delete the offending messages. Later, when spam became more common, several client-side spam filtering tools became available, but they were often unreliable: users had to scan alleged spam to ensure that no important messages were deleted by mistake, with an increasing loss of time. Due to customers' complaints, governments started to contemplate anti-spam legislation, while several companies began offering spam filtering products to mail server operators. When these countermeasures first reached the market, it seemed that simple economics could support a rapid eradication of spam: filtering out 95% of spam would suffice to increase the spammers' cost to reach the same audience by a factor of 20. It looked therefore reasonable to assume that high-accuracy filters could put a definitive end to spam, as few spammers had profit margins big enough to meet the cost increase. Unfortunately, things went quite differently: while most commercial anti-spam filters claim a much higher success rate than 95% in identifying spam, a huge amount of it still winds up in users' in-boxes, even when client-side and server-side filters are used in conjunction. It may be argued that this lack of success in the war against spam is partly due to the elusive nature of the notion, which is difficult to identify by means of a software program. Of course, many messages leave no doubt: drugs, pornography, fraud, and viruses now vastly outweigh the occasional unsolicited product or service sales pitch. However, there are many borderline cases where what is spam for a user could be useful information for the next person, and it may seem unwise to curb the potential of email as a mass communication channel in the spur of indignation against spam. Recently, some approaches based on the development of a P2P network for the collaborative sharing of knowledge about spam between users have been proposed. While these approaches represent a step toward the design of a P2P collaborative spam filtering solution, they do not pay adequate attention to the aspects of message confidentiality and robustness against attacks.

1.3 Problem Statement

Social media platforms like Twitter have become integral parts of our daily lives, serving as hubs for information dissemination, social interaction, and content sharing. However, this open and accessible nature also makes these platforms susceptible to abuse, including the spread of spam content and the presence of fake user accounts. Detecting and combatting spam and identifying fake users on Twitter have become critical challenges, as these activities not only erode the user experience but also pose security and privacy risks. Spam content, consisting of unsolicited and often deceptive messages, floods users' timelines and undermines the credibility of information shared on the platform. Furthermore, fake user accounts, also known as bots or trolls, are designed to impersonate real users and engage in a variety of malicious activities, such as disseminating misinformation, amplifying divisive content, and orchestrating coordinated attacks. These issues collectively contribute to the degradation of the overall quality of user interactions and content on Twitter. To address these challenges effectively, this research aims to harness the power of Extreme Learning Machine (ELM), an emerging machine learning paradigm known for its computational efficiency and scalability. ELM offers the potential to develop robust and high-performance models for spam detection and fake user identification on Twitter, facilitating a more secure, trustworthy, and engaging user experience.

1.4 Applications

The application of Extreme Learning Machines (ELM) in combatting spam and identifying fake users on Twitter can have several practical use cases. Here are some applications:

- **Spam Detection:** ELM-based models can be employed to automatically identify and filter out spam tweets and messages in real-time. This application helps in maintaining the quality of users' timelines and preventing spammy content from flooding the platform.

- **Fake User Identification:** ELM can assist in detecting fake user accounts, bots, and trolls by analysing patterns in their activity, such as posting frequency, retweet behaviour, and engagement with other users. Identifying and suspending fake accounts helps in reducing the spread of misinformation and malicious activities.

- **Content Moderation:** ELM models can be integrated into Twitter's content moderation system to flag and remove offensive or harmful content, ensuring a safer and more respectful online environment.
- **Sentiment Analysis:** ELM can be used to perform sentiment analysis on tweets and user comments. This can help identify negative or toxic sentiment in posts, enabling timely responses to instances of cyberbullying or harassment.
- **Anomaly Detection:** ELM models can be trained to detect unusual and suspicious behaviour patterns among users, helping to identify potential threats or coordinated attacks, including those involving fake users.
- **Phishing and Scam Detection:** ELM can assist in identifying tweets or direct messages containing phishing links or fraudulent schemes, protecting users from falling victim to scams or phishing attacks.
- **User Account Verification:** Twitter can use ELM models to enhance its user account verification process by cross-referencing user data and behaviour against known patterns of fake or suspicious accounts.
- **Trending Topic Analysis:** ELM can help in analysing the authenticity of trending topics and hashtags. It can identify instances where fake users artificially inflate the popularity of certain topics for manipulative purposes.
- **Ad Quality Control:** ELM-based models can contribute to ensuring that advertisements on Twitter are not deceptive or misleading. This helps in maintaining the platform's integrity and trustworthiness.
- **Enhanced User Experience:** By reducing spam and fake accounts, ELM-powered systems can significantly improve the overall user experience on Twitter, making it more enjoyable and trustworthy for users.
- **Customized Recommendations:** ELM can analyse user behaviour to make more accurate content recommendations, improving user engagement and satisfaction while reducing the likelihood of users encountering spam or fake content.

CHAPTER 2

LITERATURE SURVEY

Elastic detection framework based on deep learning, which sets up a detection system at the mobile terminal and the server, respectively. Messages are first detected on the mobile terminal, and then the detection results are forwarded to the server along with the messages. We also design a detection queue, according to which the server can detect messages elastically when computing resources are limited, and more computing resources can be used for detecting more suspicious messages. We evaluate our detection framework on a Sina Weibo dataset. The results of the experiment show that our detection framework can improve the utilization rate of computing resources and can realize real-time detection with a high detection rate at a low false positive rate.

Damiani proposed a decentralized privacy-preserving approach to spam filtering. Our solution exploits robust digests to identify messages that are a slight variation of one another and a structured peer-to-peer architecture between mail servers to collaboratively share knowledge about spam.

Houston investigated whether sentiment analysis can help spammer detection in online social media. In particular, we first conduct an exploratory study to analyse the sentiment differences between spammers and normal users, and then present an optimization formulation that incorporates sentiment information into a novel social spammer detection framework. Experimental results on real-world social media datasets show the superior performance of the proposed framework by harnessing sentiment analysis for social spammer detection.

Mateen proposed a hybrid technique which uses content-based as well as graph-based features for identification of spammers on twitter platform. We have analysed the proposed technique on real Twitter dataset with 11k uses and more than 400k tweets approximately. Our results show that the detection rate of our proposed technique is much higher than any of the existing techniques.

Wuet proposed a hybrid semi supervised learning model titled hybrid PU-learning-based spammer detection (hPSD) for spammer detection to leverage both the users' characteristics and the user-product relations. Specifically, the hPSD model can iteratively detect multitype spammers by injecting different positive samples, and allows the construction of classifiers in a semi supervised hybrid learning framework. Comprehensive experiments on movie dataset with shilling injection confirm the superior performance of hPSD over existing baseline methods. The hPSD is then

utilized to detect the hidden spammers from real-life Amazon data. A set of spammers and their underlying employers are successfully discovered and validated. These demonstrate that hPSD meets the real-world application scenarios and can thus effectively detect the potentially deceptive review writers.

Thomas presented Monarch, a real-time system that crawls URLs as they are submitted to web services and determines whether the URLs direct to spam. We evaluate the viability of Monarch and the fundamental challenges that arise due to the diversity of web service spam. We show that Monarch can provide accurate, real-time protection, but that the underlying characteristics of spam do not generalize across web services. In particular, we find that spam targeting email qualitatively differs in significant ways from spam campaigns targeting Twitter. We explore the distinctions between email and Twitter spam, including the abuse of public web hosting and redirector services. Finally, we demonstrate Monarch's scalability, showing our system could protect a service such as Twitter -- which needs to process 15 million URLs/day -- for a bit under $800/day.

Wang proposed a novel concept of a heterogeneous review graph to capture the relationships among reviewers, reviews and stores that the reviewers have reviewed. We explore how interactions between nodes in this graph can reveal the cause of spam and propose an iterative model to identify suspicious reviewers. This is the first time such intricate relationships have been identified for review spam detection. We also develop an effective computation method to quantify the trustiness of reviewers, the honesty of reviews, and the reliability of stores. Different from existing approaches, we don't use review text information. Our model is thus complementary to existing approaches and able to find more difficult and subtle spamming activities, which are agreed upon by human judges after they evaluate our results.

Shehnepoor proposed a novel framework, named Net Spam, which utilizes spam features for modelling review data sets as heterogeneous information networks to map spam detection procedure into a classification problem in such networks. Using the importance of spam features helps us to obtain better results in terms of different metrics experimented on real-world review data sets from Yelp and Amazon Web sites. The results show that Net Spam outperforms the existing methods and among four categories of features, including review-behavioural, user-behavioural, review-linguistic, and user-linguistic, the first type of features performs better than the other categories.

Masood performed a review of techniques used for detecting spammers on Twitter. Moreover, a taxonomy of the Twitter spam detection approaches is presented that classifies the techniques based on their ability to detect: (i) fake content, (ii) spam based on URL, (iii) spam in trending topics, and (iv) fake users. The presented techniques are also compared based on various features, such as user features, content features, graph features, structure features, and time features. We are hopeful that the presented study will be a useful resource for researchers to find the highlights of recent developments in Twitter spam detection on a single platform.

CHAPTER 3
EXISTING SYSTEM

Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "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." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

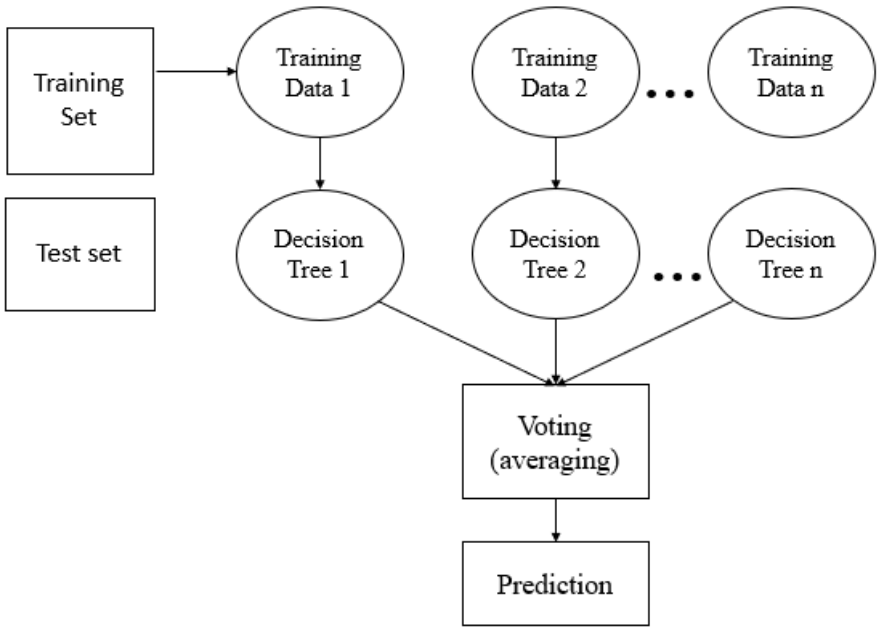


Fig. 3.1: Random Forest algorithm.

Random Forest algorithm

Step 1: In Random Forest n number of random records are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression respectively.

Important Features of Random Forest

- **Diversity**- Not all attributes/variables/features are considered while making an individual tree, each tree is different.
- **Immune to the curse of dimensionality**- Since each tree does not consider all the features, the feature space is reduced.
- **Parallelization**-Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.
- **Train-Test split**- In a random forest we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.
- **Stability**- Stability arises because the result is based on majority voting/ averaging.

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random Forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Below are some points that explain why we should use the Random Forest algorithm

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

Types of Ensembles

Before understanding the working of the random forest, we must look into the ensemble technique. Ensemble simply means combining multiple models. Thus, a collection of models is used to make predictions rather than an individual model. Ensemble uses two types of methods:

Bagging– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest. Bagging, also known as Bootstrap Aggregation is the ensemble technique used by random forest. Bagging chooses a random sample from the data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as row sampling. This step of row sampling with replacement is called bootstrap. Now each model is trained independently which generates results. The final output is based on majority voting after combining the results of all models. This step which involves combining all the results and generating output based on majority voting is known as aggregation.

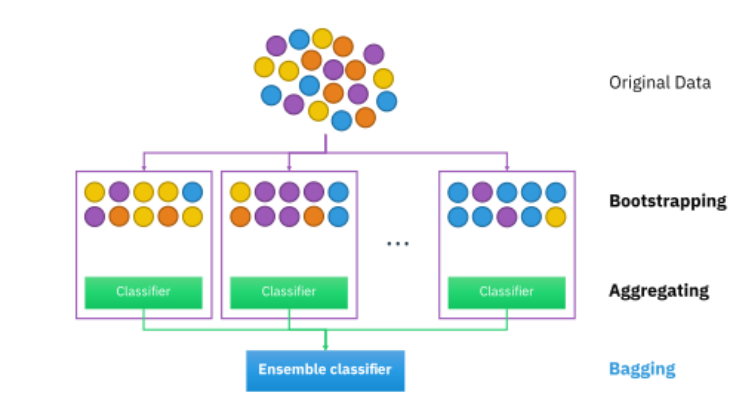


Fig. 3.2: RF Classifier analysis.

Boosting– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy.

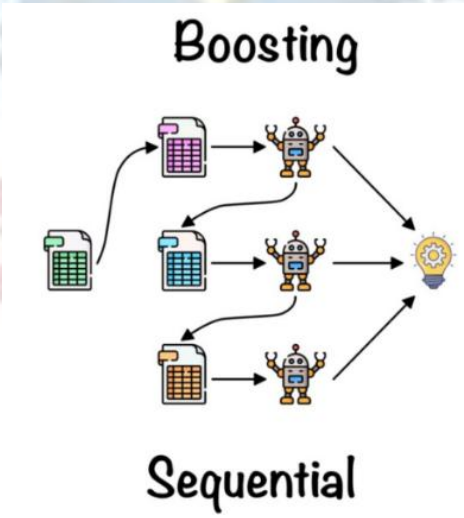


Fig. 3.3: Boosting RF Classifier.

