Appendix 5

TABLE OF CONTENTS

Declaration i

Certificate ii

Acknowledgement iii

Table of Contents iv

List of Tables vi

List of Figures vii

Abstract viii

Chapter 1. Introduction

I.1 Introduction 1
I.2 Existing System 2
I.3 Problems in Existing Definition 3
I.4 Problem Definition 7
I.5 Feasibility Study 8
I.6 Motivation 9
I.7 Project Overview/Specifications 9
I.8 Hardware Specification 9
I.9 Software Specification 10
I.10 Overview of the report 10

Chapter 2. System Analysis & Design

2.1 Requirement Specification 14
2.2 Flowcharts / DFDs / ERDs 15
2.3 Design and Test Steps / Criteria 26
2.4 Algorithms and Pseudo Code 27
2.5 Testing and Validation 30
2.5.1 Unit Testing 30
2.5.2 Integrity Testing 32
2.5.3 User Testing 34
2.5.4 Software Maintenance 34

Chapter 3. Implementation and Results

3.1	Python	35
3.2	Features of Python	35
3.3	Applications	39
3.4	Python Libraries	40
3.5	Machine Learning	41
3.6	Main Code	42

Chapter 4. Conclusion and Future Enhancements

4.1 Summary of work done 47
4.2 Proposal/scope of future enhancement 48

References/Bibliography 49

List of Tables

Table		Page No.
Table 1	Unit Testing	30
Table 2	Integrity Testing	32
Table 3	User Integrity	34

List of Figures

Figure		Page No.
Figure 1	Flow chart	15
Figure 2	Data Flow Diagram	17
Figure 3	Sequence Diagram	20
Figure 4	Class Diagram	23
Figure 5	Collobaration Diagram	24
Figure 6	Design	25
Figure 7	Implementation	57

Abstract

This project aims to determine the authenticity of account details, distinguishing between genuine and fraudulent information through the application of Artificial Neural Networks (ANN). The ANN algorithms will undergo training using a dataset comprising both authentic and fake account details. Upon completion of the training phase, these algorithms will be utilized to assess new account details for their legitimacy.

Online social networks, such as Facebook and Twitter, maintain extensive user information that can be targeted by malicious intruders intent on breaching their databases and compromising user privacy. To protect this sensitive information, we are deploying the ANN algorithm.

While the majority of online comments in public forums tend to be positive, there exists a significant minority that is toxic in nature. To address this, we utilize online data sources to compile datasets that have been cleansed of extraneous noise. The raw comments within the dataset require processing through a machine learning model, specifically employing the Term Frequency-Inverse Document Frequency (TF-IDF) technique, prior to being input into classification models. The presence of numerous errors in the comments results in a doubling of the feature set. The refined dataset is then trained using a logistic regression approach to differentiate between harmful and non-toxic comments. The multiheaded model offers two prediction categories: toxicity (including severe toxicity, obscenity, threats, insults, and identity-based hate) and non-toxicity assessment. In summary, our approach demonstrates the efficacy of machine learning algorithms in classifying harmful remarks by automatically identifying and filtering out negative comments that violate community standards, thereby fostering a more constructive online discourse.

Chapter 1 - Introduction

1.1 Introduction:

In the contemporary digital landscape, the growing reliance on computer technology has rendered the average individual susceptible to various crimes, including data breaches. Currently, there exists a limited incentive for social media platforms to improve their data security protocols. These security incidents often target major social media entities such as Facebook and Twitter, as well as financial institutions like banks. In 2017, Facebook reported a user base of 2.46 billion, reinforcing its position as the foremost social media platform. These companies derive their revenue from the data provided by their users, many of whom remain unaware that they forfeit certain rights by engaging with these services. Social media organizations stand to gain significantly at the expense of their users. Each time a user shares their location, uploads photos, expresses preferences, or tags others in their posts, Facebook reaps financial rewards through advertising and data monetization. Specifically, the average American user contributes around $26.76 quarterly, a sum that escalates rapidly with millions of users involved. Recently, Facebook faced a data breach affecting approximately 50 million users. While Facebook provides a comprehensive set of resources outlining the management of users' sensitive data, these policies do little to alleviate the persistent exploitation of security and privacy. Additionally, the presence of fake profiles poses a significant risk, as they can bypass Facebook’s security measures.

Another concern regarding the unauthorized acquisition of personal data for fraudulent activities is the existence of fake profiles and bots. Bots are automated programs capable of gathering user information without consent, a practice known as web scraping. These bots can operate discreetly or masquerade as fake friend requests on social networking sites to gain access to private information. One of the most remarkable innovations of the 21st century, the "Internet," enables individuals to connect globally using merely a smartphone and internet access. Machine learning algorithms have been suggested as a potential remedy for this issue. Large datasets of comments that have been classified as dangerous or non-toxic can teach machine learning algorithms patterns that can be used to categorizes fresh comments. In recent years, a number of Machine Learning techniques, including Naive Bayes, Decision Tree, Support Vector Machine, and Random Forest, have been evaluated for their efficiency in detecting hazardous comments.

1.2 Existing System:

Identifying Fake Profiles

Malicious actors create fake profiles to obtain login information from unsuspecting users. These deceptive profiles send friend requests to a wide array of users with public profiles, luring them in with images of individuals considered attractive. Once a user accepts the request, the owner of the fraudulent profile inundates them with additional friend requests.

The content of these counterfeit profiles often includes links that direct users to external websites where harm can occur. An unsuspecting and curious user may click on these malicious links, potentially compromising their computer. The consequences can range from acquiring a virus to the more severe outcome of installing a rootkit, which can turn the computer into a botnet. Although Facebook employs rigorous screening measures to eliminate these false accounts, it only takes one fraudulent profile to jeopardize the security of many users' computers.

Toxic Comments

The existing system utilizes the Naïve Bayes Algorithm. In this approach, texts are categorized based on posterior probabilities derived from the presence of various classes of words within the texts. This methodology allows for significantly more efficient computational resource usage compared to non-Naïve Bayes methods, which exhibit exponential complexity. However, it has been observed that Naïve Bayes tends to be less accurate for text classification tasks.

The Naïve Bayes algorithm is widely employed in machine learning for text classification, including the identification of toxic comments. In the context of the paper "Classification of Toxic Comments by Using Machine Learning Algorithms," the authors likely implemented a standard Naïve Bayes algorithm for this purpose, which involves several key steps:

1. **Data Pre-processing**: Initially, the text data, consisting of comments, undergoes preprocessing, which includes the removal of stop words, conversion of all characters to

lowercase, and elimination of punctuation marks. Subsequently, the text data is represented as a bag-of-words, where each comment is depicted as a vector of word frequencies.

2. **Training**: The Naïve Bayes algorithm is then trained on a labelled dataset of comments prior probabilities of each class (toxic and non-toxic) and the conditional probabilities of each word given each class.

Classification: When a new comment is received, the Naive Bayes algorithm computes the likelihood that the comment fits into each category by analyzing the conditional probabilities associated with the words present in the comment. Subsequently, the algorithm categorizes the comment into the class that exhibits the highest probability.

Model Evaluation: The efficacy of the Naive Bayes algorithm is assessed through various metrics, including accuracy, precision, recall, and F1 score, utilizing a test dataset comprised of comments.

1.3 Problems in Existing System

Fake Profiles

Limited adaptability: Rule-based systems depend on established rules, which may struggle to adjust to emerging types of fake profiles. As the creators of fake profiles enhance their techniques, these systems may become increasingly ineffective in identifying such profiles.

High false positive rates: Systems based on machine learning may exhibit elevated false positive rates, resulting in the misclassification of some legitimate profiles as fake. This situation can lead to user dissatisfaction and a lack of trust in the system.

Data bias: Machine learning systems may be susceptible to bias if the training data lacks diversity or is not representative. This can result in inaccurate outcomes and may disproportionately affect certain user groups.

Privacy concerns: Some existing systems may necessitate access to personal information and data from social media platforms, raising privacy issues for users.

Complexity: Hybrid systems that integrate both rule-based and machine learning approaches can be intricate to develop and maintain. Their successful implementation may require significant resources and specialized expertise.

In summary, these challenges underscore the necessity for more precise, adaptable, and privacy-conscious strategies for identifying fake profiles on social media.

Toxic Comments

1. Vulnerable to irrelevant attributes.

2. Restricted expressiveness.

3. Insufficient data.

4. Simplistic assumption of feature independence.

5. Vulnerable to irrelevant attributes.

Fake Profiles

Proposed System

In our proposed framework, we utilize an artificial neural network to evaluate the probability of a friend request being authentic or fraudulent. Microsoft Excel is employed to manage both historical and current fictitious data profiles. The algorithm subsequently organizes this information into a data frame. This dataset will be partitioned into training and testing sets. We will require a dataset from social media platforms to train our model.

In the training dataset, the features used to identify a fraudulent profile include account age, gender, user age, description link, number of messages sent, number of friend requests made, specified location, IP-based location, and the classification of fake or not. Each of these parameters is assessed and assigned a value. For the gender parameter, if the profile is classified as either female or male, a value of (1) is assigned to the training set for gender. This methodology is similarly applied to the other parameters. The country

of origin is also taken into account. We then compute the parameter for the number of messages sent by dividing the total number of messages dispatched by the account age. The parameter for the number of friend requests sent is determined by dividing the total number of friend requests made by the account age. This process is primarily utilized for multi-dimensional matrix multiplication, as we are handling a substantial amount of interdependent numerical data.

Advantages of the Proposed System

The proposed system utilizing artificial neural networks (ANNs) for the identification of fake profiles on social media presents numerous potential benefits, including:

1**. Enhanced Accuracy**: ANNs possess the capability to learn intricate patterns within data, enabling them to discern subtle distinctions that may signify a fake profile. This results in a higher level of accuracy in detecting fraudulent profiles compared to traditional rule-based or conventional machine learning systems.

2**. Flexibility**: ANNs are capable of adapting to emerging types of fake profiles and can continuously learn and refine their detection methods over time. This adaptability enhances the system's effectiveness in identifying fake profiles as they change.

3. **Scalability:** ANNs are proficient in managing extensive datasets, which is advantageous for analysing social media profiles that may encompass vast amounts of information. This characteristic renders the system scalable and efficient for application on large social media platforms.

4. **Decreased False Positive Rates**: ANNs can be trained to reduce false positive rates, allowing the system to identify fake profiles with a high level of confidence while minimizing the risk of incorrectly flagging legitimate profiles as fraudulent.

5. **Privacy Considerations**: ANNs can be trained using anonymized data, addressing privacy concerns for users. Furthermore, the proposed system does not necessitate access to personal information or data from social media platforms, thereby further alleviating privacy issues.

In summary, the proposed system employing ANNs for the detection of fake profiles on social media offers several significant advantages, positioning it as a promising solution for enhancing the accuracy and efficiency of fake profile identification.

Toxic Comments

Proposed System: The recommended strategy employs the Random Forest algorithm for text classification tasks. In traditional Random Forest applications, the importance of both the quantity and quality of feature selection is critical. However, in the context of classifying books and other large text corpora, an enhancement in both the number and quality of text features is positively associated with improved classification performance. This paper introduces a tr-k method that integrates TF-IDF, Text-Rank, and K-means to bolster the effectiveness of text classification. The full term for TF-IDF is term frequency-inverse document frequency. The Random Forest algorithm is a widely utilized machine learning technique for classification purposes and can effectively classify toxic comments. The following outlines a proposed system utilizing the Random Forest algorithm for this objective:

1. **Data Preprocessing**: The first step involves preprocessing the data, which includes cleaning and transforming it. This process may consist of removing stop words, stemming, and tokenizing the comments.

2**. Feature Extraction**: After the data has been cleaned, the next phase is to extract features. This can be accomplished through various methods, such as bag-of-words or TF-IDF.

3. **Training Data Split**: Following feature extraction, it is essential to divide the data into training and testing datasets to assess the model's performance.

4. **Training the Random Forest Model**: Once the data is partitioned, the subsequent step is to train the Random Forest model using the training dataset. Tuning the model's hyperparameters is crucial for enhancing its performance. These hyperparameters include

the number of trees in the forest, the depth of the trees, and the number of features considered for each split.

5. **Predictions**: After training, the model can be utilized to determine whether new comments are toxic or not.

6. **Model Evaluation**: The final step involves evaluating the model's performance using the testing dataset. This evaluation can be conducted using various metrics, including accuracy, precision, and recall. Therefore, careful data preprocessing and hyperparameter tuning are critical for achieving high accuracy in this task

Advantages

1. Exceptional accuracy.

2. Resilient to noisy data.

3. Capable of managing high-dimensional datasets.

4. Proficient in capturing interactions among features.

5. Offers ranking of feature importance**.**

1.4 Problem Definition

Online platforms frequently encounter difficulties in identifying fake profiles and toxic comments due to the immense volume of user-generated content, the diverse strategies employed by malicious individuals, and the inherently subjective nature of toxicity. These issues adversely affect user experience, pose risks such as fraud, harassment, or manipulation, and elevate the operational burden on moderators.

1. **Fake Profiles**: These are fraudulent accounts designed to mimic legitimate users, mislead others, promote scams (such as phishing), or disrupt platform activities (for instance, through bots). They typically display behavioral patterns that diverge from those of authentic user interactions.

2**. Toxic Comments**: These refer to remarks or content that are offensive, abusive, discriminatory, or otherwise detrimental to individuals or groups. Such comments can inflict emotional distress, exacerbate conflicts, or diminish the overall safety and inclusivity of the platform.

1.5 Feasibility Study:

Introduction: The proliferation of social media in recent years has corresponded with a significant increase in the number of fake profiles across various platforms. These profiles are created for multiple purposes, including disseminating false information, perpetrating scams, and engaging in phishing activities. The application of artificial neural networks to detect fake profiles has emerged as a promising solution to this challenge. This feasibility study seeks to assess the viability of employing artificial neural networks for the identification of fake profiles.

Objectives:

The primary aim of this feasibility study is to assess the viability of employing artificial neural networks for the detection of fake profiles on social media platforms. The specific goals of the study include:

1. Identifying the types of data necessary for training an artificial neural network to recognize fake profiles.

2. Evaluating the availability and quality of the data needed for training an artificial neural network to identify fake profiles.

3. Assessing the technical feasibility of implementing an artificial neural network for the identification of fake profiles.

4. Analysing the potential advantages and disadvantages of utilizing artificial neural networks for the detection of fake profiles.

1.6 Motivation:

The rise of social media platforms has revolutionized communication; however, it has also led to the emergence of fake profiles and harmful comments, which present significant challenges to online safety and community standards. Fake profiles are frequently employed for nefarious purposes, such as disseminating misinformation, perpetrating scams, or manipulating public sentiment, while harmful comments contribute to cyberbullying and foster hostile environments. Addressing these challenges is crucial for building trust and encouraging healthier online interactions.

Artificial Neural Networks (ANNs) represent a powerful solution to these issues. By harnessing their capacity to learn intricate patterns within data, ANNs can

effectively identify fake profiles based on behavioral indicators and detect toxic comments through sentiment analysis and natural language processing. This project seeks to create a robust system utilizing ANNs to enhance online safety, safeguard users, and support the broader initiative of maintaining ethical digital environments.

1.7 Project Overview/ Specifications

The initiative aims to create a system capable of identifying fraudulent profiles and detecting harmful comments on social media platforms through the use of Artificial Neural Networks (ANN). The existence of fake profiles presents considerable obstacles to online authenticity, while toxic comments detrimentally affect digital communities by propagating negativity. The proposed approach includes the preprocessing of data from user profiles and comments, feature extraction, and the application of ANN models to categorize profiles as either genuine or fraudulent and comments as either toxic or non-toxic.

Specifications:

1. Data Collection: Compile datasets of user profiles along with labelled comments.

2. Data Preprocessing: Refine data, standardize text, and extract features (such as sentiment and word embeddings).

3. Model Architecture: Develop and train distinct ANN models for the detection of fake profiles and the classification of toxic comments.

4. Evaluation Metrics: Employ accuracy, precision, recall, and F1-score to evaluate the performance of the models.

5. Output: Deliver insights regarding profile authenticity and comment toxicity, facilitating improved content moderation.

This project enhances the safety of online environments by utilizing artificial intelligence to uphold digital integrity.

1.8 HARDWARE SPECIFICATIONS

System: Pentium IV - 2.4 GHz

Hard Disk: 40 GB

Monitor: 15" VGA Color

RAM: 16 GB

1.9 SOFTWARE SPECIFICATIONS

Operating System: Windows 7 & Above

Coding Language: Python (3.11.1)

1.10 Overview of the Report

Identifying Fake Profiles

Economic Feasibility:

Artificial Neural Networks (ANNs) demonstrate significant effectiveness in recognizing patterns and anomalies within extensive datasets, rendering them particularly suitable for the detection of fake profiles on social media platforms. By training an ANN on a comprehensive dataset comprising both authentic and fraudulent profiles, the network can learn to identify characteristics commonly linked to fake profiles, such as inauthentic profile images and dubious login locations. Once adequately trained, the ANN can automatically flag profiles exhibiting these concerning traits, enabling platform administrators to swiftly identify and eliminate fake accounts.

Conversely, there are potential economic implications associated with the implementation of ANNs for this task. The development and training of a high-quality ANN can be necessitate specialized knowledge, which may incur substantial costs. Furthermore, the operation of ANNs demands considerable computing resources, contributing to the overall expense.

Nevertheless, in numerous instances, the advantages of employing ANNs for the identification of fake profiles may surpass the associated costs. By minimizing the prevalence of fake accounts on their platforms, social media companies can enhance user experience and mitigate the risks of fraud or misuse. This improvement can subsequently

lead to increased user engagement and revenue generation. Additionally, as the utilization of ANNs becomes more prevalent and the technology advances, the costs related to the development and implementation of these systems may decline over time.

Technical feasibility:

To employ artificial neural networks (ANNs) for the detection of fake profiles, it is essential to gather and label a dataset comprising both authentic and fraudulent profiles. This dataset must be extensive and varied to enable the ANN to generalize effectively to unfamiliar instances.

After the dataset has been compiled and annotated, the ANN can undergo training. During this process, the ANN will learn to differentiate between genuine and fake profiles by recognizing patterns within the data. Once the training is complete, the ANN will be capable of assessing new profiles and categorizing them as either real or fake.

The technical viability of utilizing ANNs for the identification of fake profiles hinges on several factors, including the quality and volume of the data, the architecture of the neural network, and the computational resources at hand. Nevertheless, with appropriate resources and expertise, it is feasible to create an efficient ANN-based system for detecting fake profiles.

From a social perspective, the effectiveness of the ANN model in accurately identifying fake profiles is paramount in assessing its social acceptability. An inaccurate model that often misclassifies legitimate profiles as fraudulent could inflict considerable harm on innocent users.

Moreover, user privacy concerns must be taken into account. Training the ANN would likely necessitate access to substantial amounts of user data, which could be perceived as a breach of privacy. Therefore, it is crucial to establish clear guidelines and maintain transparency regarding the collection, usage, and protection of user data.

Additionally, there may be legal and ethical implications associated with employing an ANN for the identification of fake profiles. Certain jurisdictions may have regulations

that restrict the use of automated systems for making decisions that significantly affect individuals, such as those related to employment or financial matters.

Lastly, cultural and societal attitudes towards the use of ANNs for identifying fake profiles may vary. Some individuals might regard it as a vital instrument for combating online fraud and safeguarding users, while others may perceive it as an infringement on privacy or an excessive exercise of authority.

Toxic Comments

Economic Feasibility

The project titled "Classification of toxic comments using machine learning algorithms" is influenced by various economic factors, including the expenses associated with development, maintenance, and deployment, alongside the anticipated benefits that may arise from the initiative.

Key considerations include:

1. Development Costs: The expenses related to the creation of machine learning models can differ based on the complexity of the algorithms, the volume of data, and the qualifications of the development team. Utilizing open-source libraries and pre-trained models can help mitigate these costs.

2. **Maintenance Costs:** After deployment, the models will likely require continuous maintenance and updates to maintain optimal performance. This includes monitoring accuracy, retraining with new data, and resolving any emerging issues.

3. **Deployment Costs**: The implementation of the model may also incur expenses related to hosting, cloud computing services, and integration with existing infrastructures.

4. **Potential Benefits:** The economic viability of the project is also contingent upon the potential advantages it may offer. For example, if the classification of toxic comments contributes to the reduction of cyberbullying, hate speech, and online harassment, it could yield substantial social and economic gains.

TECHNICAL FEASIBILITY

The endeavour of "classification of toxic comments using machine learning algorithms" is technically achievable given the advancements in contemporary machine learning methodologies.

1. **Data Availability:** Numerous datasets containing toxic comments are accessible online, which can be utilized for training and evaluating machine learning models. Notable examples include the Toxic Comment Classification Challenge dataset from Kaggle and the Wikipedia Detox dataset.

2.**Text Preprocessing Techniques:** Various text preprocessing methods, such as tokenization, stemming, and lemmatization, can be employed on the toxic comments dataset to derive significant features for training machine learning models.

SOCIAL FEASIBILITY

The social feasibility of deploying a machine learning-based system for the classification of toxic comments is contingent upon various factors, including the intended context and objectives of the system, potential ethical implications, and societal attitudes towards the technology.

Consider the following aspects:

1. Context and objectives: The specific application of a machine learning-based system for identifying toxic comments plays a crucial role in assessing its social feasibility. If the system is designed for use by online platforms to moderate comments and mitigate toxicity, it may be perceived as a constructive measure to foster civil discourse and diminish hate speech. Conversely, if the system is employed by governmental entities to surveil and restrict free speech, it could be regarded as an infringement on privacy and freedom of expression.

2. Ethical implications: The implementation of a machine learning-based system for classifying toxic comments raises several ethical concerns. A significant issue is algorithmic bias, which may result in the system inaccurately categorizing certain comments as toxic or benign, thereby leading to disproportionate impacts on specific demographic groups.

Chapter 2. System Analysis & Design

2.1 Requirement Specification:

Neural Networks are mathematical structures specifically designed to tackle optimization problems. They are composed of neurons, which serve as the essential computational elements within these networks. Each neuron receives an input (represented as x), conducts a computation (for example, multiplying it by a weight variable w and adding a bias variable b) to produce a value (formulated as $z = wx + b$). This value is subsequently processed through a non-linear function known as the activation function (f), resulting in the neuron's final output (activation). There are numerous activation functions available, with the Sigmoid function being among the most prevalent. A neuron that utilizes the Sigmoid function for its activation is termed a sigmoid neuron. Neurons are classified according to their activation functions, leading to various types such as RELU and Tan H. When neurons are organized in a

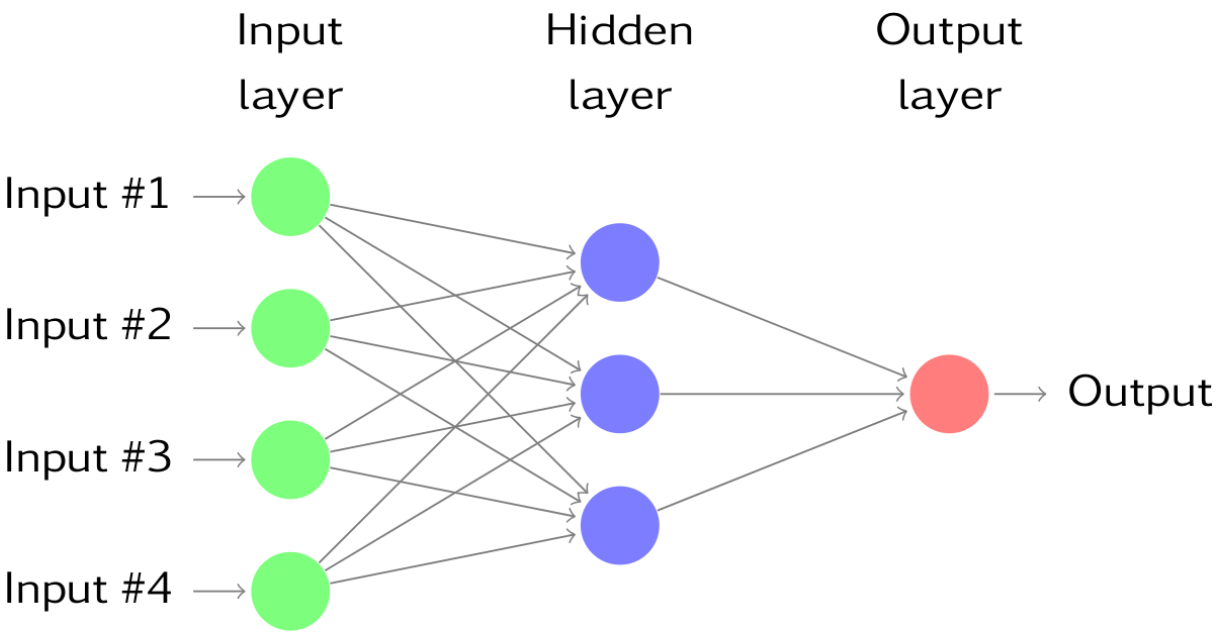
sequential manner, this arrangement is referred to as a layer, which represents the next fundamental component of neural networks.

The primary purpose of this document is to outline the functional and non-functional requirements for a system capable of accurately identifying toxic comments on online platforms. Toxic comments, characterized by their harmful, offensive, or abusive nature, can negatively impact online communities. This system aims to mitigate the spread of such content.

This document covers the requirements for a system that can:Identify toxic comments within a given text corpus.

Classify toxic comments into specific categories

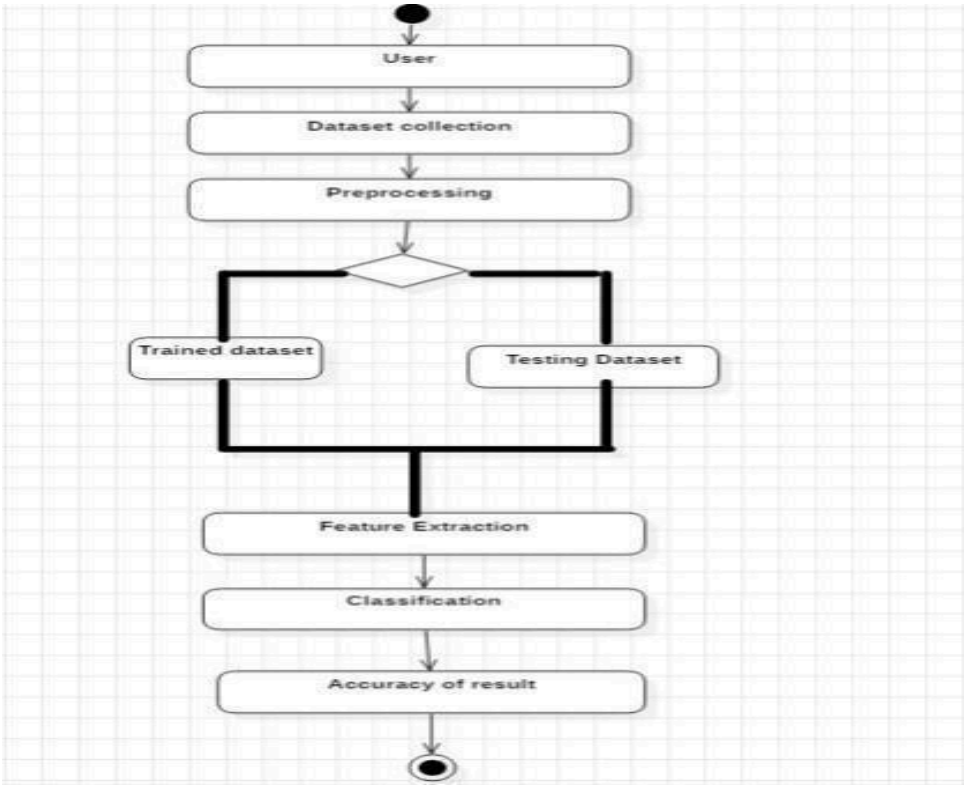
- Provide a confidence score for each classification.
- Integrate with existing online platforms to filter or flag toxic comments.