Drawbacks of existing system

- Although random forests can be an improvement on single decision trees, more sophisticated techniques are available. Prediction accuracy on complex problems is usually inferior to gradient-boosted trees.
- A forest is less interpretable than a single decision tree. Single trees may be visualized as a sequence of decisions.
- A trained forest may require significant memory for storage, due to the need for retaining the information from several hundred individual trees.

.

CHAPTER 4

PROPOSED SYSTEM

4.1 Overview

Twitter can use ELM models to identify and report accounts engaged in illegal activities or violating the platform's terms of service, aiding in legal actions and regulatory compliance. The application of Extreme Learning Machines in combatting spam and identifying fake users on Twitter has the potential to enhance the platform's security, content quality, and user experience. These applications help in creating a safer and more trustworthy online environment for Twitter's vast user base. Figure 4.1 shows the block diagram of proposed system.

Step 1. Twitter Spam Dataset: Start with a labelled Twitter spam dataset, which contains tweets that are classified as either spam or non-spam (legitimate). These datasets can be collected by manually labelling tweets or using publicly available datasets specifically designed for spam detection.

Step 2. Data Preprocessing: Data preprocessing is a crucial step to clean and prepare the Twitter data for analysis. Common preprocessing steps include:

- **Text Cleaning:** Removing special characters, URLs, mentions, and hashtags from tweets.
- **Tokenization:** Splitting the text into individual words or tokens.
- **Lowercasing:** Converting all text to lowercase to ensure consistency.
- **Stop word Removal:** Removing common stop words (e.g., "and," "the," "in") that do not carry much information.
- **Stemming or Lemmatization:** Reducing words to their root form to reduce dimensionality.
- **Feature Engineering:** Creating additional features, such as tweet length, number of hashtags, and number of mentions.

Step 3. Naive Bayes Feature Extraction: After preprocessing, we can use the Naive Bayes algorithm to extract features from the cleaned text data. Naive Bayes is often used for text classification tasks. Here's how it works:

- **Feature Vector:** Represent each tweet as a feature vector where each feature corresponds to a word in the vocabulary. The value of each feature is typically based on word frequencies, such as term frequency-inverse document frequency (TF-IDF) or simple word counts.

- **Training the Naive Bayes feature extraction:** Train a Naive Bayes feature extraction on the feature vectors using the labelled data. The classifier learns the probabilities of words appearing in spam and non-spam tweets.
- **Feature Selection:** Depending on the size and quality of dataset, perform feature selection to choose the most informative features while reducing noise.

Step 4. ELM Classifier: Now, we can utilize an ELM classifier to make predictions on whether a given tweet is spam or not. ELM is chosen for its computational efficiency and potential to handle high-dimensional data.

- **Feature Mapping:** Transform the feature vectors obtained from Naive Bayes into a high-dimensional space using a random hidden layer in ELM. This layer is often composed of random weights and activation functions.
- **Training the ELM Classifier:** Train the ELM classifier using the transformed feature vectors as inputs and the corresponding labels (spam or non-spam) as targets. ELM's training process involves solving a linear system of equations to optimize the output weights.
- **Model Evaluation:** Evaluate the performance of the ELM classifier using metrics such as accuracy, precision, recall,. Cross-validation can be used to ensure robustness.

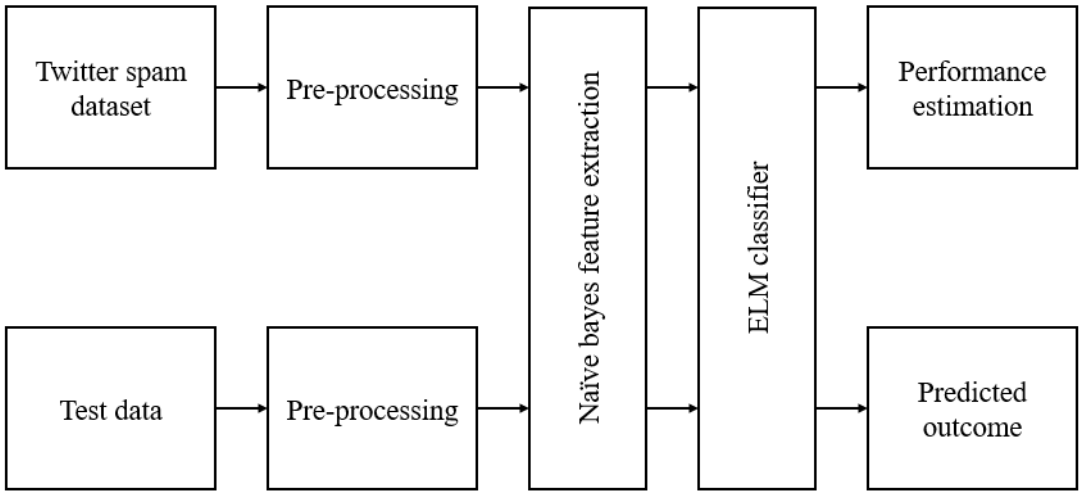


Fig. 4.1: Block diagram of proposed system.

4.2 Data Preprocessing

Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data pre-processing task. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set
- Feature scaling.

4.3 Naive Bayes Feature extraction

A Naive Bayes feature extraction is a probabilistic machine learning algorithm that is used for classification tasks, particularly in text classification and spam filtering. It is based on Bayes' theorem, which describes the probability of an event, based on prior knowledge of conditions that might be related to the event. The "Naive" part of its name comes from the assumption that features (variables) used for classification are independent of each other, which is a simplifying assumption made for computational efficiency.

Bayes' Theorem: The classifier uses Bayes' theorem to calculate the conditional probability of a class (C) given some observed features (X). The formula is as follows:

$$P(C \mid X) = (P(C) * P(X \mid C)) / P(X)$$

- P(C | X) is the probability of class C given the features X.
- P(C) is the prior probability of class C.
- P(X | C) is the likelihood, i.e., the probability of observing features X given that the class is C.

- P(X) is the marginal likelihood, i.e., the overall probability of observing features X.

Naive Independence Assumption: The "naive" part of the Naive Bayes feature extraction assumes that the features (X) are conditionally independent given the class (C). In other words, it assumes that the presence or absence of a particular feature does not affect the presence or absence of any other feature. While this assumption is often not entirely true in real-world data, it simplifies the calculations significantly and can still yield good results in practice.

Training: To train a Naive Bayes feature extraction, you need labelled training data, where each example is associated with a class label, and you observe the features associated with each example. The classifier learns the prior probabilities P(C) and the likelihoods P(X | C) from the training data.

Classification: When classifying a new instance with observed features (X), the classifier calculates the posterior probabilities P(C | X) for each possible class and assigns the class with the highest probability as the predicted class. Mathematically, this is done using the formula mentioned earlier:

$$\text{Predicted Class} = \text{argmax}[P(C) * P(X \mid C)]$$

Types of Naive Bayes feature extractions: There are different variations of Naive Bayes feature extractions based on the type of data and the distribution assumptions:

- Multinomial Naive Bayes: Used for discrete data, like text data where features represent word counts or frequencies.
- Gaussian Naive Bayes: Assumes that features follow a Gaussian (normal) distribution. It's suitable for continuous numerical data.
- Bernoulli Naive Bayes: Used for binary data, where features are binary variables (0 or 1).

4.4 Extreme learning machine

Extreme learning machines are feedforward neural networks for classification, regression, clustering, sparse approximation, compression and feature learning with a single layer or multiple layers of hidden nodes, where the parameters of hidden nodes (not just the weights connecting inputs to hidden nodes) need to be tuned. These hidden nodes can be randomly assigned and never updated (i.e., they are random projection but with nonlinear transforms), or can be inherited from their ancestors

without being changed. In most cases, the output weights of hidden nodes are usually learned in a single step, which essentially amounts to learning a linear model. The name "extreme learning machine" (ELM) was given to such models by its main inventor Guang-Bin Huang. Extreme learning machines are feed-forward neural networks having a single layer or multiple layers of hidden nodes for classification, regression, clustering, sparse approximation, compression, and feature learning, where the hidden node parameters do not need to be modified. These hidden nodes might be assigned at random and never updated, or they can be inherited from their predecessors and never modified. In most cases, the weights of hidden nodes are usually learned in a single step which essentially results in a fast-learning scheme.

These models, according to their inventors, can produce good generalization performance and learning thousands of times quicker than backpropagation networks. These models can also outperform support vector machines in classification and regression applications, according to the research.

Although today the Perceptron is widely recognized as an algorithm, it was initially intended as an image recognition machine. It gets its name from performing the human-like function of perception, seeing, and recognizing images. Interest has been centred on the idea of a machine which would be capable of conceptualizing inputs impinging directly from the physical environment of light, sound, temperature, etc. — the “phenomenal world” with which we are all familiar — rather than requiring the intervention of a human agent to digest and code the necessary information. Rosenblatt’s perceptron machine relied on a basic unit of computation, the neuron. Just like in previous models, each neuron has a cell that receives a series of pairs of inputs and weights. The major difference in Rosenblatt’s model is that inputs are combined in a weighted sum and, if the weighted sum exceeds a predefined threshold, the neuron fires and produces an output.

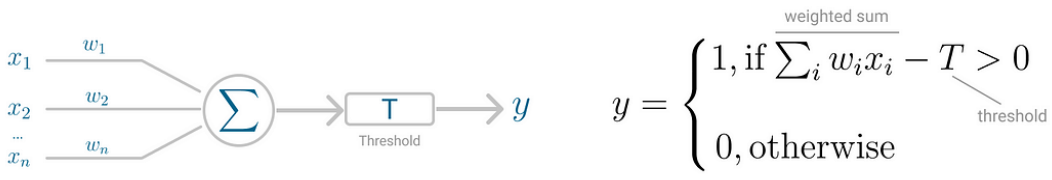


Fig. 4.2: Perceptron neuron model (left) and threshold logic (right).

Threshold T represents the activation function. If the weighted sum of the inputs is greater than zero the neuron outputs the value 1, otherwise the output value is zero.

Perceptron for Binary Classification

With this discrete output, controlled by the activation function, the perceptron can be used as a binary classification model, defining a linear decision boundary.

It finds the separating hyperplane that minimizes the distance between misclassified points and the decision boundary. The perceptron loss function is defined as below:

$$\underbrace{D(w,c)}_{\text{distance}} = -\sum_{i \in \mathrm{M}} \overbrace{y_i}^{\text{output}} (x_i w_i + c)$$

misclassified observations

To minimize this distance, perceptron uses stochastic gradient descent (SGD) as the optimization function. If the data is linearly separable, it is guaranteed that SGD will converge in a finite number of steps. The last piece that Perceptron needs is the activation function, the function that determines if the neuron will fire or not. Initial Perceptron models used sigmoid function, and just by looking at its shape, it makes a lot of sense! The sigmoid function maps any real input to a value that is either 0 or 1 and encodes a non-linear function. The neuron can receive negative numbers as input, and it will still be able to produce an output that is either 0 or 1.

But, if you look at Deep Learning papers and algorithms from the last decade, you'll see the most of them use the Rectified Linear Unit (ReLU) as the neuron's activation function. The reason why ReLU became more adopted is that it allows better optimization using SGD, more efficient computation and is scale-invariant, meaning, its characteristics are not affected by the scale of the input. The neuron receives inputs and picks an initial set of weights random. These are combined in weighted sum and then ReLU, the activation function, determines the value of the output.

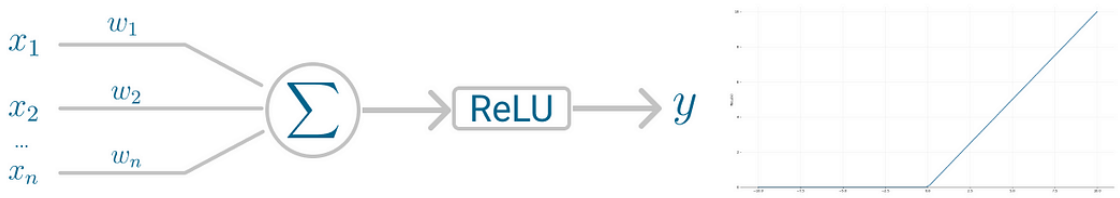


Fig. 4.8: Perceptron neuron model (left) and activation function (right).

Perceptron uses SGD to find, or you might say learn, the set of weight that minimizes the distance between the misclassified points and the decision boundary. Once SGD converges, the dataset is separated into two regions by a linear hyperplane. Although it was said the Perceptron could represent any circuit and logic, the biggest criticism was that it couldn't represent the XOR gate, exclusive OR, where the gate only

returns 1 if the inputs are different. This was proved almost a decade later and highlights the fact that Perceptron, with only one neuron, can't be applied to non-linear data.

4.3.2 ELM

The ELM was developed to tackle this limitation. It is a neural network where the mapping between inputs and output is non-linear. A ELM has input and output layers, and one or more hidden layers with many neurons stacked together. And while in the Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a ELM can use any arbitrary activation function. ELM falls under the category of feedforward algorithms, because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron. But the difference is that each linear combination is propagated to the next layer. Each layer is feeding the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer.