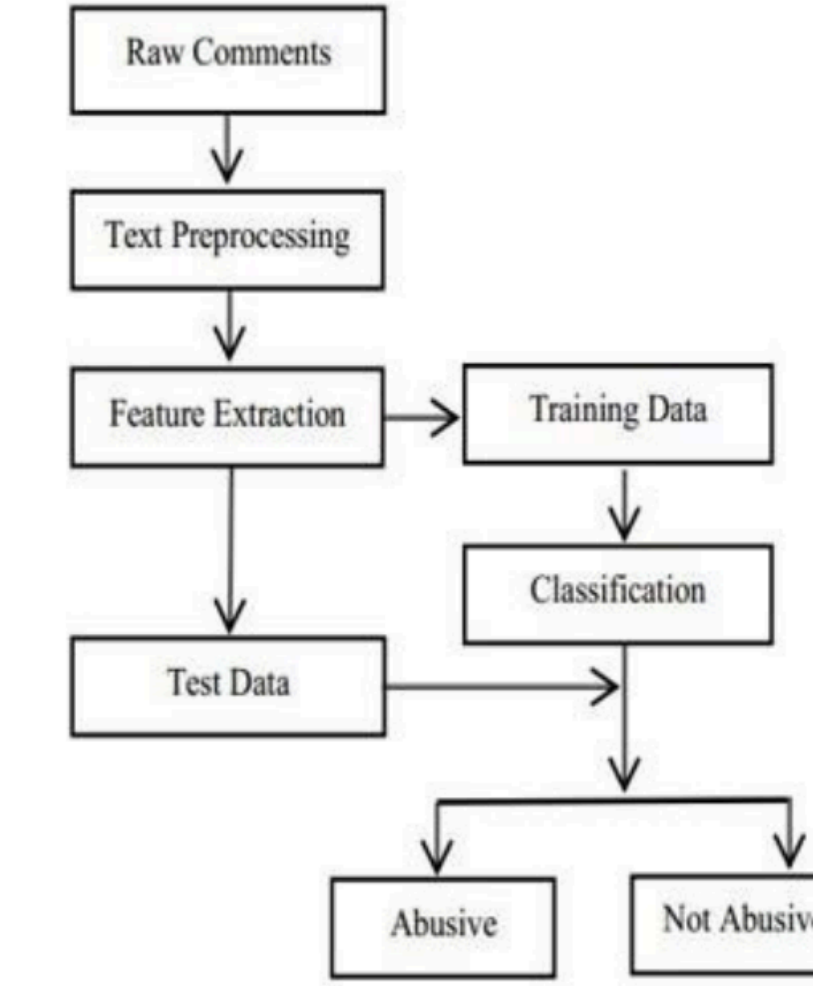
2.2 Flowcharts / DFDs / ERDs

Flow Chart

Fake Profiles



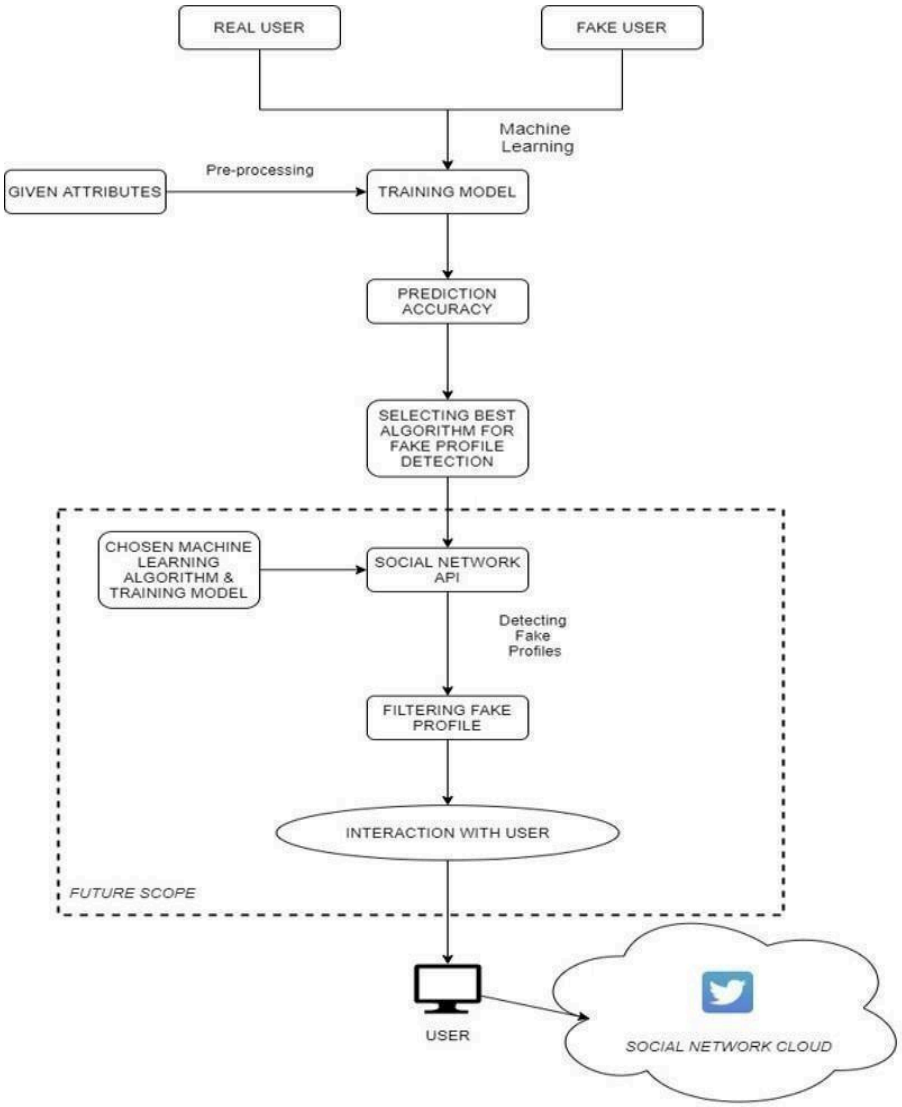
Toxic Comments



DATA FLOW DIAGRAM

Fake Profiles

A Data Flow Diagram (DFD) serves as a visual representation of the various functions and tasks that process, capture, store, and disseminate information within a system and its components. This straightforward graphical illustration facilitates communication and interaction between the user and the system designer.



Toxic Comments

The system's requirements are shaped by a combination of internal and external factors. This encompasses individuals, use cases, and various elements that engage the actors responsible for the creation of use case diagrams. It illustrates how an entity from the external environment can engage with a component of the system.

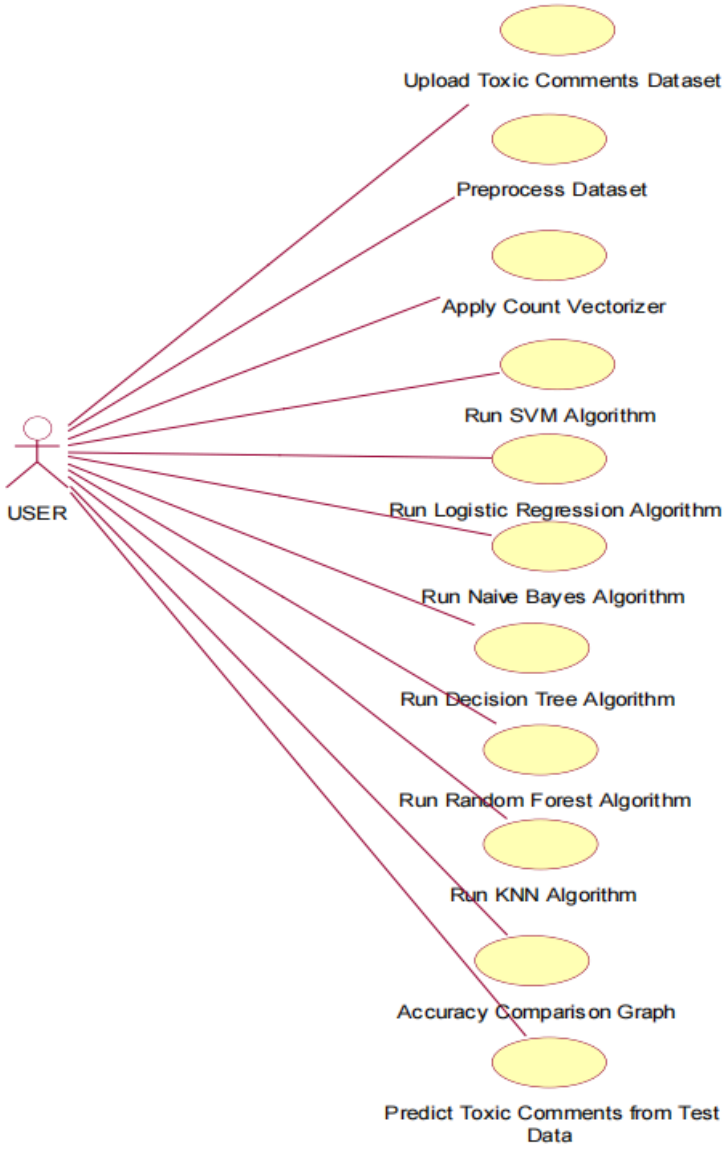
The following outlines the objectives of a data flow diagram:

1. It consolidates the requirements of the system.

2. It illustrates the system's external perspective.

3. It identifies both internal and external influences on the system.

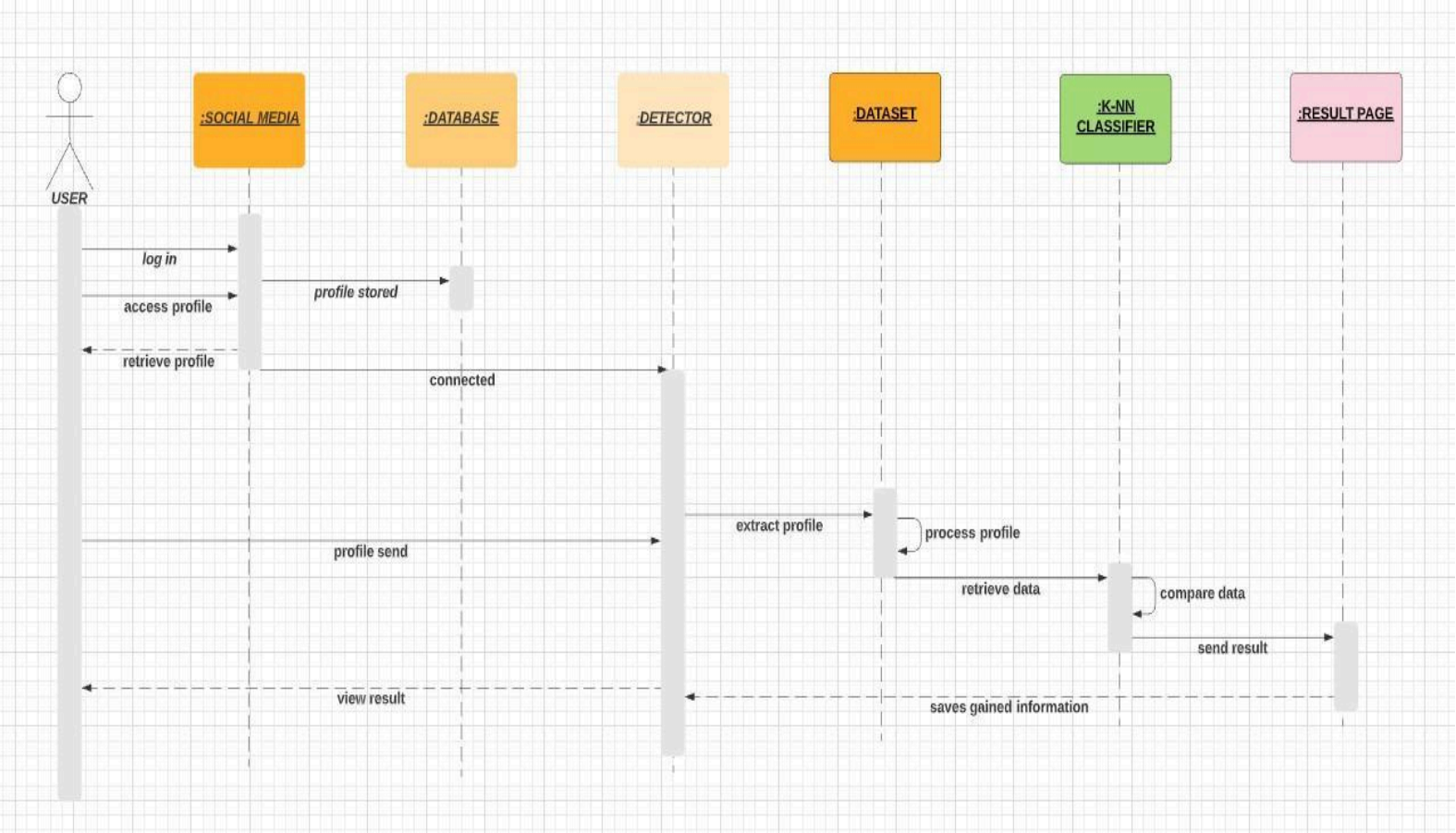
4. It demonstrates the interactions among the actors.



SEQUENCE DIAGRAM

Fake Profiles

The arrangement graph, also referred to as a connection chart, illustrates the interactions among the entities over a timeline. It depicts the items and categories involved in the scenario, as well as the communication occurring between them.



Toxic Comments

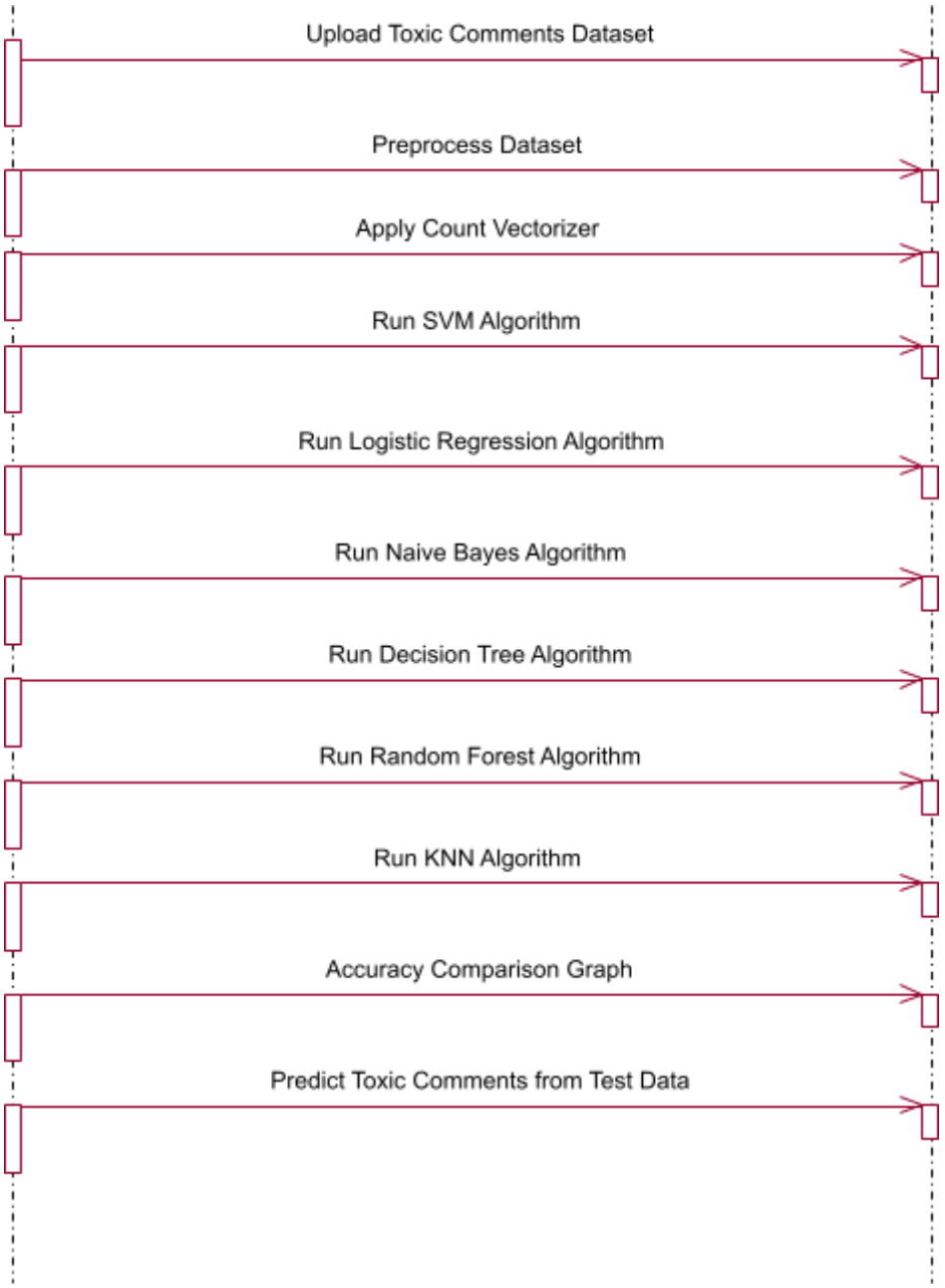
The sequence diagram illustrates the flow of messages within a system and is also referred to as an event diagram. It aids in visualizing various dynamic scenarios. This diagram depicts the communication between two lifelines as a time-ordered sequence of events that occur during runtime. In Unified Modeling Language (UML), a lifeline is represented by a vertical bar, while the flow of messages is indicated by a vertical dotted line that spans the bottom of the diagram. It encompasses both iterations and branching.

Purpose of Sequence Diagram

• To model the high-level interactions among active objects within a system.

• To represent interactions among objects within a collaboration that fulfills a use case.

• It can model either generic interactions or specific instances of interaction.



CLASS DIAGRAM

Toxic Comments

A class diagram is a specific type of UML diagram that illustrates the architecture of a system by depicting the classes, their attributes, methods, and the interconnections among them.

The components of a class diagram include:

1. Class: A class acts as a fundamental blueprint or template for object creation. It includes attributes (data members) and methods (functions or operations) that define the behaviour of the objects derived from it.

2. Object: An object signifies a particular instance of a class. It is instantiated according to the class definition and possesses its own unique values for the attributes.

3. Attributes: Attributes are the data members linked to a class that characterize the properties of an object.

4. Methods: Methods are the functions or operations that outline the behavior of the objects. They can manipulate the object's attributes.

5. Inheritance: Inheritance denotes a relationship between two classes where one class (the child or derived class) acquires the attributes and methods of another class (the parent or base class).

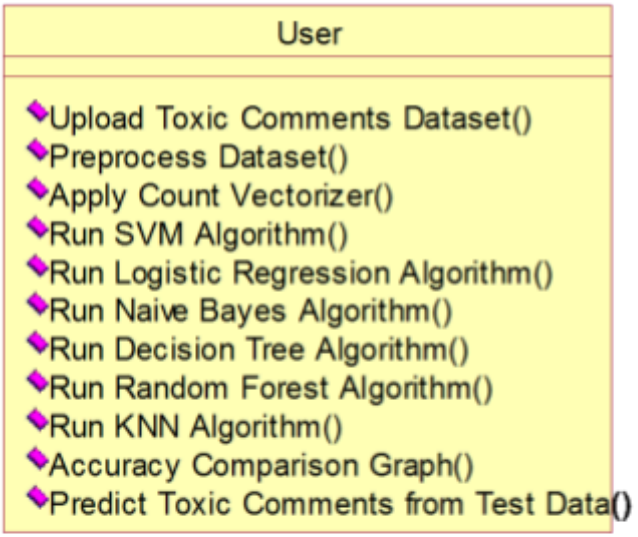
6. Association: Association represents a relationship between two classes, indicating that one class is connected to another.

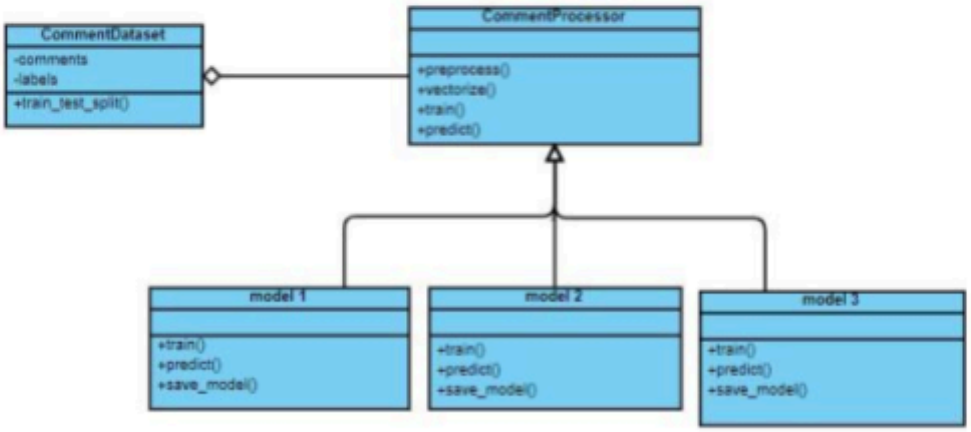
7. Multiplicity: Multiplicity indicates the number of instances of one class that can be associated with instances of another class, represented by a range such as 0..1 or 1..*.

8. Aggregation: Aggregation is a form of association that illustrates a "has-a" relationship between two classes, where one class serves as a container for another.

9. Composition: Composition is a specific type of aggregation that signifies a "part-of" relationship between two classes, where one class is made up of another class.

In summary, a class diagram offers a visual depiction of the system's structure, aiding developers in designing and comprehending the relationships among various classes within a software system.





COLLABORATION DIAGRAM

The collaboration diagram is designed to represent the interrelationships among objects in a system. Although sequence and collaboration diagrams provide equivalent information, they do so through different approaches. Instead of illustrating the sequence of messages, the collaboration diagram emphasizes the structural arrangement of objects within the system, aligning with the principles of object-oriented programming. Each object contains a range of attributes, and numerous objects within the system are linked together.

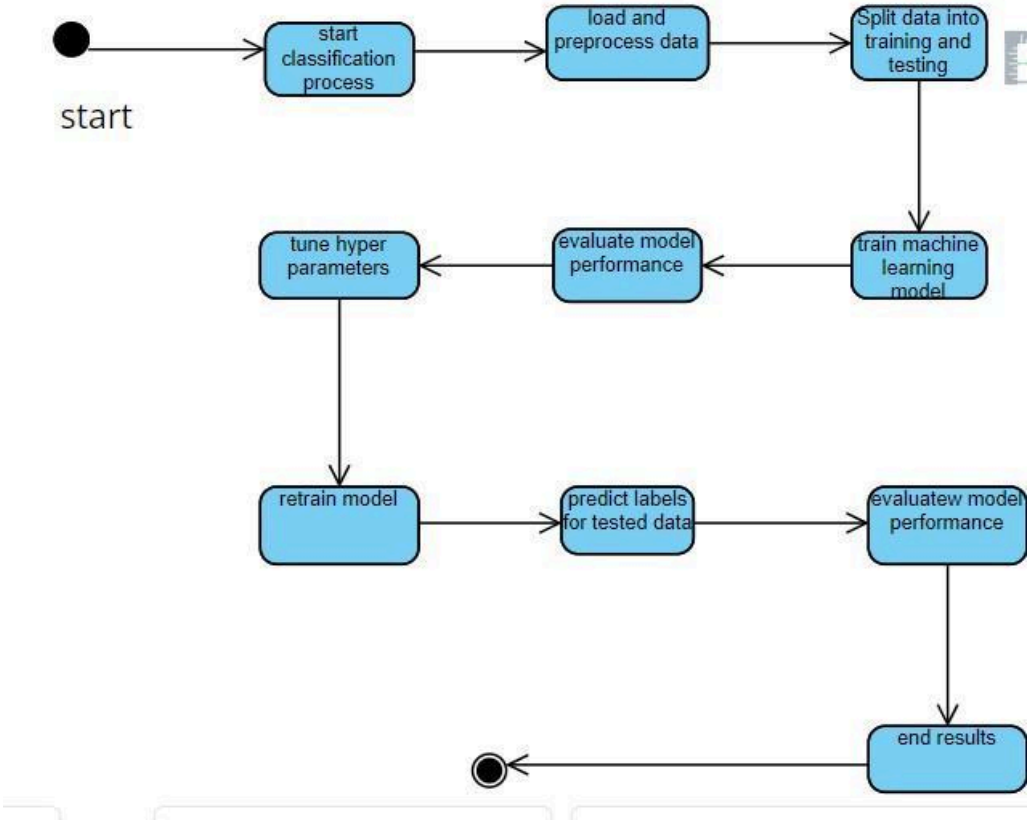
This diagram, which is also known as a communication diagram, is employed to depict the architectural framework of objects in the system.

1: Upload Toxic Comments Dataset
2: Preprocess Dataset
3: Apply Count Vectorizer
4: Run SVM Algorithm
5: Run Logistic Regression Algorithm
6: Run Naive Bayes Algorithm
7: Run Decision Tree Algorithm
8: Run Random Forest Algorithm
9: Run KNN Algorithm
10: Accuracy Comparison Graph
11: Predict Toxic Comments from Test Data

STATE CHART DIAGRAM

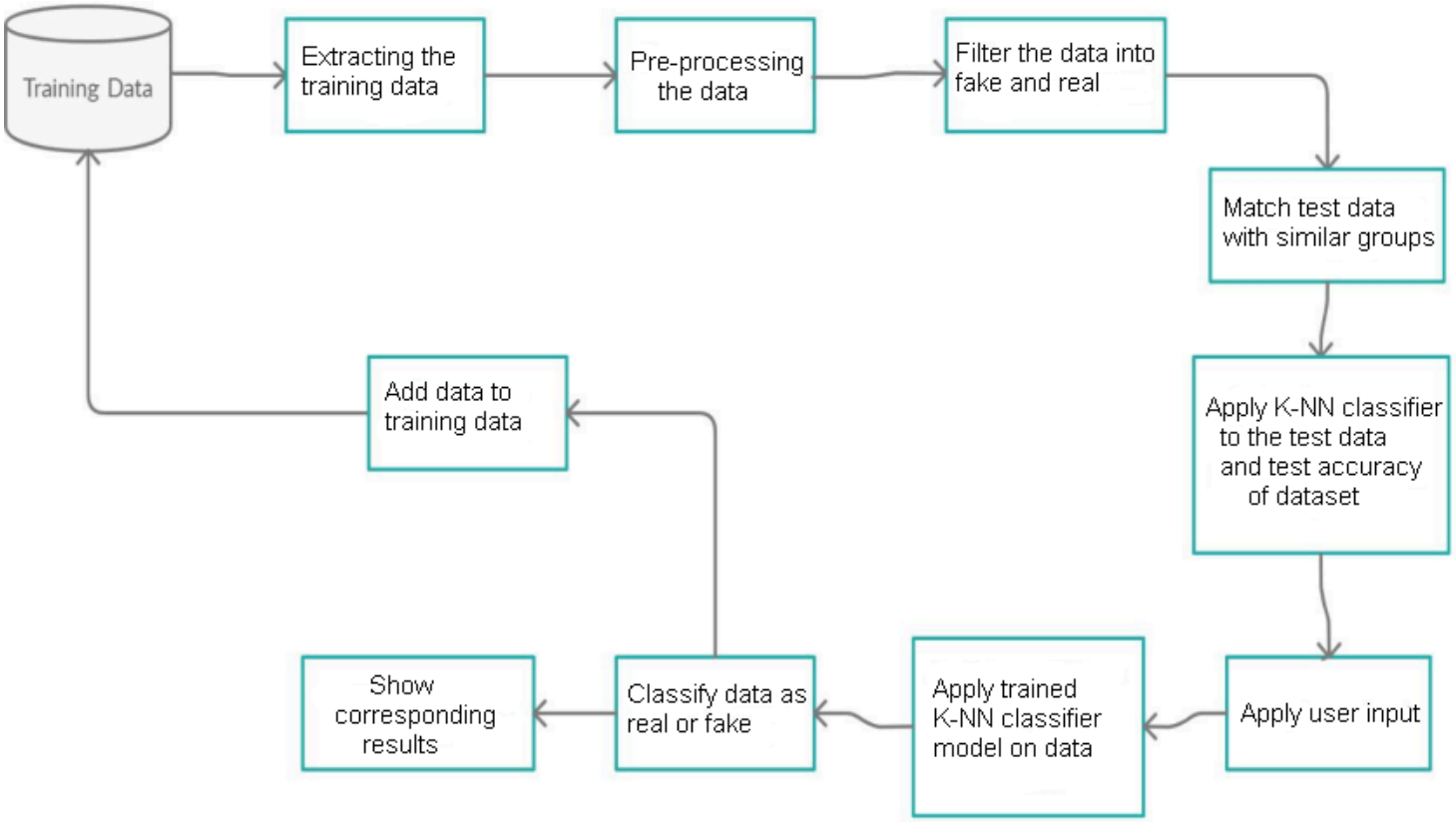
Toxic Comments

A state chart diagram, often referred to as a state machine diagram or state transition diagram, is a specific type of behavioral diagram within the Unified Modeling Language (UML) that illustrates the behavior of a system or object over a period. This diagram comprises a collection of states, transitions, and events that collectively define the system's or object's behavior. States denote the various conditions or situations that the system or object may occupy, while transitions signify the events that facilitate the movement from one state to another. Events act as the catalysts that initiate state transitions. State chart diagrams are instrumental in modeling the behavior of intricate systems, aiding in the identification of potential issues and opportunities for enhancement or optimization. They find widespread application in software engineering, control engineering, and other domains where the analysis and design of complex systems are essential.