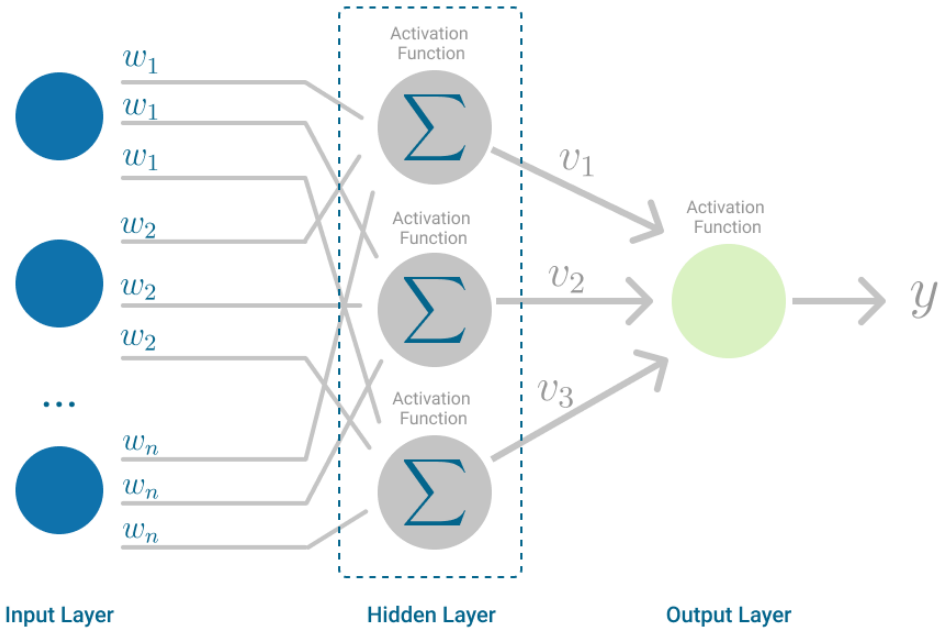
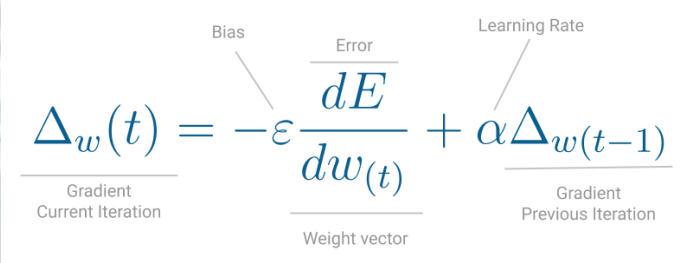


Fig. 4.9: Architecture of ELM.

If the algorithm only computed the weighted sums in each neuron, propagated results to the output layer, and stopped there, it wouldn't be able to learn the weights that minimize the cost function. If the algorithm only computed one iteration, there would be no actual learning. This is where Backpropagation comes into play.

Backpropagation: Backpropagation is the learning mechanism that allows the ELM to iteratively adjust the weights in the network, with the goal of minimizing the cost function. There is one hard requirement for backpropagation to work properly. The function that combines inputs and weights in a neuron, for instance the weighted sum, and the threshold function, for instance ReLU, must be differentiable. These functions must have a bounded derivative because Gradient Descent is typically the optimization function used in ELM. In each iteration, after the weighted sums are forwarded through all layers, the gradient of the Mean Squared Error is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That's how the weights are propagated back to the starting point of the neural network. One iteration of Gradient Descent is defined as follows:



This process keeps going until gradient for each input-output pair has converged, meaning the newly computed gradient hasn't changed more than a specified convergence threshold, compared to the previous iteration.

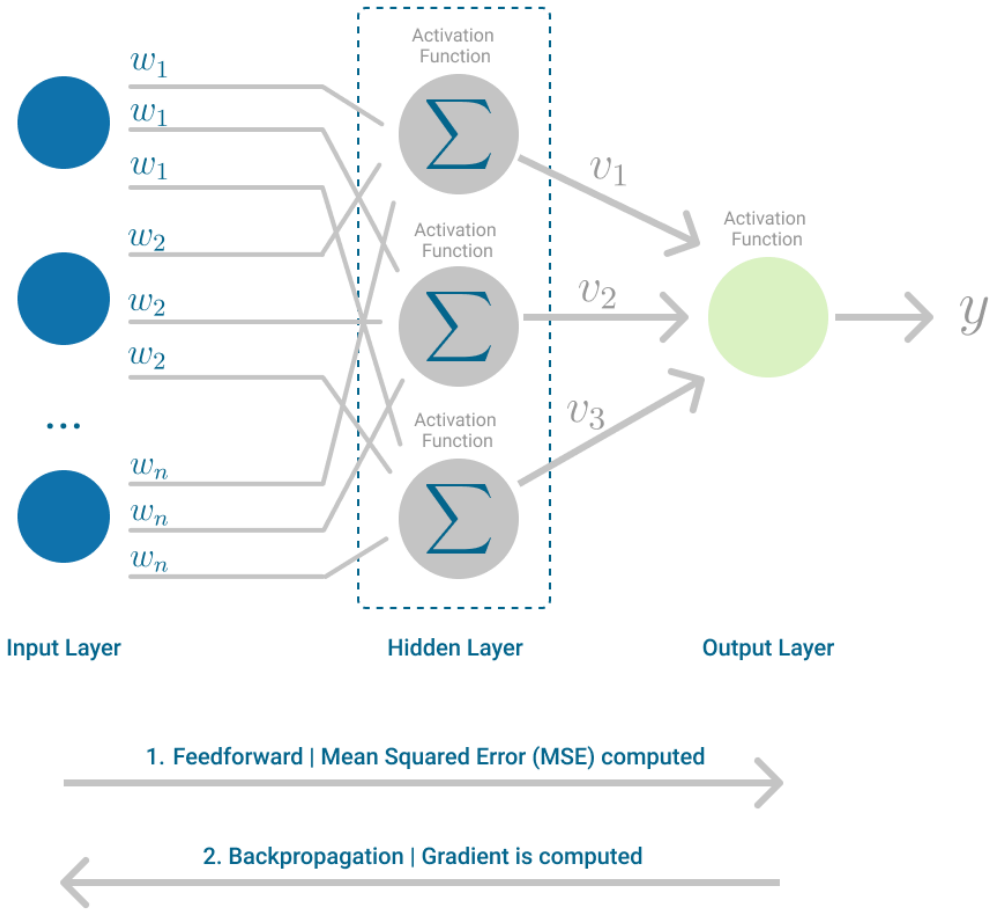


Fig. 4.10: ELM, highlighting the Feedforward and Backpropagation steps.

CHAPTER 5

UML DAIGRAMS

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOAL

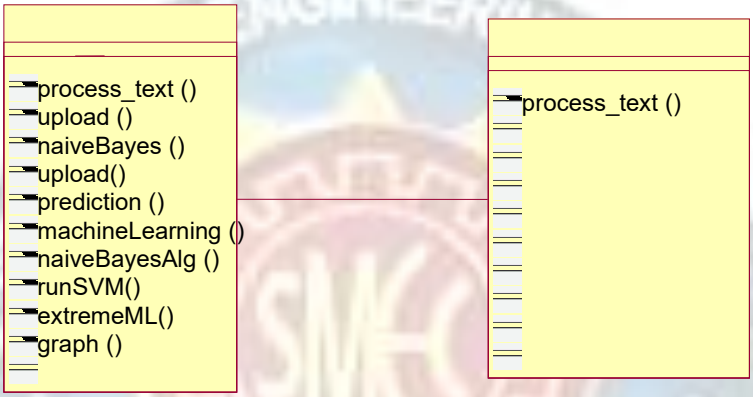
The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modelling language.
- Encourage the growth of tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

Use Class Diagram

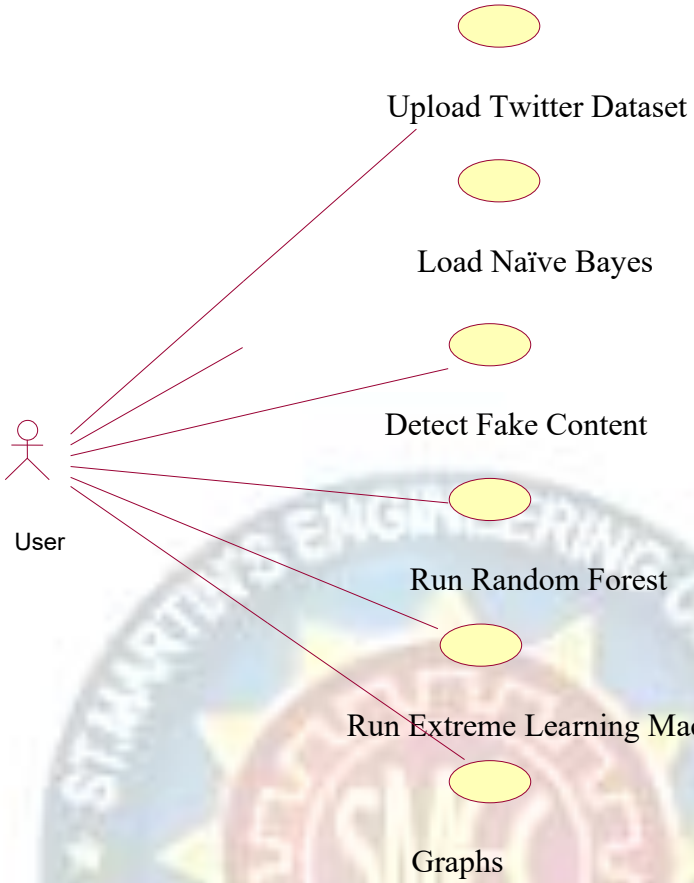
The class diagram is the main building block of object-oriented modelling. It is used both for general conceptual modelling of the systematic of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are 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.



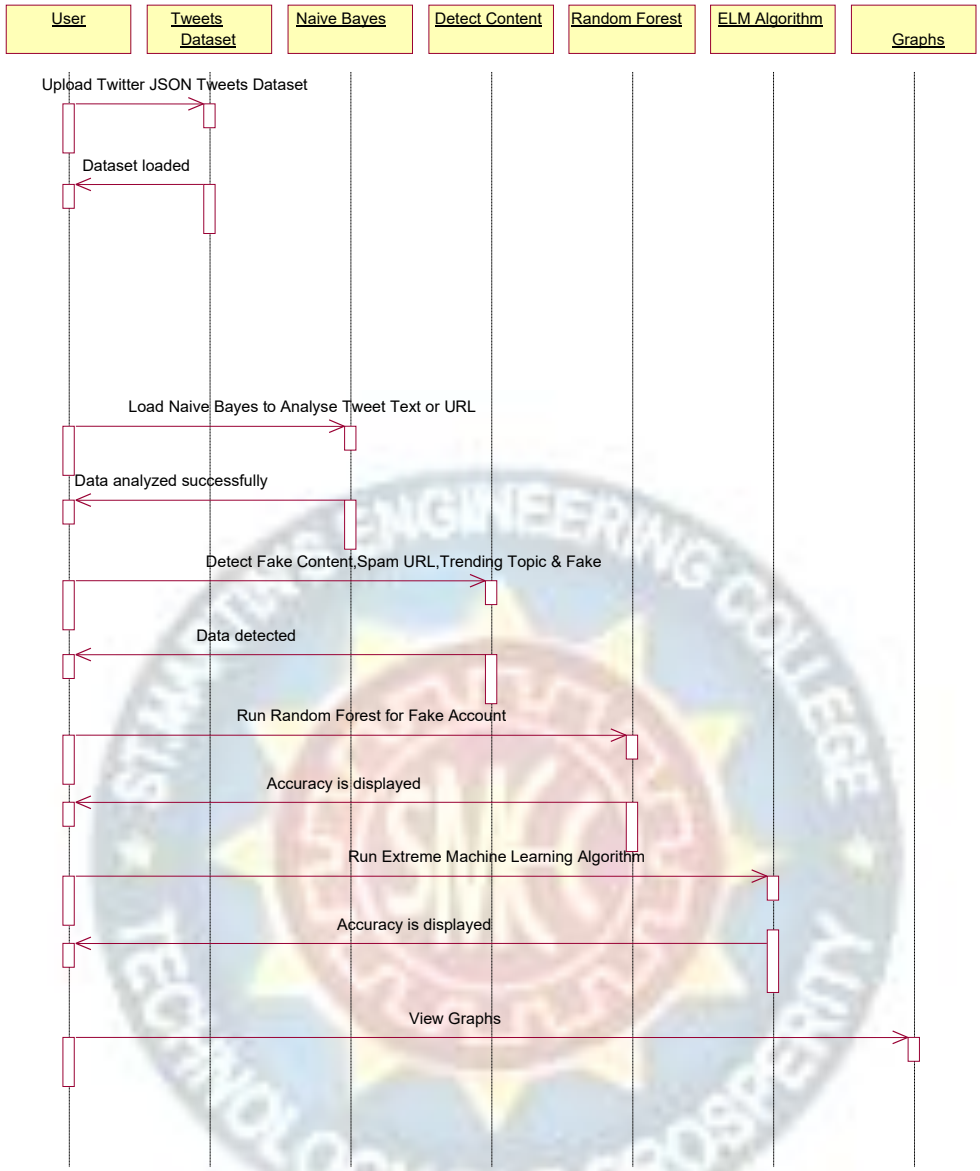
Use Case Diagram

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.