start
start classification process
load and preprocess data
Split data into training and testing
train machine learning model
evaluate model performance
tune hyper parameters
retrain model
predict labels for tested data
evaluatew model performance
end results

2.3 Design

System design describes the structure and behavior of a complex system. It comprises all the components and the overview of the whole system.



2.3 Test Steps

Identifying Fake Profiles

The feasibility study will be executed utilizing the following methodology:

1. Literature Review: A comprehensive literature review will be undertaken to ascertain the current advancements in the application of artificial neural networks for the detection of fake profiles. This review will also delineate the types of data necessary for training an artificial neural network aimed at identifying such profiles.

2. Data Availability Assessment: An evaluation will be performed to assess the accessibility and quality of the data essential for training an artificial neural network to detect fake profiles. This evaluation will encompass a review of publicly accessible datasets alongside an analysis of the data's quality.

3. Technical Feasibility Assessment: A technical feasibility assessment will be carried out to ascertain the practicality of deploying an artificial neural network for the identification of fake profiles. This assessment will involve an examination of the hardware and software prerequisites, as well as an evaluation of the resources needed for the development and training of the network.

4. **Benefits and Drawbacks Analysis:** An analysis of the benefits and drawbacks will be conducted to assess the potential advantages and disadvantages of employing artificial neural networks for the identification of fake profiles. This analysis will include an evaluation of the network's accuracy, the likelihood of false positives and false negatives, and the potential implications for privacy.

Toxic Comments

The Random Forest algorithm is applicable for test classification by first training the model on a labelled dataset and then using it to forecast the class labels of new, unseen test data. To effectively implement the Random Forest algorithm for test classification, follow these steps:

Prepare the Data: Arrange the data for both training and testing purposes by dividing the labelled dataset into distinct training and testing subsets. The training subset will be employed to develop the algorithm, whereas the testing subset will be used to evaluate the precision of the algorithm's predictions.

Train the model: Employ the training dataset to train the Random Forest algorithm. This algorithm constructs numerous decision trees by utilizing random subsets of the training data along with its features**.**

Evaluate the model: Apply the testing dataset to assess the accuracy of the predictions made by the Random Forest algorithm. This involves comparing the predicted class labels against the actual class labels to derive metrics such as accuracy, precision, recall, and F1 score.

Tune the model: Should the model's performance fall short of expectations, modify the hyperparameters of the Random Forest algorithm, including the number of trees, maximum depth, and the number of features to consider at each split. Repeat the evaluation and tuning process until satisfactory performance is attained.

Predict on new data: After the model has been adequately trained and fine-tuned, it can be employed to predict the class labels for new, unseen test data.

2.4 Algorithms and Pseudocode

Algorithm

The method proposed herein is grounded in the Random Forest algorithm and is designed for text classification tasks. In the conventional Random Forest approach, the quantity and quality of feature selections play a crucial role. For extensive text classification endeavors, enhancing both the number and quality of text features (which serve as attributes for the classification decision tree) will significantly improve the classification process's efficacy. This paper presents a tr-k method that amalgamates TF-IDF, Text-Rank, and K-means to bolster the effectiveness of text classification. The full designation of the TF-IDF method is term frequency-inverse document frequency.

RANDOM FOREST ALGORITHM

The Random Forest algorithm can be effectively utilized for text classification by training it on a labelled dataset and subsequently employing it to predict the class labels of new, unseen test data. To use the Random Forest algorithm for test classification, follow these steps:

1. Data Preparation: Begin by organizing the data for both training and testing purposes by dividing the labelled dataset into distinct training and testing subsets. The training subset will serve to instruct the algorithm, while the testing subset will be utilized to assess the accuracy of the algorithm's predictions.

2. Model Training: Employ the training subset to train the Random Forest algorithm. This algorithm will generate multiple decision trees based on random selections of the training data and its features.

3.Model Evaluation: Utilize the testing subset to determine the accuracy of the predictions made by the Random Forest algorithm. This involves comparing the predicted class labels against the actual class labels to compute various metrics, including accuracy, precision, recall, and F1 score.

4.Model Tuning: Should the model's performance fall short of expectations, modify the hyperparameters of the Random Forest algorithm, such as the number of trees, maximum depth, and the number of features considered at each split. Repeat the training and evaluation steps until the desired performance level is reached.

5.Prediction on New Data: After the model has been adequately trained and fine-tuned, it can be employed to predict the class labels for new, previously unseen test data.

Pseudocode

Identifying Fake Profiles

```
import pandas as pd
import numpy as np
import tensorflow as tf
dataset = pd.read_csv('fake_profiles.csv')
X = dataset.iloc[:, 0:7].values
y = dataset.iloc[:, 7].values
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
X[:, 4] = labelencoder_X.fit_transform(X[:, 4])
X[:, 6] = labelencoder_X.fit_transform(X[:, 6])
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(units=6, activation='relu', input_dim=7))
model.add(tf.keras.layers.Dense(units=6, activation='relu'))
model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=32, epochs=100)
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

Toxic Comments

PSEUDO CODE

```
# importing libraries import
numpy as nm import
matplotlib.pyplot as mtp import
pandas as pd

#importing datasets data_set=
pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values y=
data_set.iloc[:, 4].values

from sklearn.model_selection import train_test_split x_train, x_test, y_train,
y_test= train_test_split(x, y, test_size= 0.25, random_sta te=0)

import StandardScaler st_x= StandardScaler()
x_train= st_x.fit_transform(x_train) x_test=
st_x.transform(x_test)
```

2.5 Testing and Validation

2.5.1 Unit Testing

Fake Profiles

This phase involves the testing of software, focusing on the assessment of specific components or elements within applications. The objective is to ensure that each application system functions according to its intended design. A unit represents the smallest testable component of the applications. The development and review of the unit test occur during the initial phase of the project. Following this, test cases and scripts are created to evaluate the application. In this project, individual modules such as data

pre-processing, clustering, user registration, and login are segmented into smaller fundamental units to assess their functionalities and confirm that they operate as anticipated.

Test Id	Test Actions	Input	Expected Output	Actual Output	Pa / Fa
T1	Check the integrity of the dataset	Access the dataset and check the attributes of the slices	Access to the dataset and all its attributes	Success	Pa
T2	Check the fake profile	Record the fake profile	100% fake	Success	Pa
T3	Check the genuine profile	Record the genuine profile	0% fake	Success	Pa
T4	Check with incorrect data	Record of social profile	Error	Success	Pa
T5	Training the data	Pre-processed data file	Prediction of fake profile with accuracy	Success	Pa

UNIT TESTING

Toxic Comments

Unit testing plays a vital role in the software development lifecycle, particularly when dealing with machine learning algorithms. In the context of "Classification of Toxic Comments using Machine Learning Algorithms," unit testing entails evaluating each function and method within the code to confirm that they yield the anticipated results for a range of inputs. The unit tests may encompass assessing the accuracy of the trained machine learning models, examining the pre-processing functions responsible for text cleaning and normalization, and validating the output of the classification algorithm against a pre-labelled dataset of toxic comments.

2.5.2 Integrity Testing

Integration Testing is the subsequent phase of software testing in which a minimum of two modules of a system are combined and tested together. This process is primarily aimed at evaluating the interaction between different components of a system or between various parts of the system. In this phase, testing is conducted manually.

Test ID	Test Objective	Input	Expected Result
T1	Check the link between converting j.son files to CSV files	Dataset	Integration done successfully
T2	Training the model with the provided dataset.	500+ records	model is able to read the data and learn the pattern

Toxic Comments

In the case of "Classification of Toxic Comments using Machine Learning Algorithms," functional testing would involve testing the application's features and user interactions to ensure that they meet the functional requirements. For example, the functional testing could involve checking whether the application can correctly classify different types of toxic comments, such as racist, sexist, or vulgar comments. The functional tests could also check if the application's user interface is user-friendly, responsive, and easy to navigate.

Test Case Name	Test Case	Test Steps			Test Case Status	Test Priority
		Step	Expected	Actual		
upload Toxic comments dataset	Test whether the Toxic comments dataset is uploaded or not into the system	If the dataset may not uploaded.	we cannot do further operations	dataset uploaded we will do further operations	High	High
Apply count vectorizer	Test Whether the count vectorizer calculated or not	If the dataset may not uploaded.	we cannot do calculated	we can do calculated	High	High
Generate Machine-Learning models	Verify the Machine-learning Models generated or not	Without loading the dataset	cannot generate Machine learning Models	Machine learning Models generated successfully	High	High
Accuracy comparison graph	Verify the Accuracy comparison graph generated or not	Without loading the dataset	cannot Verify the Accuracy comparison graph	Verify the Accuracy comparison graph generated successfully	High	High
Predicting Online toxic comments	Verify online toxic comments predicted or not	Without loading the dataset	cannot predicted online toxic comments	predicted online toxic comments successfully	High	

2.5.3 USER TESTING

User testing refers to the process in which real users engage in specific tasks under realistic conditions to evaluate the functionality and features of a website, device, software, or service.

2.5.4 Software Maintenance

Operational testing is a type of testing that evaluates the system's performance, reliability, and maintainability in its operational environment. In the case of "Classification of Toxic Comments using Machine Learning Algorithms," operational testing would involve testing the application in a production-like environment to ensure that it meets the performance and reliability requirements. For example, operational testing could include measuring the response time of application, the ability to handle high traffic, and its scalability to handle increasing loads. It could also involve testing the application's maintainability, such as its ability to handle errors and exceptions and its logging and monitoring capabilities. Operational testing ensures that the application can perform effectively and efficiently in its operational environment, reduces the risk of system failures, and increases the overall reliability and maintainability of the application.

Chapter 3 - Implementation and Results

Deep learning algorithm is written in the Python language. The used libraries are:

- Pandas
- NumPy
- scikit-learn
- Matlab
- TensorFlow

We employ Microsoft Excel for the storage of both historical and contemporary fictitious data profiles. Subsequently, the algorithm organizes this information within a data frame.

3.1 Python: Python is acknowledged as a high-level and adaptable programming language, and it is widely embraced across various sectors. The most recent iteration, Python 3, is employed in multiple fields, such as web development and machine learning, in addition to being integral to numerous advanced technologies in the software industry. The Python programming language is particularly advantageous for beginners, while also catering to seasoned programmers familiar with other languages like C++ and Java.

3.2 Features of Python

Easy to read

Python is widely regarded as one of the most accessible programming languages due to its syntax, which prioritizes readability and simplicity. The language's clean and direct syntax enables programmers to compose code with fewer lines while maintaining clarity, thereby facilitating an understanding of the code's functionality. Furthermore, Python employs indentation to define code blocks, which enhances readability and allows programmers to easily discern the structure of their code. The inclusion of straightforward English keywords, such as "if," "else," "for," and "while," further contributes to its readability, minimizing the necessity for intricate syntax or symbols. Collectively, these design features render Python an excellent choice for both novices and

seasoned professionals, allowing them to write code swiftly, effectively, and with reduced ambiguity.

Object-Oriented Language

Python is a programming language that adheres to the principles of object-oriented programming (OOP). This paradigm focuses on the use of objects, which are instances of classes that encapsulate both data and functionality.

In Python, classes are defined using the class keyword and can contain attributes and methods. Attributes are the data members of the class, while methods are the functions that operate on the data.

GUI Programming Support

Python offers a variety of options for developing graphical user interfaces (GUIs), establishing it as a flexible language for desktop application creation. A widely used library for GUI programming in Python is Tkinter (TK), which is included with the majority of Python installations. Tk provides an easy-to-use and straightforward method for designing windows, buttons, text fields, menus, and other GUI elements. Additionally, it facilitates event-driven programming, enabling developers to react to user interactions such as mouse clicks and keyboard entries.

Other popular GUI libraries for Python include pygt, py-side, and wx-Python, which are all based on widely-used C++ GUI frameworks. These libraries offer more advanced features and better performance than TK but require additional setup and installation steps.

In addition to these libraries, Python also supports the development of web-based GUIs using popular web frameworks such as Django or Flask. These frameworks allow developers to build web applications with interactive user interfaces that can be accessed from any device with a web browser.

High-level Language

Python is regarded as a high-level programming language due to its emphasis on readability and ease of writing, prioritizing abstraction and simplicity. High-level programming languages are crafted to resemble human language more closely than

machine language. Consequently, such languages offer inherent abstractions, data types, and functions that facilitate programmers in articulating intricate concepts and tasks without the need to concern themselves with low-level aspects like memory management, machine architecture, or hardware-specific commands.

Python offers a variety of built-in functions and data types, including lists, dictionaries, and tuples, enabling developers to create efficient and succinct code while minimizing concerns about low-level implementation details. Furthermore, Python's reliance on whitespace and indentation to define code blocks enhances readability and comprehension, thereby decreasing the time and effort needed for debugging and maintenance.

Extensible feature

Python is an exceptionally extensible programming language that facilitates the incorporation of new features and capabilities. A primary method by which Python achieves extensibility is through modules, enabling the creation of self-sufficient code units that can be reused in various applications. Furthermore, Python accommodates packages, which are groups of related modules that can be structured in a hierarchical manner for improved organization. The standard library of Python is also notably extensible, offering a diverse array of functionalities for activities such as data manipulation, networking, and web development.

Python is a Portable Language

Python is regarded as a portable programming language due to its ability to run on diverse operating systems and architectures without necessitating substantial modifications to the code. This portability is attributed to the interpreted nature of Python, which eliminates the need for compilation into machine-specific code.

Furthermore, Python offers platform-agnostic support for a range of functionalities, including file input/output, networking, and threading. The Python standard library is universally accessible across all platforms, enabling developers to write code that leverages this library and execute it on any system that supports Python.

Additionally, Python's wide array of third-party libraries and frameworks are crafted to be cross-platform, facilitating developers in building applications that function seamlessly

across different environments. The existence of tools such as py-installer and cx-Freeze further enhances this capability, allowing for the creation of standalone applications that can operate on multiple platforms.

Interpreted language

Python is classified as an interpreted language, which indicates that it is executed by an interpreter instead of being transformed into machine code through compilation. When a Python program is executed, the interpreter processes each line of code sequentially, converting it into machine code in real-time and executing it immediately. This characteristic facilitates a more rapid development cycle and simplifies the process of testing code modifications, as the interpreter does not require recompilation of the entire program for each change. Furthermore, due to its interpreted nature, Python is platform-independent, allowing any Python-written program to run on any system equipped with a compatible interpreter, eliminating the necessity for additional compilation tailored to that specific system.

Large Standard Library

Python boasts an extensive and comprehensive standard library, which encompasses a diverse array of modules and packages that offer built-in assistance for numerous common programming activities. These modules provide a broad spectrum of functionalities, including file input/output, regular expressions, networking, database connectivity, and much more. The modules within the standard library are crafted to be portable, allowing them to function on any platform that supports Python. They are also rigorously tested and well-documented, facilitating developers in swiftly locating and utilizing the required functionalities. Additionally, the standard library is continuously advancing, with new modules and updates being introduced regularly to address the evolving demands of the Python community. This characteristic renders Python an exceptionally adaptable language, suitable for a variety of programming endeavors, ranging from web development and data analysis to scientific computing and machine learning.

Dynamically Typed Language

Python is classified as a dynamically typed language, indicating that the types of variables are established during runtime instead of being explicitly defined by the programmer. This makes Python very flexible and easy to use, as variables can be assigned values of different types without needing to specify their types beforehand. A variable in Python can be assigned an integer value on one line and subsequently a string value on another line without any complications. However, the dynamic typing nature of Python may result in errors that are not detected until runtime, particularly when attempting to execute operations on variables of differing types. Nonetheless, this dynamic typing feature is considered one of Python's advantages, as it facilitates quicker development cycles and enables the creation of code that is more readable and succinct.

Frontend and Backend Development

Python is a highly adaptable programming language that excels in both frontend and backend development. In the realm of frontend development, Python facilitates the creation of dynamic and interactive web applications through the use of well-established frameworks like Django and Flask. These frameworks offer a range of powerful tools that enable developers to construct web applications swiftly and effectively, featuring built-in database capabilities, user-friendly templates, and strong security protocols.

Python is highly regarded for backend development, particularly for server-side applications and APIs. Its ability to scale effectively enables it to handle significant traffic with ease, rendering it an excellent option for large-scale initiatives. Furthermore, Python's extensive standard library, along with a wide array of third-party packages, facilitates seamless integration with diverse technologies and services.

3.3 APPLICATIONS

Python is a versatile programming language used in a wide variety of applications, including:

1.**Web Development:** Python serves as a versatile tool for developing web applications, web frameworks, and content management systems. Prominent web frameworks available for Python include Django, Flask, Pyramid, and Bottle.

2.**Data Science:** In the realm of Data Science, Python is extensively utilized for data analysis, visualization, and machine learning. Libraries such as NumPy, Pandas,

Matplotlib, and Scikit-learn are frequently employed for various data science applications.

3.**Scientific Computing:** For Scientific Computing, Python finds application in disciplines like physics, chemistry, and biology. Notable scientific packages in Python include SciPy, Sym-Py, and Bio-python.

4.**Game Development:** In the field of Game Development, Python can be leveraged to design games and game engines. Py-game is a widely recognized library in Python that facilitates game development**.**

5.Desktop GUI: Python can be used to develop desktop graphical user interfaces (GUIs). Python's standard library includes the TK module for creating GUIs.

6.**Automation and Scripting:** Python can be used for automating repetitive tasks and scripting. It can also be used for system administration, network programming, and web scraping.

7.**Education:** Python is widely used in education for teaching programming and computer science concepts. Its simple syntax and readability make it easy to learn for beginners.

8.**Finance:** Python is used in finance for tasks such as financial modeling, risk management, and algorithmic trading.

9.**Audio and Video Processing:** Python can be used for audio and video processing tasks. Python libraries like Py-Dub and Movie-Py are used for audio and video manipulation.

10.**Internet of Things (IoT):** Python can be used in IoT for tasks such as sensor data analysis, device control, and data visualization.

3.4 Python Libraries

Pandas

Pandas is a prominent open-source library utilized for data manipulation and analysis within the Python programming language. It offers user-friendly and robust data structures designed for managing structured or tabular data, including spreadsheets and SQL tables. With Pandas, users can effectively perform a range of tasks associated with data analysis, transformation, and cleaning, such as filtering, aggregation, sorting, and merging datasets. Additionally, it facilitates data visualization and seamless integration with other libraries, rendering it a favored option among data scientists and analysts engaged in data-centric projects.

Matplotlib

Matplotlib serves as a robust data visualization library for Python. It empowers users to create high-quality visual representations in both 2D and 3D formats, such as histograms, scatter plots, bar charts, and more. Users can tailor the aesthetics of their visualizations by adjusting elements like axis labels, legends, font sizes, and line colors. The library supports multiple backends, including PDF, SVG, and PNG formats. As an open-source tool, Matplotlib is extensively utilized in scientific computing, data analysis, and machine learning contexts. It is particularly favored for producing publication-quality graphs and charts, and its comprehensive documentation, along with a large user community, facilitates easy access to support and examples online. In summary, Matplotlib is an essential resource for anyone aiming to develop engaging visualizations of their data using Python.

3.5 Machine Learning

Machine learning represents a subset of artificial intelligence (AI) focused on developing algorithms and models that empower computers to learn from data, thereby enhancing their capacity to make predictions or decisions. Essentially, machine learning entails the formulation of computer programs capable of analyzing extensive datasets to identify patterns or rules, enabling them to make forecasts or take actions based on new information.

Types

Machine learning algorithms can be categorized into three primary types:

Supervised Learning: This category involves training a model using labeled data, where the model learns to identify patterns and correlations between the input data and the corresponding output labels.

Unsupervised Learning: Unlike supervised learning, unsupervised learning algorithms operate on data that lacks labeled targets. The objective of unsupervised learning is to uncover structures and patterns within the data independently of predefined labels. Common forms of unsupervised learning include clustering, which organizes similar data points, and dimensionality reduction, which seeks to streamline high-dimensional datasets.

Reinforcement Learning: This algorithm is employed to instruct a model in decision-making through a process of trial and error. The model engages with its environment and acquires knowledge via positive and negative reinforcement. Reinforcement learning is frequently applied in scenarios such as gaming or robotics, where the model is required to master navigation within a complex setting.

3.6 MAIN CODE

```
from tkinter import messagebox from tkinter import * from
tkinter import simpledialog import tkinter import
matplotlib.pyplot as plt import numpy as np from tkinter
import ttk from tkinter import filedialog import pandas as
pd from sklearn.model_selection import train_test_split
from string import punctuation from nltk.corpus import
stopwords import nltk from nltk.stem import
WordNetLemmatizer from sklearn.feature_extraction.text
import TfidfVectorizer from sklearn.metrics import
accuracy_score from sklearn import svm from
sklearn.naive_bayes import GaussianNB from
sklearn.ensemble import RandomForestClassifier from
sklearn.linear_model import LogisticRegression from
sklearn.neighbors import KNeighborsClassifier from
sklearn.tree import DecisionTreeClassifier from
sklearn.metrics import hamming_loss

main = Tk()
main.title("Classification of Online Toxic Comments Using Machine Learning
Algorithms") main.geometry("1300x1200")

global filename global X, Y1, Y2, Y3,
Y4, Y5, Y6 accuracy = [] global
X_train, X_test, y_train, y_test loss =
[]
```

```
textdata = [] global
classifier1,classifier2,classifier3,classifier4,classifier5,classifier6
global tfidf_vectorizer

def cleanPost(doc): tokens = doc.split() table =
str.maketrans('', '', punctuation) tokens = [w.translate(table)
for w in tokens] tokens = [word for word in tokens if
word.isalpha()] tokens = [w for w in tokens if not w in
stop_words] tokens = [word for word in tokens if len(word)
> 1] tokens = [lemmatizer.lemmatize(token) for token in
tokens] tokens = ' '.join(tokens) return tokens

def uploadDataset(): global filename text.delete('1.0',
END) filename =
filedialog.askopenfilename(initialdir="Dataset")
text.insert(END,filename+" loaded\n")


def preprocess():
textdata.clear()
  global X, Y1, Y2, Y3, Y4, Y5, Y6
  Y1 = []
  Y2 = []
  Y3 = []
  Y4 = []
```

```
   Y5 = []     Y6 = []     text.delete('1.0', END)     dataset =
pd.read_csv(filename,encoding='iso-8859-1',nrows = 300)     for i in
range(len(dataset)):
       msg = dataset._get_value(i, 'comment_text')
toxic = dataset._get_value(i, 'toxic')        severe_toxic =
dataset._get_value(i, 'severe_toxic')        obscene =
dataset._get_value(i, 'obscene')        threat =
dataset._get_value(i, 'threat')        insult =
dataset._get_value(i, 'insult')        identity_hate =
dataset._get_value(i, 'identity_hate')        msg =
str(msg)
       msg = msg.strip().lower()
Y1.append(int(toxic))
       Y2.append(int(severe_toxic))
       Y3.append(int(obscene))
       Y4.append(int(threat))
       Y5.append(int(insult))
Y6.append(int(identity_hate))        clean
= cleanPost(msg)
textdata.append(clean)
text.insert(END,clean+"\n")
   Y1 = np.asarray(Y1)
   Y2 = np.asarray(Y2)
   Y3 = np.asarray(Y3)
   Y4 = np.asarray(Y4)
   Y5 = np.asarray(Y5)
   Y6 = np.asarray(Y6)
```

```
def countVector():
    global X, Y1, Y2, Y3, Y4, Y5, Y6     global tfidf_vectorizer     global
X_train, X_test, y_train, y_test     text.delete('1.0', END)
stopwords=stopwords = nltk.corpus.stopwords.words("english")
tfidf_vectorizer = TfidfVectorizer(stop_words=stopwords, use_idf=True,
smooth_idf=False, norm=None, decode_error='replace') tfidf =
tfidf_vectorizer.fit_transform(textdata).toarray()         df = pd.DataFrame(tfidf,
columns=tfidf_vectorizer.get_feature_names_out()) text.insert(END,str(df))

    print(df.shape)
df = df.values
    X = df[:, 0:df.shape[1]]    indices
=                 np.arange(X.shape[0])
np.random.shuffle(indices)
    X = X[indices]
    Y1 = Y1[indices]
    Y2 = Y2[indices]
    Y3 = Y3[indices]
    Y4 = Y4[indices]
    Y5 = Y5[indices]
    Y6 = Y6[indices]
    X_train, X_test, y_train, y_test = train_test_split(X, Y1,
test_size=0.2,random_state=0)     text.insert(END,"\n\nTotal Comments found
in dataset : "+str(len(X))+"\n")     text.insert(END,"Total records used to train
machine learning algorithms :
"+str(len(X_train))+"\n")     text.insert(END,"Total records used to test
machine learning algorithms  : "+str(len(X_test))+"\n")


def train(Xdata,Ydata,cls):
```

```
   X_train, X_test, y_train, y_test = train_test_split(Xdata, Ydata,
test_size=0.2,random_state=0)  cls.fit(X_train, y_train)  predict
= cls.predict(X_test)  acc = accuracy_score(y_test,predict)*100
loss = hamming_loss(y_test,predict)*100 return acc,loss


def train1(Xdata,Ydata,cls):
   X_train, X_test, y_train, y_test = train_test_split(Xdata, Ydata,
test_size=0.2,random_state=0)  cls.fit(Xdata, Ydata)  predict =
cls.predict(X_test)      acc = accuracy_score(y_test,predict)*100
loss = hamming_loss(y_test,predict)*100    return acc,loss


def runSVM():
   global X, Y1, Y2, Y3, Y4, Y5, Y6
text.delete('1.0', END)
accuracy.clear()   loss.clear()
cls1 = svm.SVC()   acc1,loss1 =
train(X,Y1,cls1)   cls2 =
svm.SVC()   acc2,loss2 =
train(X,Y2,cls2)   cls3 =
svm.SVC()   acc3,loss3 =
train(X,Y3,cls3)   cls4 =
svm.SVC()   acc4,loss4 =
train(X,Y4,cls4) cls5 = svm.SVC()
acc5,loss5 = train(X,Y5,cls5) cls6 =
svm.SVC()
```