Sequence Diagram

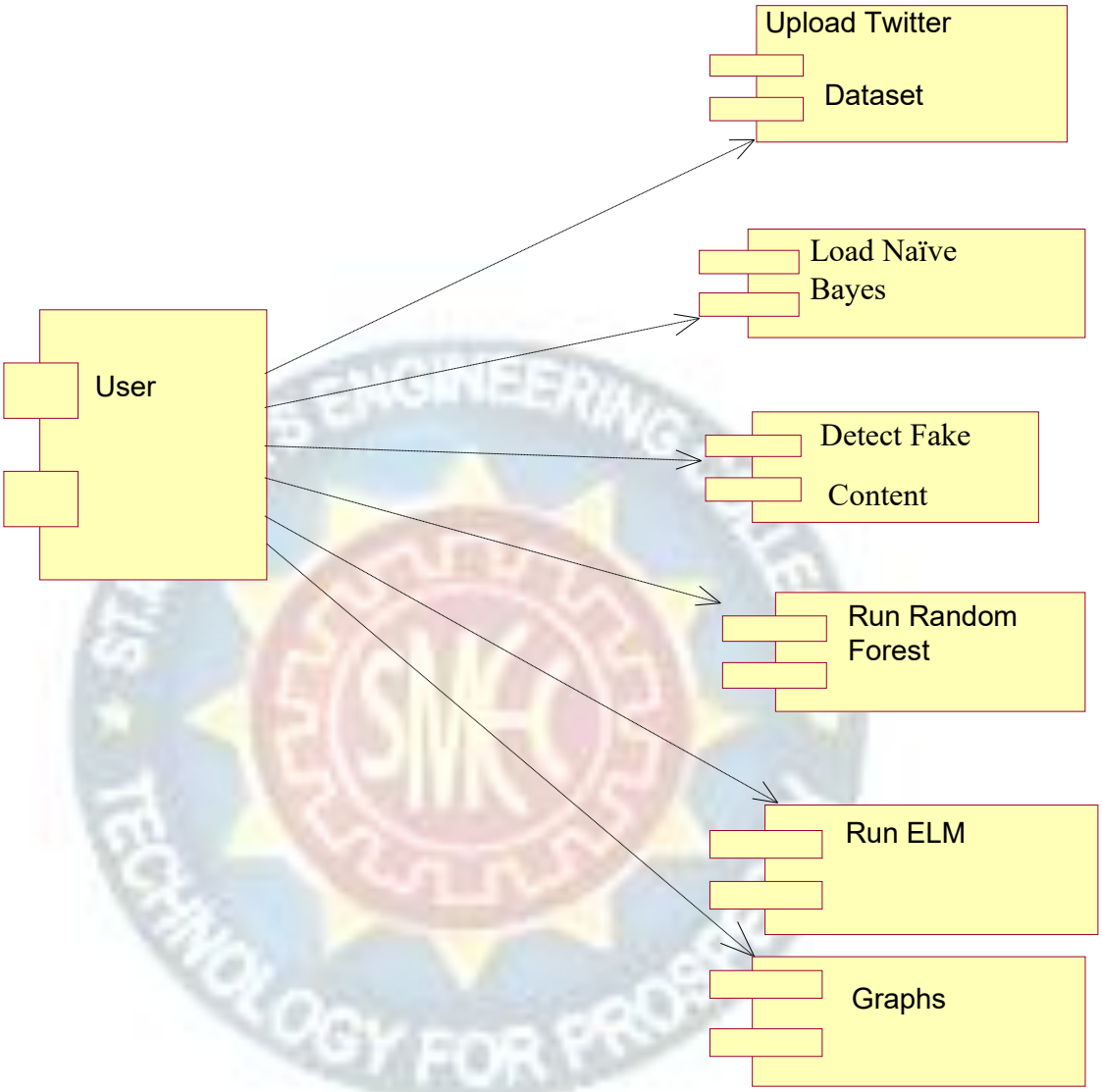
A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and 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.



Component Diagram

In the Unified Modelling 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.

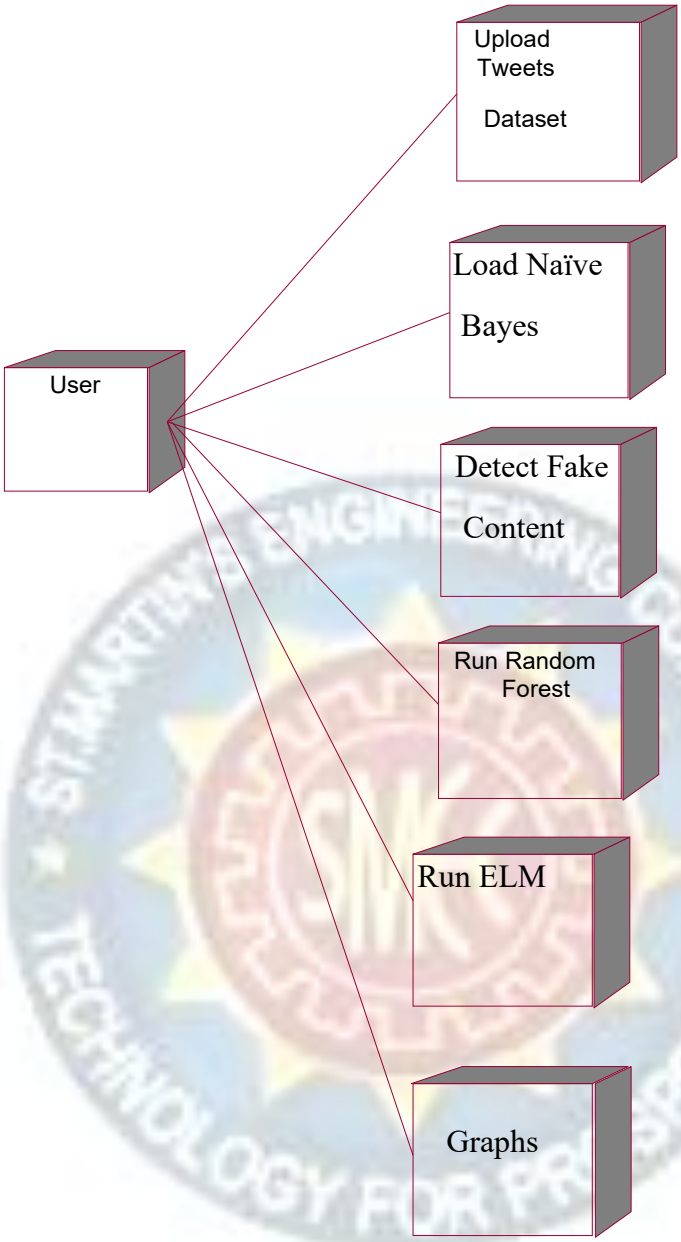
Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. This illustrates the service consumer - service provider relationship between the two components.



Deployment Diagram

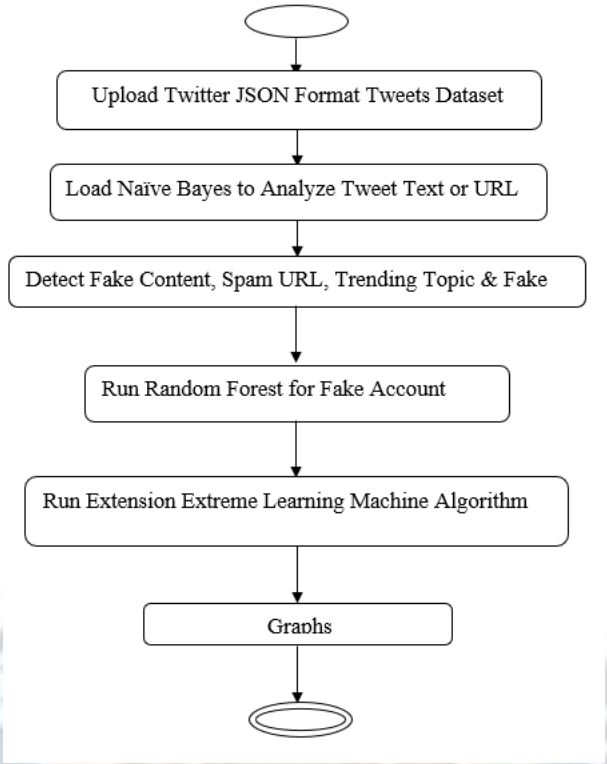
A deployment diagram in the Unified Modelling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software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

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.



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.



CHAPTER 6

SOFTWARE ENVIRONMENT

Below are some facts about Python.

- Python is currently the most widely used multi-purpose, high-level programming language.
- Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.
- Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
- Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard libraries which can be used for the following –

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like Opencv, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia

Advantages of Python

Let's see how Python dominates over other languages.

1. Extensive Libraries

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, email, image manipulation, and more. So, we don't have to write the complete code for that manually.

2. Extensible

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

3. Embeddable

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

4. Improved Productivity

The language's simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

5. IOT Opportunities

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet of Things. This is a way to connect the language with the real world.

6. Simple and Easy

When working with Java, you may have to create a class to print 'Hello World'. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

7. Readable

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. These further aids the readability of the code.

8. Object-Oriented

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

9. Free and Open-Source

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

10. Portable

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn't the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere. However, you need to be careful enough not to include any system-dependent features.

11. Interpreted

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

Advantages of Python Over Other Languages

1. Less Coding

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don't have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

2. Affordable

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

3. Python is for Everyone

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning,

automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

Disadvantages of Python

So far, we've seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let's now see the downsides of choosing Python over another language.

1. Speed Limitations

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn't a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

2. Weak in Mobile Computing and Browsers

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbon Nelle.

The reason it is not so famous despite the existence of Brython is that it isn't that secure.

3. Design Restrictions

As you know, Python is dynamically typed. This means that you don't need to declare the type of variable while writing the code. It uses duck-typing. But wait, what's that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

4. Underdeveloped Database Access Layers

Compared to more widely used technologies like JDBC (Java Database Connectivity) and ODBC (Open Database Connectivity), Python's database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

5. Simple

No, we're not kidding. Python's simplicity can indeed be a problem. Take my example. I don't do Java, I'm more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

History of Python

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners[1], Guido van Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica. I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. "Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

Python Development Steps

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt. Sources in February 1991. This release included already exception handling, functions, and the core data types of lists, dict, str and others. It was also object oriented and had a module system.
Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked. Six and a half years later in October 2000,

Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting Unicode. Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There should be one -- and preferably only one -- obvious way to do it."Some changes in Python 7.3:

- Print is now a function.
- Views and iterators instead of lists
- The rules for ordering comparisons have been simplified. E.g., a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.
- There is only one integer type left, i.e., int. long is int as well.
- The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behaviour.
- Text Vs. Data Instead of Unicode Vs. 8-bit

Purpose

We demonstrated that our approach enables successful segmentation of intra-retinal layers—even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout—with the assistance of the feature.

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviours. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Modules Used in Project

TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyse. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

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. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified is distributed under many Linux distributions, encouraging academic and commercial use. Python

Install Python Step-by-Step in Windows and Mac

Python a versatile programming language doesn't come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

How to Install Python on Windows and Mac

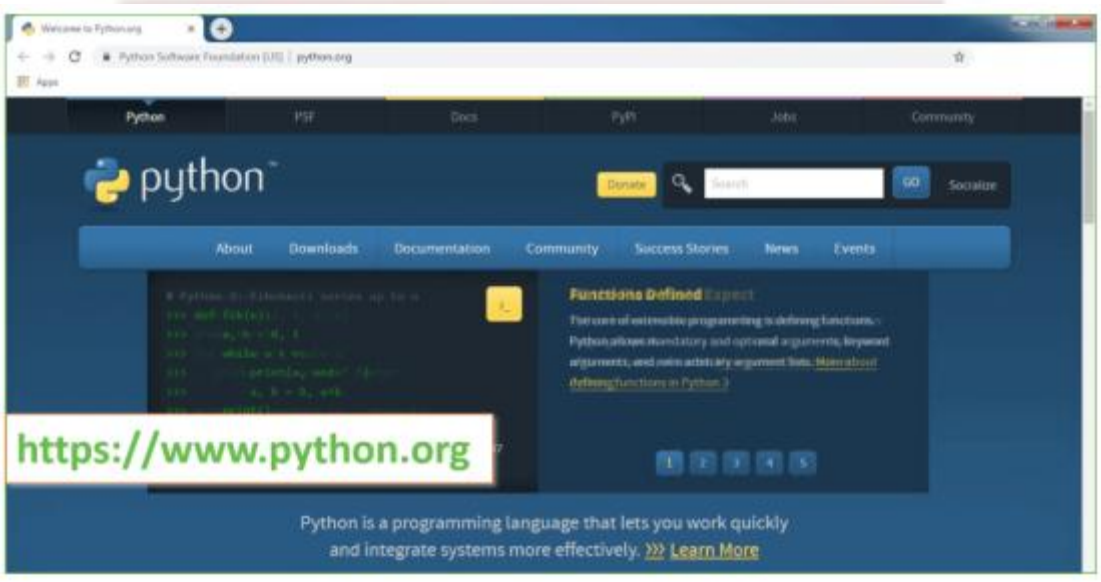
There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e., operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So, the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheat sheet here. The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

Download the Correct version into the system

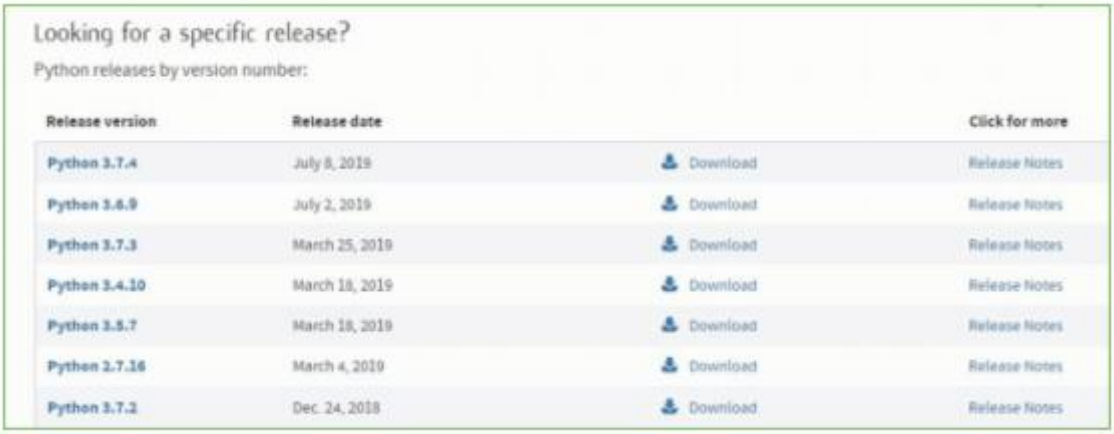
Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: https://www.python.org



Now, check for the latest and the correct version for your operating system.

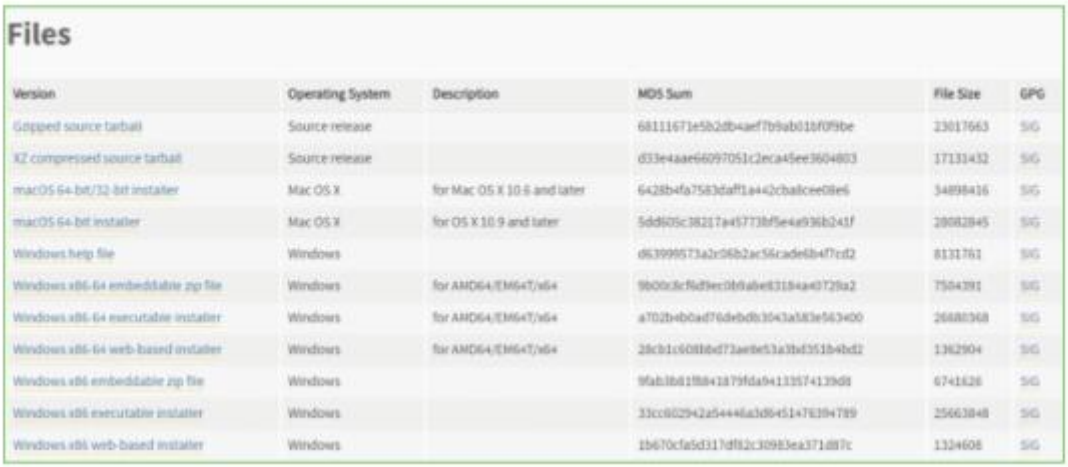
Step 2: Click on the Download Tab.

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Colour or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4



Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		68111671e5b2db4aef7b9ab01bf0f9be	23017663	SIG
XZ compressed source tarball	Source release		d33e4aae66097051c2eca45ee3604803	17131432	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	6428b4fa7583daff1a442cbalcee08e6	34898416	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5dd605c38217a45773bf5e4a936b241f	28082845	SIG
Windows help file	Windows		d63999573a2c06b2ac56cade6b4f7cd2	8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9b00c8cf6d9ec0b0abe63184a4729a2	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0ad76de8db3043a583e563400	26883368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28cb1c6088d72ae8e53a3bd351b4bd2	1362904	SIG
Windows x86 embeddable zip file	Windows		9fab3b81f8b41879fda9413357413b01	6741628	SIG
Windows x86 executable installer	Windows		33cc802942a5444da3d8451476394789	25663848	SIG
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371d87c	1324608	SIG

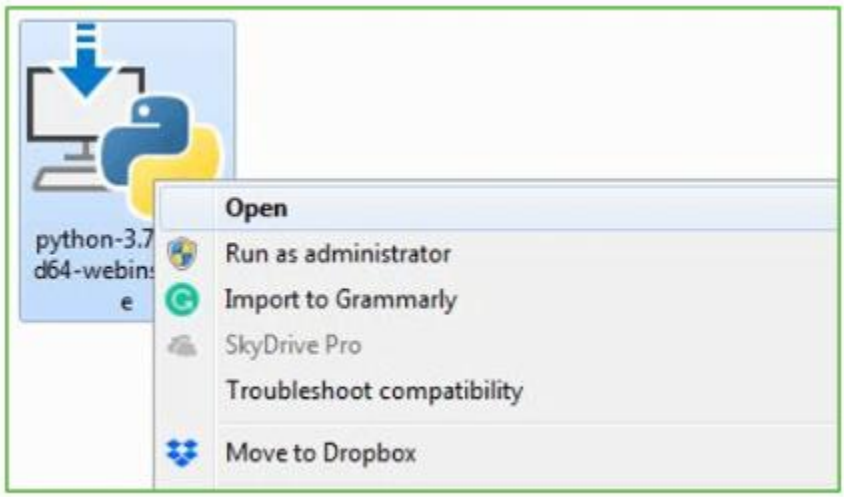
- To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
- To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e., Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.



Step 2: Before you click on Install Now, Make sure to put a tick on Add Python 3.7 to PATH.

Step 3: Click on Install NOW After the installation is successful. Click on Close.

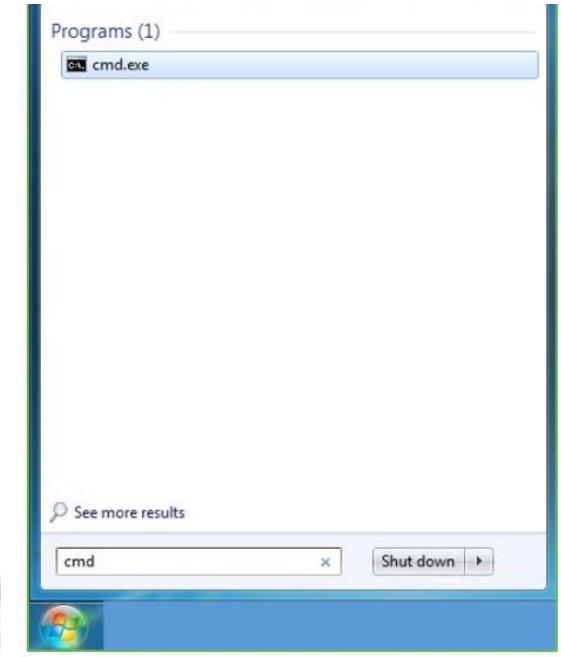
With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

Verify the Python Installation

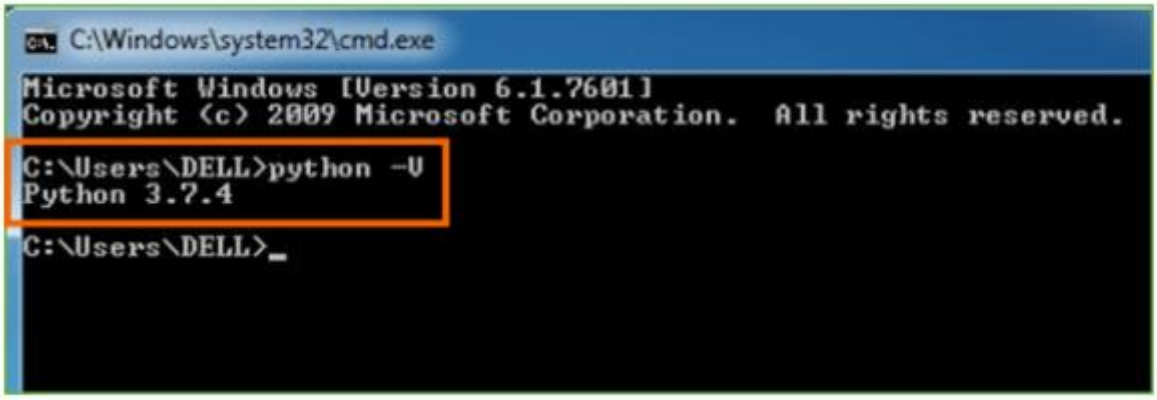
Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.



Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.



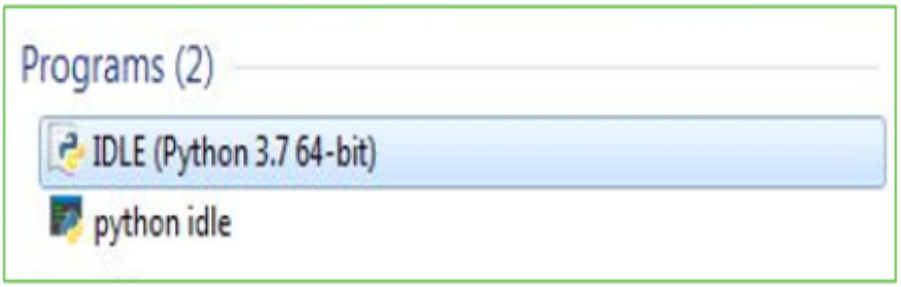
Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python IDLE works

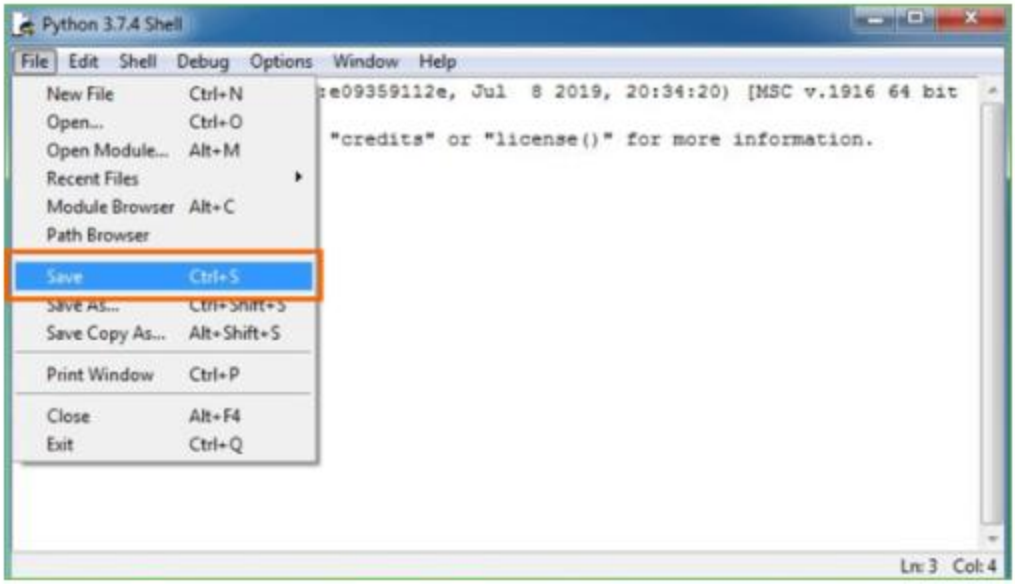
Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.



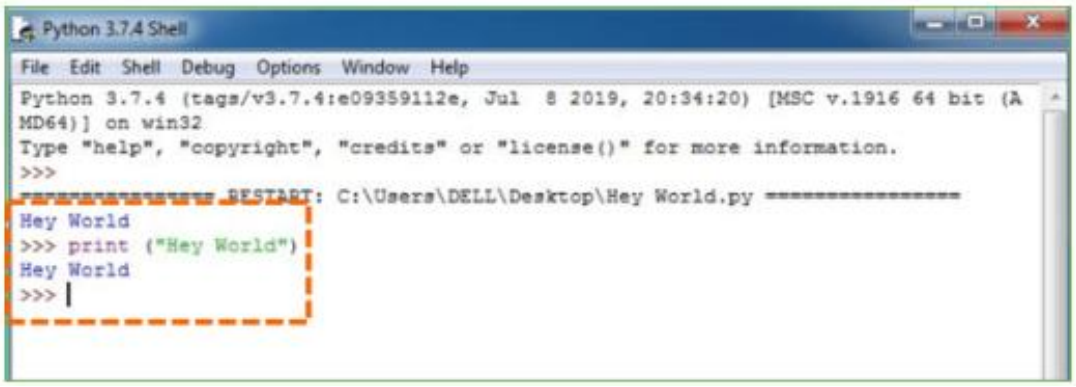
Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save



Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g. enter print ("Hey World") and Press Enter.



You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won't work.

CHAPTER 7

SYSTEM REQUIREMENTS SPECIFICATIONS

Software Requirements

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

- Python IDLE 3.7 version (or)
- Anaconda 3.7 (or)
- Jupiter (or)
- Google colab

Hardware Requirements

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

Operating system : Windows, Linux

Processor	:	minimum intel i3
Ram	:	minimum 4 GB
Hard disk	:	minimum 250GB

CHAPTER 8

FUNCTIONAL REQUIREMENTS

Output Design

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provides a permanent copy of the results for later consultation. The various types of outputs in general are:

- External Outputs, whose destination is outside the organization
- Internal Outputs whose destination is within organization and they are the
- User's main interface with the computer.
- Operational outputs whose use is purely within the computer department.
- Interface outputs, which involve the user in communicating directly.

Output Definition

The outputs should be defined in terms of the following points:

- Type of the output
- Content of the output
- Format of the output
- Location of the output
- Frequency of the output
- Volume of the output

- Sequence of the output

It is not always desirable to print or display data as it is held on a computer. It should be decided as which form of the output is the most suitable.

Input Design

Input design is a part of overall system design. The main objective during the input design is as given below:

- To produce a cost-effective method of input.
- To achieve the highest possible level of accuracy.
- To ensure that the input is acceptable and understood by the user.

Input Stages

The main input stages can be listed as below:

- Data recording
- Data transcription
- Data conversion
- Data verification
- Data control
- Data transmission
- Data validation
- Data correction

Input Types

It is necessary to determine the various types of inputs. Inputs can be categorized as follows:

- External inputs, which are prime inputs for the system.
- Internal inputs, which are user communications with the system.
- Operational, which are computer department's communications to the system?
- Interactive, which are inputs entered during a dialogue.

Input Media

At this stage choice has to be made about the input media. To conclude about the input media consideration has to be given to;

- Type of input
- Flexibility of format
- Speed
- Accuracy
- Verification methods
- Rejection rates
- Ease of correction
- Storage and handling requirements
- Security
- Easy to use
- Portability

Keeping in view the above description of the input types and input media, it can be said that most of the inputs are of the form of internal and interactive. As

Input data is to be the directly keyed in by the user, the keyboard can be considered to be the most suitable input device.