```
    acc6,loss6 = train(X,Y6,cls6) acc = acc1 +
    acc2 + acc3 + acc4 + acc5 + acc6 acc = (acc /
    600) * 100
    lossValue = loss1 + loss2 + loss3 + loss4 + loss5 + loss6
accuracy.append(acc)     lossValue = lossValue / 600
loss.append(lossValue)     text.insert(END,"SVM Accuracy :
"+str(acc)+"\n")     text.insert(END,"SVM Hamming Loss :
"+str(lossValue)+"\n\n")


def runLR():
    global X, Y1, Y2, Y3, Y4, Y5, Y6     cls1 =
LogisticRegression()     acc1,loss1 =
train(X,Y1,cls1)     cls2 = LogisticRegression()
acc2,loss2 = train(X,Y2,cls2)     cls3 =
LogisticRegression()     acc3,loss3 =
train(X,Y3,cls3)     cls4 = LogisticRegression()
acc4,loss4 = train(X,Y4,cls4)     cls5 =
LogisticRegression()     acc5,loss5 =
train(X,Y5,cls5)     cls6 = LogisticRegression()
acc6,loss6 = train(X,Y6,cls6)     acc = acc1 +
acc2 + acc3 + acc4 + acc5 + acc6     acc = (acc /
600) * 100
    accuracy.append(acc) lossValue = loss1 + loss2 + loss3
    + loss4 + loss5 + loss6 lossValue = lossValue / 600
    loss.append(lossValue)     text.insert(END,"Logistic Regression
Accuracy : "+str(acc)+"\n")     text.insert(END,"Logistic Regression
Hamming Loss : "+str(lossValue)+"\n\n")
```

```
def runNB():
    global X, Y1, Y2, Y3, Y4, Y5, Y6     cls1 =
GaussianNB()     acc1,loss1 = train(X,Y1,cls1)     cls2 =
GaussianNB()     acc2,loss2 = train(X,Y2,cls2)     cls3 =
GaussianNB()     acc3,loss3 = train(X,Y3,cls3)     cls4 =
GaussianNB()     acc4,loss4 = train(X,Y4,cls4)     cls5 =
GaussianNB()     acc5,loss5 = train(X,Y5,cls5)     cls6 =
GaussianNB()     acc6,loss6 = train(X,Y6,cls6)     acc =
acc1 + acc2 + acc3 + acc4 + acc5 + acc6     acc = (acc /
600) * 100     accuracy.append(acc)     lossValue = loss1 +
loss2 + loss3 + loss4 + loss5 + loss6     lossValue =
lossValue / 600
    loss.append(lossValue) text.insert(END,"Naive Bayes Accuracy :
    "+str(acc)+"\n") text.insert(END,"Naive Bayes Hamming Loss :
    "+str(lossValue)+"\n\n")

def runDecisionTree():
    global X, Y1, Y2, Y3, Y4, Y5, Y6     cls1 = DecisionTreeClassifier()
acc1,loss1 = train(X,Y1,cls1)     cls2 = DecisionTreeClassifier()
acc2,loss2 = train(X,Y2,cls2)     cls3 = DecisionTreeClassifier()
acc3,loss3 = train(X,Y3,cls3)     cls4 = DecisionTreeClassifier()
acc4,loss4 = train(X,Y4,cls4)     cls5 = DecisionTreeClassifier()
acc5,loss5 = train(X,Y5,cls5)     cls6 = DecisionTreeClassifier()
acc6,loss6 = train(X,Y6,cls6)     acc = acc1 + acc2 + acc3 + acc4 + acc5 +
acc6     acc = (acc / 600) * 100     accuracy.append(acc)     lossValue = loss1
+ loss2 + loss3 + loss4 + loss5 + loss6     lossValue = lossValue / 600
loss.append(lossValue)     text.insert(END,"Decision Tree Accuracy :
```

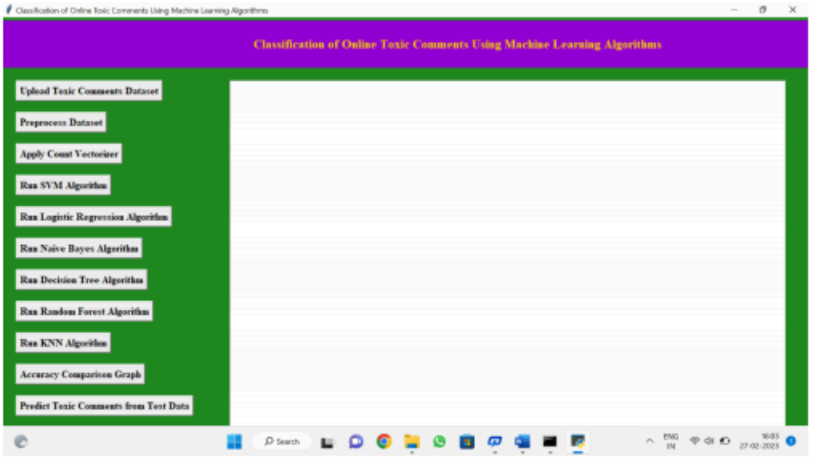
```
"+str(acc)+"\n")     text.insert(END,"Decision Tree Hamming Loss :
"+str(lossValue)+"\n\n") def runRandomForest():

   global classifier1,classifier2,classifier3,classifier4,classifier5,classifier6
global X, Y1, Y2, Y3, Y4, Y5, Y6     cls1 = RandomForestClassifier()
acc1,loss1 = train1(X,Y1,cls1)     cls2 = RandomForestClassifier()
acc2,loss2 = train1(X,Y2,cls2)     cls3 = RandomForestClassifier()
acc3,loss3 = train1(X,Y3,cls3)     cls4 = RandomForestClassifier()
acc4,loss4 = train1(X,Y4,cls4)     cls5 = RandomForestClassifier()
acc5,loss5 = train1(X,Y5,cls5)     cls6 = RandomForestClassifier()
acc6,loss6 = train1(X,Y6,cls6)     classifier1 = cls1     classifier2 = cls2
classifier3 = cls3     classifier4 = cls4     classifier5 = cls5     classifier6 =
cls6     acc = acc1 + acc2 + acc3 + acc4 + acc5 + acc6     acc = (acc / 600)
* 100     accuracy.append(acc)     lossValue = loss1 + loss2 + loss3 +
loss4 + loss5 + loss6     lossValue = lossValue / 600
loss.append(lossValue)

   text.insert(END,"Random Forest Accuracy : "+str(acc)+"\n")
text.insert(END,"Random Forest Hamming Loss : "+str(lossValue)+"\n\n")

def runKNN():
   global X, Y1, Y2, Y3, Y4, Y5, Y6     cls1 =
KNeighborsClassifier(n_neighbors = 2)     acc1,loss1 =
train(X,Y1,cls1)     cls2 = KNeighborsClassifier(n_neighbors = 2)
acc2,loss2 = train(X,Y2,cls2)     cls3 =
KNeighborsClassifier(n_neighbors = 2)     acc3,loss3 =
train(X,Y3,cls3)     cls4 = KNeighborsClassifier(n_neighbors = 2)
acc4,loss4 = train(X,Y4,cls4)     cls5 =
KNeighborsClassifier(n_neighbors = 2)     acc5,loss5 =
train(X,Y5,cls5)     cls6 = KNeighborsClassifier(n_neighbors = 2)
```

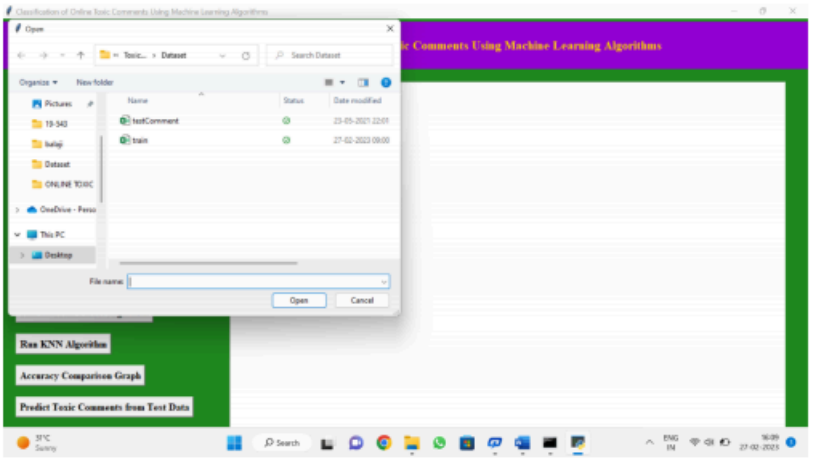
```
acc6,loss6 = train(X,Y6,cls6)     acc = acc1 + acc2 + acc3 + acc4 +
acc5 + acc6     acc = (acc / 600) * 100     accuracy.append(acc)
lossValue = loss1 + loss2 + loss3 + loss4 + loss5 + loss6
lossValue = lossValue / 600     loss.append(lossValue)
text.insert(END,"KNN Accuracy : "+str(acc)+"\n")
text.insert(END,"KNN Hamming Loss : "+str(lossValue)+"\n\n")
def graph():
   df = pd.DataFrame([['SVM','Accuracy',accuracy[0]],['SVM','Hamming
Loss',loss[0]],
                      ['Logistic Regression','Accuracy',accuracy[1]],['Logistic
Regression','Hamming Loss',loss[1]],
                      ['Naive Bayes','Accuracy',accuracy[2]],['Naive Bayes','Hamming
Loss',loss[2]],
                      ['Decision Tree','Accuracy',accuracy[3]],['Decision
Tree','Hamming Loss',loss[3]],
                      ['Random Forest','Accuracy',accuracy[4]],['Random
Forest','Hamming Loss',loss[4]],
                      ['KNN','Accuracy',accuracy[5]],['KNN','Hamming Loss',loss[5]],

                    ],columns=['Parameters','Algorithms','Value'])
df.pivot("Parameters", "Algorithms", "Value").plot(kind='bar')     plt.show()


def predict():
   global
classifier1,classifier2,classifier3,classifier4,classifier5,classifier6
testfile = filedialog.askopenfilename(initialdir="Dataset")     testData =
pd.read_csv(testfile)     text.delete('1.0', END)     for i in
range(len(testData)):
```

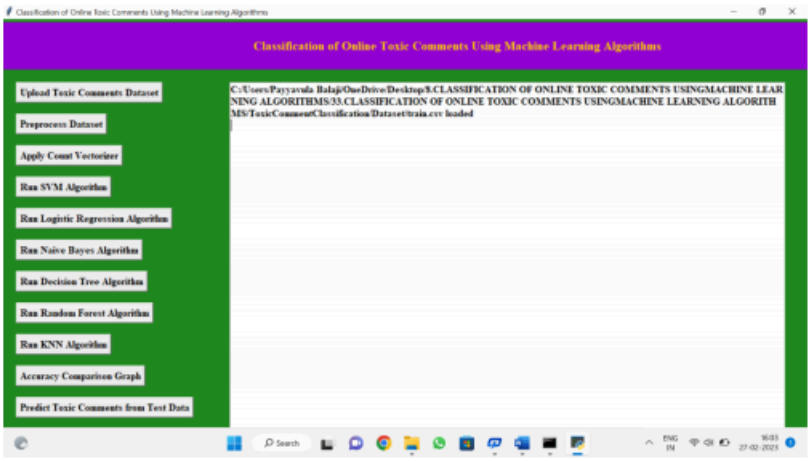
```
msg = testData._get_value(i, 'comment') review =
msg.lower() review = review.strip().lower() review =
cleanPost(review) testReview =
tfidf_vectorizer.transform([review]).toarray() predict1 =
classifier1.predict(testReview) predict2 =
classifier2.predict(testReview) predict3 =
classifier3.predict(testReview) predict4 =
classifier4.predict(testReview) predict5 =
classifier5.predict(testReview) predict6 =
classifier6.predict(testReview) if predict1 == 0 and
predict2 == 0 and predict3 == 0 and predict4 == 0 and predict5
== 0 and predict6 == 0:
    text.insert(END,msg+" [NOT CONTAINS TOXIC Comments]\n\n")
elif predict1 == 1:
    text.insert(END,msg+" [CONTAINS TOXIC Comments]\n\n")
elif predict2 == 1:
    text.insert(END,msg+" [CONTAINS SEVERE_TOXIC
Comments]\n\n")
  elif predict3 == 1:
    text.insert(END,msg+" [CONTAINS OBSCENE Comments]\n\n")
elif predict4 == 1:
    text.insert(END,msg+" [CONTAINS THREAT Comments]\n\n")
elif predict5 == 1:
    text.insert(END,msg+" [CONTAINS INSULT Comments]\n\n")
elif predict6 == 1:
    text.insert(END,msg+" [CONTAINS IDENTITY_HATE
Comments]\n\n")
```