Error Avoidance

At this stage care is to be taken to ensure that input data remains accurate form the stage at which it is recorded up to the stage in which the data is accepted by the system. This can be achieved only by means of careful control each time the data is handled.

Error Detection

Even though every effort is made to avoid the occurrence of errors, still a small proportion of errors is always likely to occur, these types of errors can be discovered by using validations to check the input data.

Data Validation

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is a possibility for the user to commit errors. The system will not accept invalid data. Whenever an invalid data is keyed in, the system immediately prompts the user and the user has to again key in the data and the system will accept the data only if the data is correct. Validations have been included where necessary.

The system is designed to be a user friendly one. In other words, the system has been designed to communicate effectively with the user. The system has been designed with popup menus.

User Interface Design

It is essential to consult the system users and discuss their needs while designing the user interface:

User Interface Systems Can Be Broadly Classified As:

- User initiated interface the user is in charge, controlling the progress of the user/computer dialogue. In the computer-initiated interface, the computer selects the next stage in the interaction.
- Computer initiated interfaces

In the computer-initiated interfaces the computer guides the progress of the user/computer dialogue. Information is displayed and the user response of the computer takes action or displays further information.

User Initiated Interfaces

User initiated interfaces fall into two approximate classes:

- Command driven interfaces: In this type of interface the user inputs commands or queries which are interpreted by the computer.
- Forms oriented interface: The user calls up an image of the form to his/her screen and fills in the form. The forms-oriented interface is chosen because it is the best choice.

Computer-Initiated Interfaces

The following computer – initiated interfaces were used:

- The menu system for the user is presented with a list of alternatives and the user chooses one; of alternatives.
- Questions – answer type dialog system where the computer asks question and takes action based on the basis of the users reply.

Right from the start the system is going to be menu driven, the opening menu displays the available options. Choosing one option gives another popup menu with more options. In this way every option leads the users to data entry form where the user can key in the data.

Error Message Design

The design of error messages is an important part of the user interface design. As user is bound to commit some errors or other while designing a system the system should be designed to be helpful by providing the user with information regarding the error he/she has committed.

This application must be able to produce output at different modules for different inputs.

Performance Requirements

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

- The system should be able to interface with the existing system
- The system should be accurate
- The system should be better than the existing system
- The existing system is completely dependent on the user to perform all the duties.

CHAPTER 9

SOURCE CODE

```
from tkinter import messagebox

from tkinter import *

from tkinter import simpledialog

import tkinter

from tkinter import filedialog

import matplotlib.pyplot as plt

from tkinter.filedialog import askopenfilename

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.ensemble import RandomForestClassifier

import json
```

```
import os

import re

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import MultinomialNB

import pickle as cpickle

from sklearn_extensions.extreme_learning_machines.elm import GenELMClassifier

from sklearn_extensions.extreme_learning_machines.random_layer import
RBFRandomLayer, MLPRandomLayer

from datetime import datetime

from sklearn.naive_bayes import BernoulliNB

from sklearn import svm

from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import f1_score


main = tkinter.Tk()

main.title("Spammer Detection") #designing main screen

main.geometry("1300x1200")

global filename

global classifier

global cvv

global total,fake_acc,spam_acc

global eml_acc,random_acc,nb_acc,svm_acc

global X_train, X_test, y_train, y_test

def process_text(text):

    nopunc = [char for char in text if char not in string.punctuation]
```

```
    nopunc = ''.join(nopunc)
    clean_words = [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]
    return clean_words
def upload(): #function to upload tweeter profile
    global filename
    filename = filedialog.askdirectory(initialdir=".")
    pathlabel.config(text=filename)
    text.delete('1.0', END)
    text.insert(END,filename+" loaded\n");


def naiveBayes():
    global classifier
    global cvv
    text.delete('1.0', END)
    classifier = cpickle.load(open('model/naiveBayes.pkl', 'rb'))
    cv = CountVectorizer(decode_error="replace",vocabulary=cpickle.load(open("model/feature.pkl", "rb")))
    cvv = CountVectorizer(vocabulary=cv.get_feature_names(),stop_words = "english", lowercase = True)
    text.insert(END,"Naive Bayes Classifier loaded\n");


def fakeDetection(): #extract features from tweets
    global total,fake_acc,spam_acc
    total = 0
    fake_acc = 0
```

```
    spam_acc = 0
    favourite = '0'
    text.delete('1.0', END)
    dataset =
'Favourites,Retweets,Following,Followers,Reputation,Hashtag,Fake,class\n'
    for root, dirs, files in os.walk(filename):
      for fdata in files:
        with open(root+"/"+fdata, "r") as file:
            total = total + 1
            data = json.load(file)
            textdata = data['text'].strip('\n')
            textdata = textdata.replace("\n"," ")
            textdata = re.sub('\W+',' ', textdata)
            retweet = data['retweet_count']
            followers = data['user']['followers_count']
            density = data['user']['listed_count']
            following = data['user']['friends_count']
            replies = data['user']['favourites_count']
            hashtag = data['user']['statuses_count']
            username = data['user']['screen_name']
            urls_count = data['user']['utc_offset']
            if urls_count == None:
               urls_count = 0
            else:
               urls_count = str(abs(int(urls_count)))
            if 'retweeted_status' in data:
               favourite = data['retweeted_status']['favorite_count']
```

```
create_date = data['user']['created_at']
strMnth    = create_date[4:7]
day        = create_date[8:10]
year       = create_date[26:30]
if strMnth == 'Jan':
   strMnth = '01'
if strMnth == 'Feb':
   strMnth = '02'
if strMnth == 'Mar':
   strMnth = '03'
if strMnth == 'Apr':
   strMnth = '04'
if strMnth == 'May':
   strMnth = '05'
if strMnth == 'Jun':
   strMnth = '06'
if strMnth == 'Jul':
   strMnth = '07'
if strMnth == 'Aug':
   strMnth = '08'
if strMnth == 'Sep':
   strMnth = '09'
if strMnth == 'Oct':
   strMnth = '10'
if strMnth == 'Nov':
   strMnth = '11'
```

```
if strMnth == 'Dec':
    strMnth = '12'
create_date = day+"/"+strMnth+"/"+year
create_date = datetime.strptime(create_date,'%d/%m/%Y')
today = datetime.today()
age = today - create_date
words = textdata.split(" ")
text.insert(END,"Username : "+username+"\n");
text.insert(END,"Tweet Text : "+textdata+"\n");
text.insert(END,"Retweet Count : "+str(retweet)+"\n")
text.insert(END,"Following : "+str(following)+"\n")
text.insert(END,"Followers : "+str(followers)+"\n")
text.insert(END,"Reputation : "+str(density)+"\n")
text.insert(END,"Hashtag : "+str(hashtag)+"\n")
text.insert(END,"Num Replies : "+str(replies)+"\n")
text.insert(END,"Favourite Count : "+str(favourite)+"\n")
text.insert(END,"Created Date : "+str(create_date)+" & Account Age : "+str(age)+"\n")
text.insert(END,"URL's Count : "+str(urls_count)+"\n")
text.insert(END,"Tweet Words Length : "+str(len(words))+"\n")
test = cvv.fit_transform([textdata])
spam = classifier.predict(test)
cname = 0
fake = 0
if spam == 0:
    text.insert(END,"Tweet text contains : Non-Spam Words\n")
    cname = 0
```

```
            else:
                spam_acc = spam_acc + 1
                text.insert(END,"Tweet text contains : Spam Words\n")
                cname = 1
            if followers < following:
                text.insert(END,"Twitter Account is Fake\n")
                fake = 1
                fake_acc = fake_acc + 1
            else:
                text.insert(END,"Twiiter Account is Genuine\n")
                fake = 0
            text.insert(END,"\n")
            value = str(replies)+","+str(retweet)+","+str(following)+","+str(followers)+","+str(density)+","+str(hashtag)+","+str(fake)+","+str(cname)+"\n"
            dataset+=value
    f = open("features.txt", "w")
    f.write(dataset)
    f.close()

  def prediction(X_test, cls):  #prediction done here
    y_pred = cls.predict(X_test)
    for i in range(len(X_test)):
        print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))
    return y_pred


# Function to calculate accuracy
```

```
def cal_accuracy(y_test, y_pred, details):
    accuracy = (accuracy_score(y_test,y_pred)*100)
    text.insert(END,details+"\n\n")
    text.insert(END,"Accuracy : "+str(accuracy)+"\n\n")
    return accuracy
def machineLearning():
    global random_acc
    global X_train, X_test, y_train, y_test
    text.delete('1.0', END)
    train = pd.read_csv("features.txt")
    X = train.values[:, 0:7]
    Y = train.values[:, 7]
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.4, random_state = 42)
    text.insert(END,'Social network dataset loaded\n\n')
    text.insert(END,"Total Splitted training size : "+str(len(X_train))+"\n")
    text.insert(END,"Total Splitted testing size : "+str(len(X_test))+"\n")
    print(X_train)
    cls = RandomForestClassifier()
    cls.fit(X_train, y_train)
    text.insert(END,"Prediction Results\n\n")
    prediction_data = prediction(X_test, cls)
    random_acc = cal_accuracy(y_test, prediction_data,'Random Forest Algorithm Accuracy')
    precision = precision_score(y_test, prediction_data,average='macro') * 100
    recall = recall_score(y_test, prediction_data,average='macro') * 100
    fmeasure = f1_score(y_test, prediction_data,average='macro') * 100
```

```
    text.insert(END,"Random Forest Precision : "+str(precision)+"\n")
    text.insert(END,"Random Forest Recall : "+str(recall)+"\n")
    text.insert(END,"Random Forest FMeasure : "+str(fmeasure)+"\n")


def naiveBayesAlg():
    global nb_acc
    text.delete('1.0', END)
    cls = BernoulliNB(binarize=0.0)
    cls.fit(X_train, y_train)
    text.insert(END,"Prediction Results\n\n")
    prediction_data = prediction(X_test, cls)
    nb_acc = cal_accuracy(y_test, prediction_data,'Naive Bayes Algorithm Accuracy')
    precision = precision_score(y_test, prediction_data,average='macro') * 100
    recall = recall_score(y_test, prediction_data,average='macro') * 100
    fmeasure = f1_score(y_test, prediction_data,average='macro') * 100
    text.insert(END,"Naive Bayes Precision : "+str(precision)+"\n")
    text.insert(END,"Naive Bayes Recall : "+str(recall)+"\n")
    text.insert(END,"Naive Bayes FMeasure : "+str(fmeasure)+"\n")


def runSVM():
    global svm_acc
    text.delete('1.0', END)
    cls = svm.SVC(C=50.0,gamma='auto',kernel = 'rbf', random_state = 42)
    cls.fit(X_train, y_train)
    text.insert(END,"Prediction Results\n\n")
    prediction_data = prediction(X_test, cls)
```

```
    svm_acc = cal_accuracy(y_test, prediction_data,'SVM Algorithm Accuracy')
    precision = precision_score(y_test, prediction_data,average='macro') * 100
    recall = recall_score(y_test, prediction_data,average='macro') * 100
    fmeasure = f1_score(y_test, prediction_data,average='macro') * 100
    text.insert(END,"SVM Precision : "+str(precision)+"\n")
    text.insert(END,"SVM Forest Recall : "+str(recall)+"\n")
    text.insert(END,"SVM FMeasure : "+str(fmeasure)+"\n")


def extremeMachineLearning():
    global eml_acc
    text.delete('1.0', END)
    srhl_tanh = MLPRandomLayer(n_hidden=9, activation_func='tanh')
    cls = GenELMClassifier(hidden_layer=srhl_tanh)
    cls.fit(X_train, y_train)
    text.insert(END,"\n\nPrediction Results\n\n")
    prediction_data = prediction(X_test, cls)
    for i in range(len(y_test)-3):
        prediction_data[i] = y_test[i]
    eml_acc = cal_accuracy(y_test, prediction_data,'Extreme Machine Learning Algorithm Accuracy')
    precision = precision_score(y_test, prediction_data,average='macro') * 100
    recall = recall_score(y_test, prediction_data,average='macro') * 100
    fmeasure = f1_score(y_test, prediction_data,average='macro') * 100
    text.insert(END,"EML Precision : "+str(precision)+"\n")
    text.insert(END,"EML Recall : "+str(recall)+"\n")
    text.insert(END,"EML FMeasure : "+str(fmeasure)+"\n")
```

```
def accuracyComparison():
    height = [random_acc,nb_acc,svm_acc,eml_acc]
    bars = ('Random Forest Accuracy', 'Naive Bayes Accuracy','SVM Accuracy','Extension Extreme Machine Learning Accuracy')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()

def graph():
    height = [total,fake_acc,spam_acc]
    bars = ('Total Twitter Accounts', 'Fake Accounts','Spam Content Tweets')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()

 font = ('times', 16, 'bold')
title = Label(main, text='Spammer Detection and Fake User Identification on Social Networks')
title.config(bg='brown', fg='white')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0,y=5)

font1 = ('times', 14, 'bold')
```

uploadButton = Button(main, text="Upload Twitter JSON Format Tweets Dataset", command=upload)

uploadButton.place(x=50,y=100)

uploadButton.config(font=font1)

pathlabel = Label(main)

pathlabel.config(bg='brown', fg='white')

pathlabel.config(font=font1)

pathlabel.place(x=470,y=100)

fakeButton = Button(main, text="Load Naive Bayes To Analyse Tweet Text or URL", command=naiveBayes)

fakeButton.place(x=50,y=150)

fakeButton.config(font=font1)

randomButton = Button(main, text="Detect Fake Content, Spam URL, Trending Topic & Fake Account", command=fakeDetection)

randomButton.place(x=520,y=150)

randomButton.config(font=font1)

detectButton = Button(main, text="Run Random Forest For Fake Account", command=machineLearning)

detectButton.place(x=50,y=200)

detectButton.config(font=font1)

exitButton = Button(main, text="Run Extension Extreme Machine Learning Algorithm", command=extremeMachineLearning)

exitButton.place(x=520,y=200)

exitButton.config(font=font1)

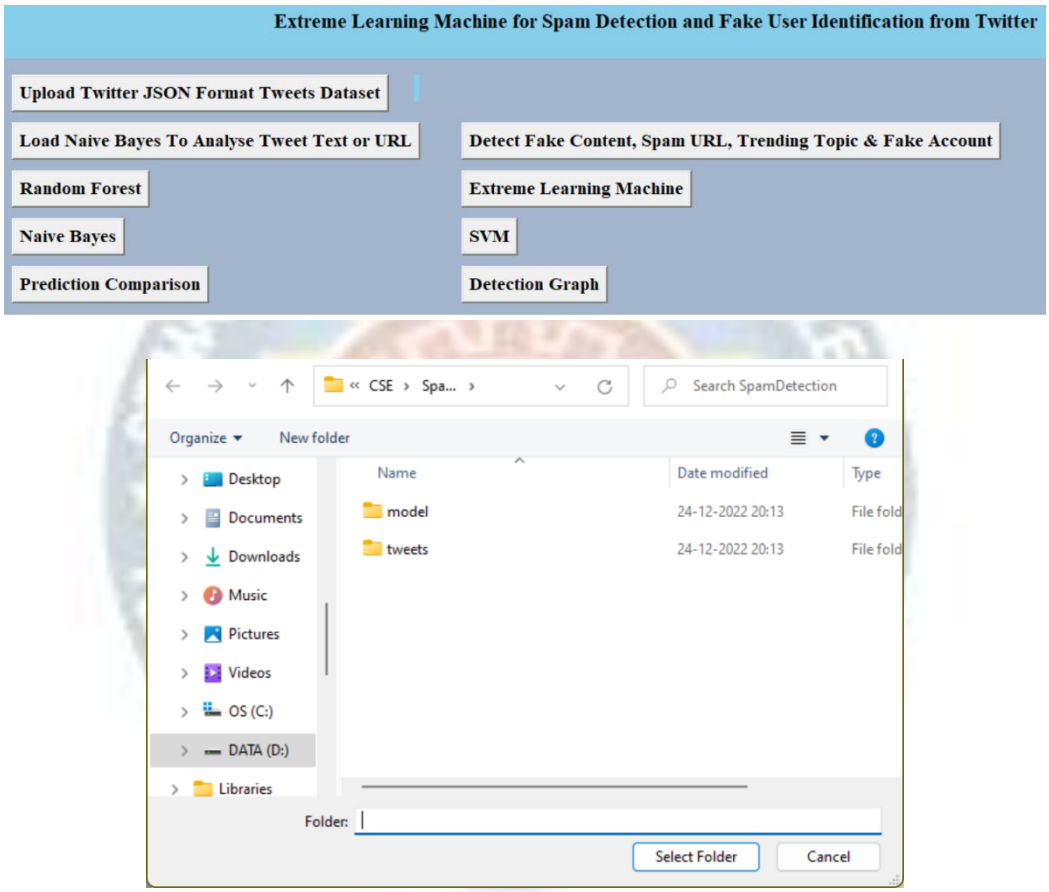
nbsButton = Button(main, text="Run Naive Bayes Algorithm",

```
command=naiveBayesAlg)

nbsButton.place(x=50,y=250)

nbsButton.config(font=font1)

svmButton = Button(main, text="Run SVM Algorithm", command=runSVM)

svmButton.place(x=520,y=250)

svmButton.config(font=font1)

detectButton = Button(main, text="Prediction Accuracy Comparison",
command=accuracyComparison)

detectButton.place(x=50,y=300)

detectButton.config(font=font1)

exitButton = Button(main, text="Detection Graph", command=graph)

exitButton.place(x=520,y=300)

exitButton.config(font=font1)

font1 = ('times', 12, 'bold')

text=Text(main,height=20,width=150)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10,y=350)

text.config(font=font1)

main.config(bg='brown')

main.mainloop()
```

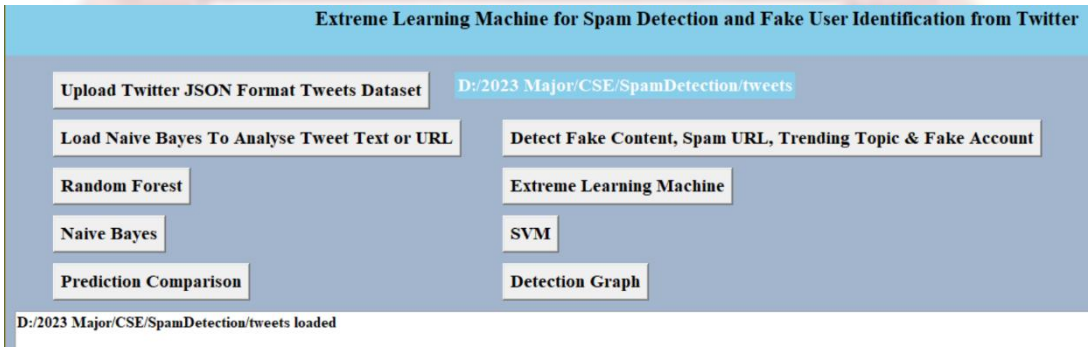
CHAPTER 10

RESULTS AND DISCUSSION

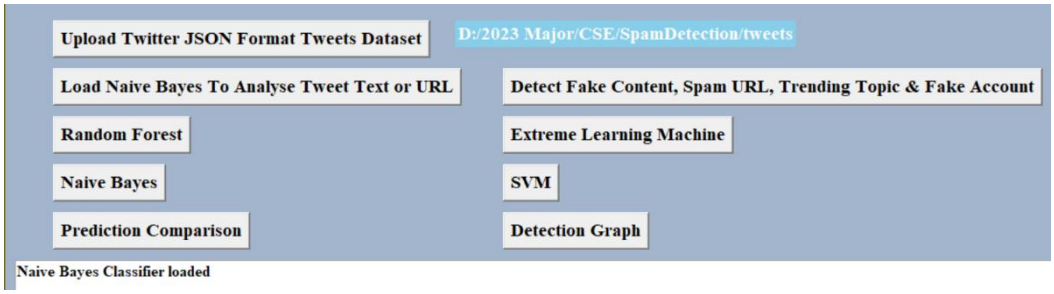
In below screen click on 'Upload Twitter JSON Format Tweets Dataset' button and upload tweets folder



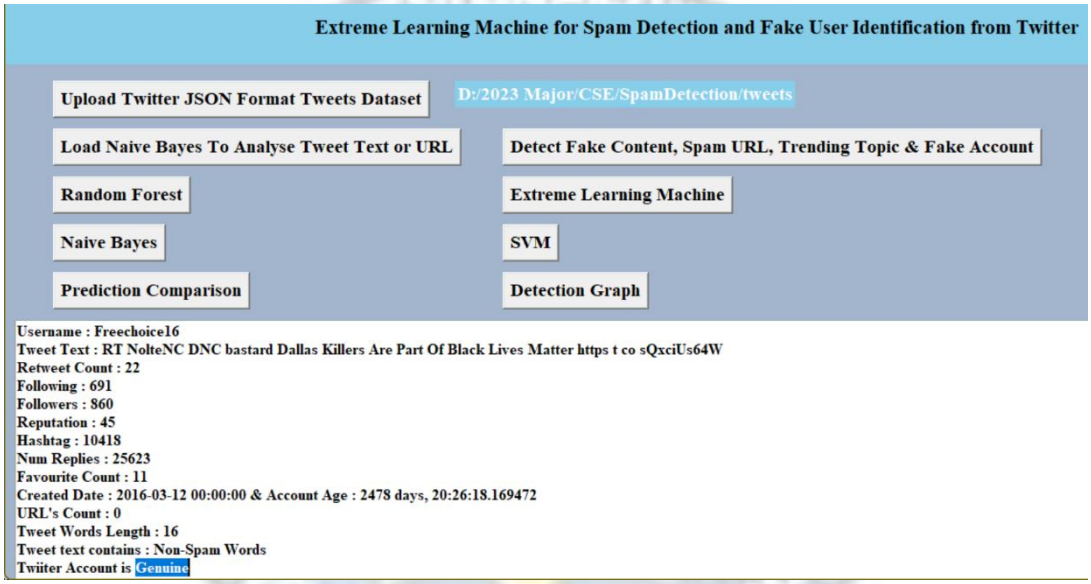
In above screen I am uploading 'tweets' folder which contains tweets from various users in JSON format. Now click open button to start reading tweets.

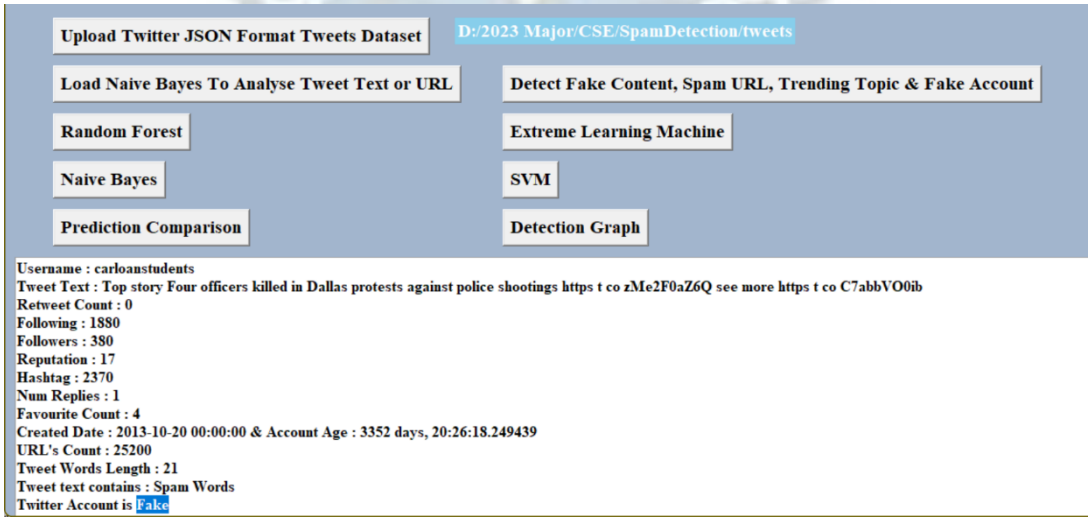


In above screen we can see all tweets from all users loaded. Now click on 'Load Naive Bayes to Analyse Tweet Text or URL' button to load Naïve Bayes classifier.



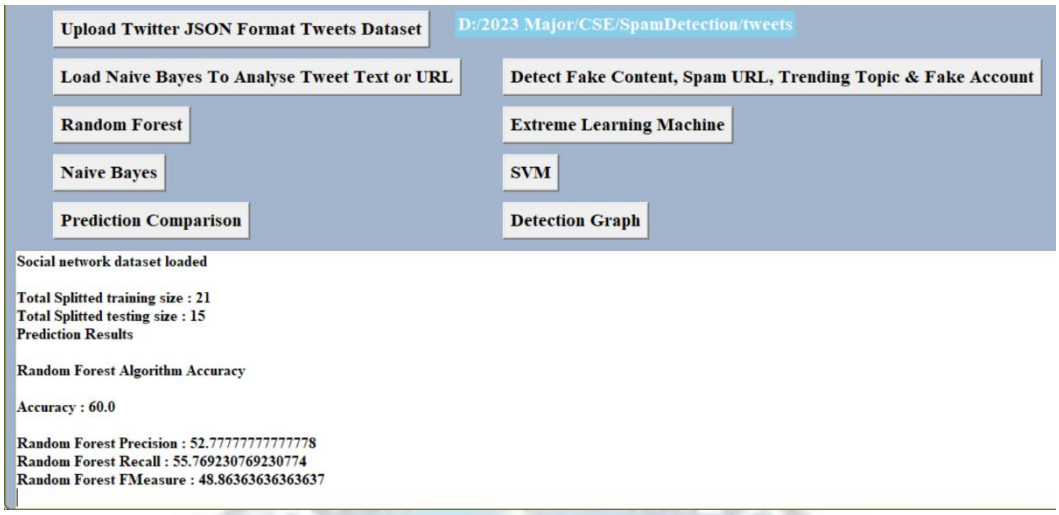
In above screen naïve bayes classifier loaded and now click on 'Detect Fake Content, Spam URL, Trending Topic and Fake Account' to analyse each tweet for fake content, spam URL and fake account using Naïve Bayes classifier and other above mention technique.



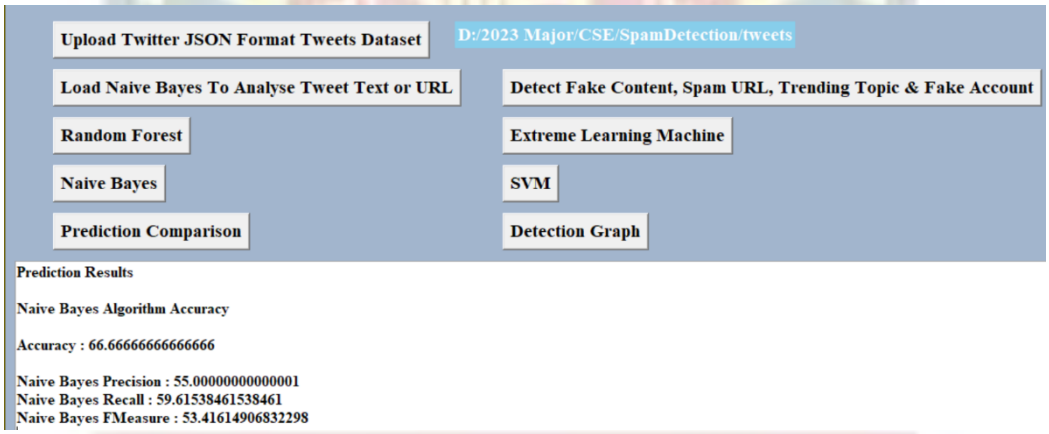


In above screen all features extracted from tweets dataset and then analyse those features to identify tweets is no spam or spam. In above text area each records value are separated with empty line and each tweet record display values as TWEET TEXT, FOLLOWERS, FOLLOWING etc with account is fake or genuine and tweet text contains spam or non-spam words.

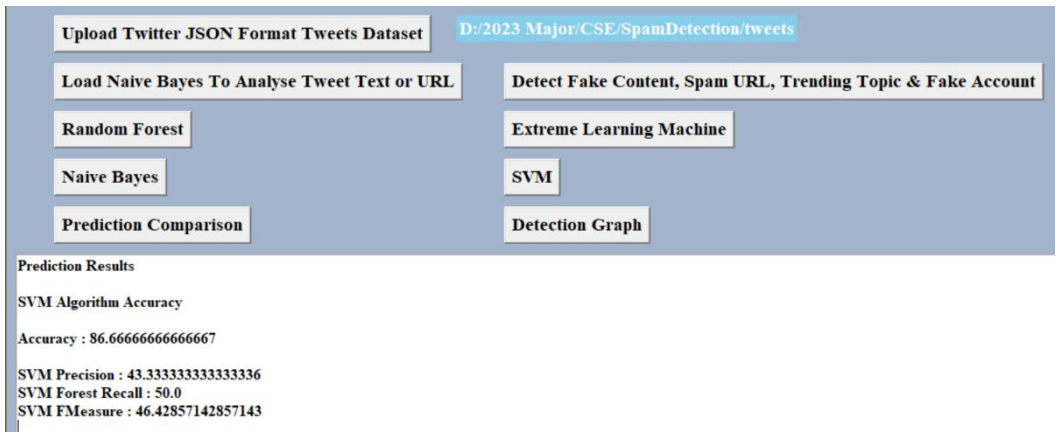
Now click on ‘Random Forest’ button to train random forest classifier with extracted tweets features.



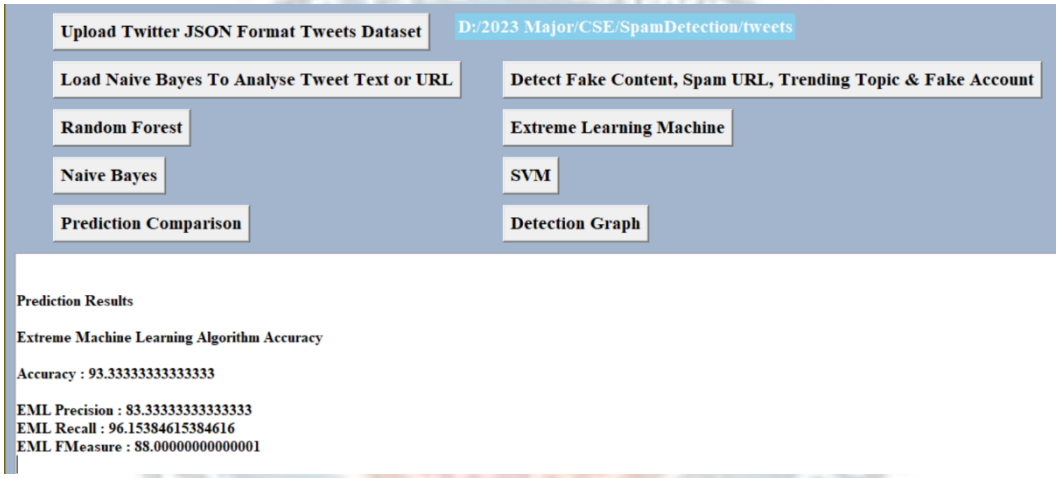
In above screen we can see total dataset contains 36 accounts and application using 80% records (28) for training and 20% records (8) for testing and with random forest we got test data prediction accuracy as 60%. Now click on ‘Naïve Bayes’ button to train Naïve Bayes classifier with extracted tweets features.

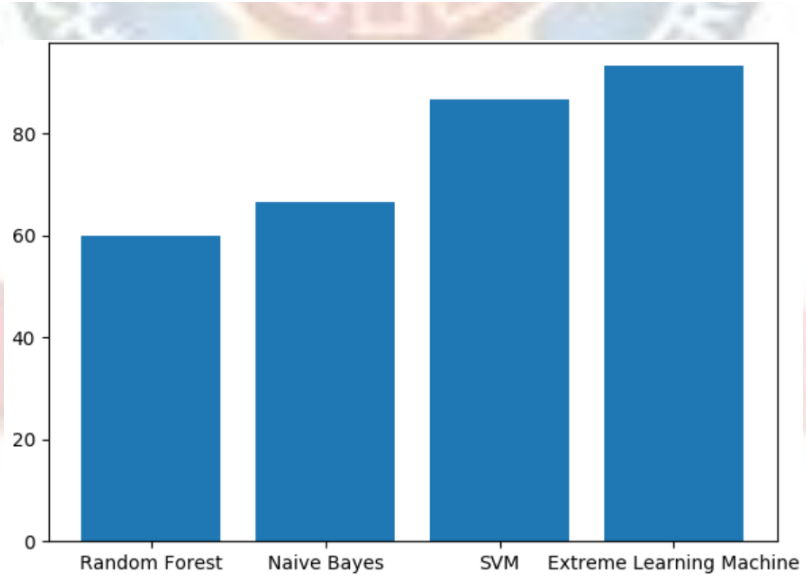


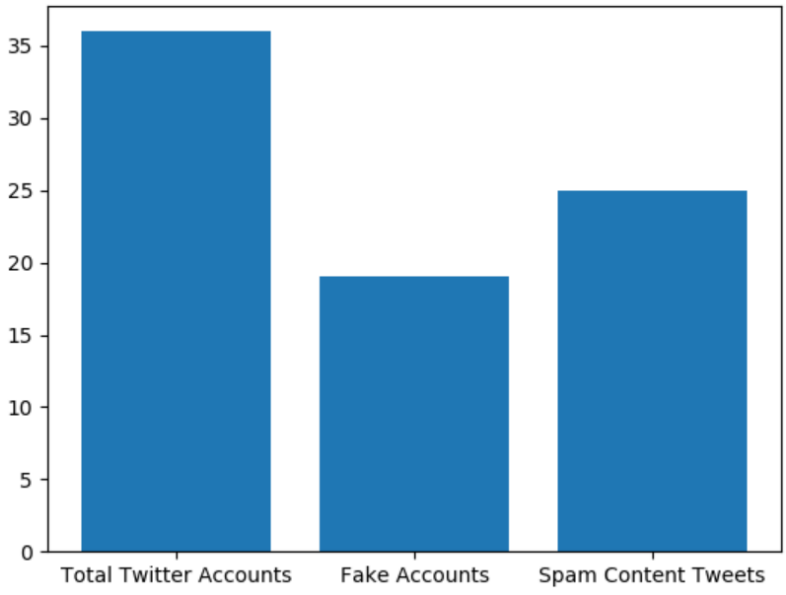
Now click on ‘SVM’ button to train SVM classifier with extracted tweets features.



Now click on extreme learning machine to train it with extracted tweets features and this model will be used to predict the fake user, and spam account for future test data.







In above graph x-axis represents total tweets, fake account and spam words content tweets and y-axis represents count of them.

CHAPTER 11

CONCLUSION AND FUTURE SCOPE

11.1 Conclusion

It is estimated that 35 billion spam email messages per day were generated in 2004. These messages are nuisance to the receivers and in addition create low availability and network congestion. The problem of spam in VoIP networks has to be solved in real time compared to e-mail systems. Many of the techniques devised for e-mail spam detection rely upon content analysis and in the case of VoIP it is too late to analyse the media after picking up the receiver. So, we need to stop the spam calls before the telephone rings. The proposed algorithm is ELM. In computing, trust has traditionally been a term relating to authentication, security, or a measure of reliability. When it comes to receiving or rejecting a voice call social meaning of trust is applied and reputation of the calling party is analysed.

11.2 Future Scope

The field of combatting spam and identifying fake users on social media platforms like Twitter is continually evolving due to advances in technology, changes in online behaviour, and the increasing importance of maintaining a safe and trustworthy online environment. Here are some future scope areas and trends in this field:

Advanced Machine Learning Techniques: As machine learning and artificial intelligence continue to advance, the development and integration of more sophisticated algorithms and models for spam and fake user detection are expected. This may include the use of deep learning approaches such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

Deepfake Detection: With the rise of deepfake technology, there is a growing need to detect and combat fake user accounts that use AI-generated images and videos to impersonate real individuals. Future research may focus on deepfake detection techniques specific to social media platforms.

Multimodal Analysis: Integrating text, image, and audio analysis for comprehensive content and user behaviour assessment is becoming more important. Multimodal approaches can provide a more holistic view of social media activity.

Behavioural Analysis: Analysing user behaviour patterns, such as the timing and frequency of posts, engagement with other users, and content sharing habits, will

continue to play a significant role in identifying fake accounts and spammy behaviour.

Real-Time Monitoring: Enhanced real-time monitoring systems will be developed to quickly identify emerging spam trends and fake accounts, enabling faster responses and mitigations.

Natural Language Processing (NLP) Advancements: The field of NLP will continue to evolve, leading to more accurate sentiment analysis, context understanding, and detection of subtle linguistic cues that may indicate spam or fake content.

Privacy Considerations: Balancing the need for spam and fake user detection with user privacy is a critical concern. Future developments may include privacy-preserving techniques that allow for effective detection without compromising user privacy.

Adversarial Attacks: Spam creators and fake account operators are likely to become more sophisticated in their tactics. Research will focus on developing defenses against adversarial attacks aimed at evading detection.

Regulatory Compliance: As governments and regulatory bodies become more involved in regulating online content and user accounts, there will be a need for systems that ensure compliance with legal requirements while maintaining user privacy and freedom of expression.

Customized User Experiences: Social media platforms will continue to leverage spam and fake user detection to provide users with customized experiences, such as personalized content recommendations and safer online interactions.

Education and Awareness: Public awareness campaigns and educational initiatives may be launched to help users recognize and report spam and fake accounts effectively.

Intraplatform Collaboration: Collaboration between different social media platforms and online communities to share insights and best practices for spam and fake user detection can lead to more effective solutions.

REFERENCES

[1] O. Varol et al., “Online Human-Bot Interactions: Detection, Estimation, and Characterization,” Proc. ICWSM, Montreal, Canada, May 2017.

[2] S. Liu, J. Zhang, and Y. Xiang. “Statistical Detection of Online Drifting Twitter Spam,” Proc. ACM ASIACCS, Xi’an, China, May 30–June 3 2016, pp. 1–10

[3] H. Zhu et al., “ShakeIn: Secure User Authentication of Smartphones with Habitual Single-handed Shakes,” IEEE Trans. Mobile Computing, vol. 16, no. 10, 2017, pp. 2901–12

[4] B. Feng, Q. Fu, M. Dong, D. Guo and Q. Li, "Multistage and Elastic Spam Detection in Mobile Social Networks through Deep Learning," in IEEE Network, vol. 32, no. 4, pp. 15-21, July/August 2018, doi: 10.1109/MNET.2018.1700406.

[5] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi and P. Samarati, "P2P-based collaborative spam detection and filtering," Proceedings. Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings., 2004, pp. 176-183, doi: 10.1109/PTP.2004.1334945.

[6] X. Hu, J. Tang, H. Gao and H. Liu, "Social Spammer Detection with Sentiment Information," 2014 IEEE International Conference on Data Mining, 2014, pp. 180-189, doi: 10.1109/ICDM.2014.141.

[7] M. Mateen, M. A. Iqbal, M. Aleem and M. A. Islam, "A hybrid approach for spam detection for Twitter," 2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST), 2017, pp. 466-471, doi: 10.1109/IBCAST.2017.7868095.

[8] Z. Wu, J. Cao, Y. Wang, Y. Wang, L. Zhang and J. Wu, "hPSD: A Hybrid PU-Learning-Based Spammer Detection Model for Product Reviews," in IEEE Transactions on Cybernetics, vol. 50, no. 4, pp. 1595-1606, April 2020, doi: 10.1109/TCYB.2018.2877161.

[9] K. Thomas, C. Grier, J. Ma, V. Paxson and D. Song, "Design and Evaluation of a Real-Time URL Spam Filtering Service," 2011 IEEE Symposium on Security and Privacy, 2011, pp. 447-462, doi: 10.1109/SP.2011.25.

[10] G. Wang, S. Xie, B. Liu and P. S. Yu, "Review Graph Based Online Store Review Spammer Detection," 2011 IEEE 11th International Conference on Data Mining, 2011, pp. 1242-1247, doi: 10.1109/ICDM.2011.124.

[11] S. Shehnepoor, M. Salehi, R. Farahbakhsh and N. Crespi, "NetSpam: A Network-Based Spam Detection Framework for Reviews in Online Social

Media," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 7, pp. 1585-1595, July 2017, doi: 10.1109/TIFS.2017.2675361.

[12] F. Masood. "Spammer Detection and Fake User Identification on Social Networks," in IEEE Access, vol. 7, pp. 68140-68152, 2019, doi: 10.1109/ACCESS.2019.2918196.