```
font = ('times', 15, 'bold')
title = Label(main, text='Classification of Online Toxic Comments Using
Machine Learning Algorithms') title.config(bg='darkviolet', fg='gold')
title.config(font=font)           title.config(height=3,
width=120)       title.place(x=0,y=5)

font1 = ('times', 13, 'bold') ff
= ('times', 12, 'bold')

uploadButton = Button(main, text="Upload Toxic Comments Dataset",
command=uploadDataset) uploadButton.place(x=20,y=100)
uploadButton.config(font=ff)


processButton = Button(main, text="Preprocess Dataset",
command=preprocess) processButton.place(x=20,y=150)
processButton.config(font=ff)

cvButton = Button(main, text="Apply Count Vectorizer",
command=countVector) cvButton.place(x=20,y=200)
cvButton.config(font=ff)

svmButton = Button(main, text="Run SVM Algorithm", command=runSVM)
svmButton.place(x=20,y=250) svmButton.config(font=ff)

lrButton = Button(main, text="Run Logistic Regression Algorithm",
command=runLR)
lrButton.place(x=20,y=300) lrButton.config(font=ff)
```

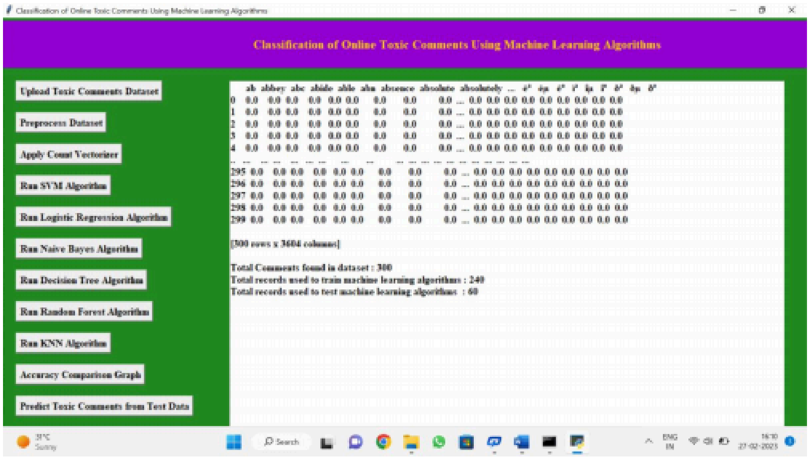
```
nbButton = Button(main, text="Run Naive Bayes Algorithm",
command=runNB) nbButton.place(x=20,y=350)
nbButton.config(font=ff)

dtButton = Button(main, text="Run Decision Tree Algorithm",
command=runDecisionTree) dtButton.place(x=20,y=400)
dtButton.config(font=ff)

rfButton = Button(main, text="Run Random Forest Algorithm",
command=runRandomForest) rfButton.place(x=20,y=450)
rfButton.config(font=ff)

knnButton = Button(main, text="Run KNN Algorithm", command=runKNN)
knnButton.place(x=20,y=500) knnButton.config(font=ff)

graphButton = Button(main, text="Accuracy Comparison Graph",
command=graph) graphButton.place(x=20,y=550)
graphButton.config(font=ff)

predictButton = Button(main, text="Predict Toxic Comments from Test Data",
command=predict)
predictButton.place(x=20,y=600) predictButton.config(font=ff)


font1 = ('times', 12, 'bold')
text=Text(main,height=30,width=110)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=360,y=100) text.config(font=font1)
```

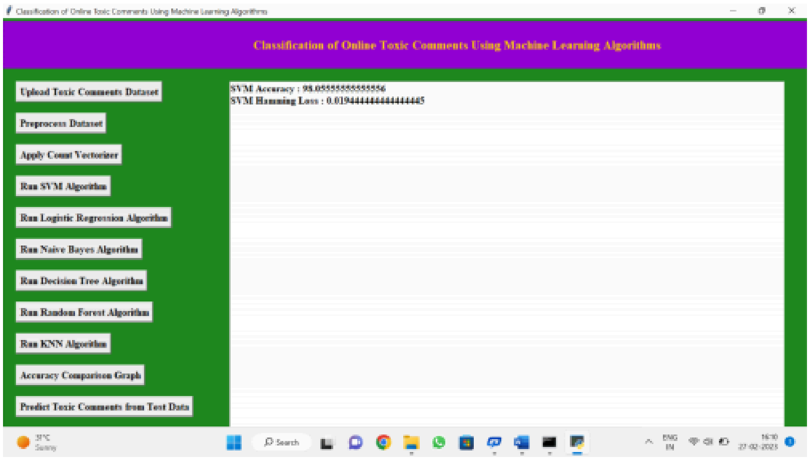

Classification of Online Toxic Comments Using Machine Learning Algorithms
Upload Toxic Comments Dataset
Preprocess Dataset
Apply Count Vectorizer
Run SVM Algorithm
Run Logistic Regression Algorithm
Run Naive Bayes Algorithm
Run Decision Tree Algorithm
Run Random Forest Algorithm
Run KNN Algorithm
Accuracy Comparison Graph
Predict Toxic Comments from Test Data


Open
Toxic... > Dataset
Search Dataset
Organize
New folder
Pictures
19-543
balaji
Dataset
ONLINE TOXIC
OneDrive - Perso
This PC
Desktop
Name
Status
Date modified
testComment
train
File name:
Open
Cancel
Run KNN Algorithm
Accuracy Comparison Graph
Predict Toxic Comments from Test Data


Classification of Online Toxic Comments Using Machine Learning Algorithms
Classification of Online Toxic Comments Using Machine Learning Algorithms
Upload Toxic Comments Dataset
Preprocess Dataset
Apply Count Vectorizer
Run SVM Algorithm
Run Logistic Regression Algorithm
Run Naive Bayes Algorithm
Run Decision Tree Algorithm
Run Random Forest Algorithm
Run KNN Algorithm
Accuracy Comparison Graph
Predict Toxic Comments from Test Data
C:/Users/Payyavula Balaji/OneDrive/Desktop/8.CLASSIFICATION OF ONLINE TOXIC COMMENTS USINGMACHINE LEARNING ALGORITHMS/33.CLASSIFICATION OF ONLINE TOXIC COMMENTS USINGMACHINE LEARNING ALGORITHMS/ToxicCommentClassification/Dataset/train.csv loaded


Classification of Online Toxic Comments Using Machine Learning Algorithms
Classification of Online Toxic Comments Using Machine Learning Algorithms
Upload Toxic Comments Dataset
Preprocess Dataset
Apply Count Vectorizer
Run SVM Algorithm
Run Logistic Regression Algorithm
Run Naive Bayes Algorithm
Run Decision Tree Algorithm
Run Random Forest Algorithm
Run KNN Algorithm
Accuracy Comparison Graph
Predict Toxic Comments from Test Data
explanation edits made username hardcore metallica fan reverted werent vandalism closure gas voted new york doll fac please dont
remove template talk page since im retired
daww match background colour im seemingly stuck thanks talk january utc
hey man im really trying edit war guy constantly removing relevant information talking edits instead talk page seems care formatting
actual info
cant make real suggestion improvement wondered section statistic later subsection type accident think reference may need tidying e
xact format ie date format etc later noone else first preference formatting style reference want please let know appears backlog artic
le review guess may delay reviewer turn listed relevant form eg wikipediagoodarticlenominationstransport
sir hero chance remember page thats
congratulation well use tool well talk
cocksucker piss around work
vandalism matt shirvington article reverted please dont banned
sorry word nonsense offensive anyway im intending write anything articlewow would jump vandalism im merely requesting encyclope
dic one use school reference selective breeding page almost stub point animal breeding short messy article give info must someone a
round expertise eugenics
alignment subject contrary dulithgow
fair use rationale imagewonjqjg thanks uploading imagewonjqjg notice image page specifies image used fair use explanation ration
ale use wikipedia article constitutes fair use addition boilerplate fair use template must also write image description page specific ex
planation rationale using image article consistent fair use please go image description page edit include fair use rationale uploaded fa
ir use medium consider checking specified fair use rationale page find list image page edited clicking contribution link located top wi
kipedia page logged selecting image dropdown box note fair use image uploaded may lacking explanation deleted one week uploaded
described criterion speedy deletion question please ask medium copyright question page thank talk contribs unspecified source ima
gewonjqjg thanks uploading imagewonjqjg noticed file description page currently doesnt specify created content copyright status u
nclear create file need specify owner copyright obtained website link website taken together restatement website term use content u
sually sufficient information however copyright holder different website publisher copyright also acknowledged well adding source pl
ease add proper copyright licensing tag file doesnt one already createdtook picture audio video tag used release gfdl believe medium
meet criterion wikipediafair use use tag one tag listed wikipediaimage copyright tagsfair use see wikipediaimage copyright tag full li
t copyright tag use uploaded file consider checking specified source tagged find list file uploaded following link unsourced untagged i
mages may deleted one week tanned described criterion unsedy deletion images copyrighted nonfree license see wikiedisfair use ima


Classification of Online Toxic Comments Using Machine Learning Algorithms
Upload Toxic Comments Dataset
Preprocess Dataset
Apply Count Vectorizer
Run SVM Algorithm
Run Logistic Regression Algorithm
Run Naive Bayes Algorithm
Run Decision Tree Algorithm
Run Random Forest Algorithm
Run KNN Algorithm
Accuracy Comparison Graph
Predict Toxic Comments from Test Data
ab abbey abc abide able abu absence absolute absolutely ...
[300 rows x 3604 columns]
Total Comments found in dataset : 300
Total records used to train machine learning algorithms : 240
Total records used to test machine learning algorithms : 60


Classification of Online Toxic Comments Using Machine Learning Algorithms
Upload Toxic Comments Dataset
Preprocess Dataset
Apply Count Vectorizer
Run SVM Algorithm
Run Logistic Regression Algorithm
Run Naive Bayes Algorithm
Run Decision Tree Algorithm
Run Random Forest Algorithm
Run KNN Algorithm
Accuracy Comparison Graph
Predict Toxic Comments from Test Data
SVM Accuracy : 98.05555555555556
SVM Hamming Loss : 0.019444444444444445


Classification of Online Toxic Comments Using Machine Learning Algorithms
Upload Toxic Comments Dataset
Preprocess Dataset
Apply Count Vectorizer
Run SVM Algorithm
Run Logistic Regression Algorithm
Run Naive Bayes Algorithm
Run Decision Tree Algorithm
Run Random Forest Algorithm
Run KNN Algorithm
Accuracy Comparison Graph
Predict Toxic Comments from Test Data
COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK [CONTAINS TOXIC Comments]
How could I post before the block expires? The funny thing is you think I'm being uncivil! [NOT CONTAINS TOXIC Comments]
REDIRECT Talk:Frank Herbert Mason [NOT CONTAINS TOXIC Comments]
Tony Sidaway is obviously a fistfuckee. He loves an arm up his ass [CONTAINS TOXIC Comments]

Chapter 4 - Conclusion and Future Enhancements

4.1 Summary of work done

In this project, we utilize machine learning methodologies that incorporate artificial neural networks (ANN) to evaluate the probability of a friend request being authentic or deceptive. Every equation associated with each neuron (node) is processed using a Sigmoid function. This would allow the presented deep learning algorithm to learn the patterns of bot behaviour by backpropagation, minimizing the final cost function and adjusting each neuron's weight and bias. We examine the Sigmoid function and the methodology for determining and utilizing the weights.

The use of artificial neural networks to identify fake profiles is a promising solution to the problem of fake profiles on social media platforms. This feasibility study aims to evaluate the feasibility of using artificial neural networks to identify fake profiles. The study will provide valuable insights into the types of data required to train an artificial neural network, the availability and quality of the data, the technical feasibility of implementing a network, and the potential benefits and drawbacks of using such a network. The results of this study will be useful in determining the viability of using artificial neural networks to identify fake profiles.

In summary, the task of employing machine learning algorithms to classify harmful comments presents significant challenges that require a comprehensive approach involving data preparation, feature engineering, and model selection. Various machine learning techniques, such as logistic regression, decision trees, random forests, and neural networks, have demonstrated their effectiveness in identifying harmful comments through numerous studies and experiments. The precision of classification models can be enhanced through data preparation methods, including tokenization, removal of stop words, stemming, and lemmatization. Additionally, the application of pre-trained word embeddings, such as Word2Vec, can further improve model performance. Feature engineering plays a crucial role in enhancing model accuracy by selecting the most relevant features and discarding those that are less informative. Techniques like TF-IDF and n-grams have proven beneficial for feature extraction from textual data.

It has been widely recognized that social networking platforms represent the future of communication. They facilitate connections among individuals in a highly effective and efficient manner, allowing voices and opinions to reach thousands of users instantaneously. Therefore, there is no doubt that social media is a permanent fixture in our society and will continue to be utilized extensively by the public in the years to come.

• The most reliable method to determine whether a profile is authentic or fraudulent is to examine its characteristics. For example, one should assess the profile picture, username, follower count, following count, status updates, biography, and other relevant details. With this in mind, we have categorized the profile features into groups of three, arranged in ascending order. This grouping is designed to enhance the model's accuracy. Consequently, future APIs can select a suitable algorithm to incorporate into their systems for the purpose of identifying and filtering out fake profiles. This approach will render the method highly effective. Online social networks (OSNs) can adopt an algorithm that aligns with their specific features, thereby creating a more secure environment for users.

Such measures can significantly reduce the prevalence of cybercrimes that frequently occur on social networking platforms.

4.2 Proposal/scope of future enhancement

Our model can also be made mandatory as a safety feature for new users or young adults such as teenagers. According to a study, social media is the reason for mental health issues in young adolescents. This can be successfully prevented by protecting them from fake profiles that are involved in cyber bullying and at times are also known to promote self-harm and trigger anxiety.

1. Integrating the algorithm with highest accuracy with API of new OSN or an existing one.
2. Lateral increase of versatility of developed model over different platforms.
3. Promoting it as a default safety feature and a mandate for new/young users.

References:

[1] “Detection of fake profile in online social network using machine” by Naman Singh et al (ICACCE-2018)

[2] “Use of artificial neural networks to identify fake profiles” by Gergo Hajdu (IEEE Journal -2017)

[3] “Analysis and detection of fake profile over social network” by Dr. Vijay Tiwari (ICCA 2017)

[4] “Towards detecting compromised accounts on social network” by Manuel Egele et al (IEEE 2017)

[5] “Using machine learning to detect fake identities: Bots vs. Humans” by Estee Wan der Walt and Jan Eloff (IEEE Access-2019)

[6] “Friend or foe? fake profile identification in online social networks” by Michael Fire et al (Springer Journal 2012)

[7] “Profile characteristics of fake twitter accounts” by Supraja Gurajala et al. (Big Data & Society 2016)

[8] “Mining Anonymity: Identifying Sensitive Accounts on Twitter” by Sai Teja Peddinti, Keith W. Ross†, Justin Cappos (International AAAI Conference on Web and Social Media (ICWSM), 2016)

[9] C. Xiao, D. M. Freeman, and T. Hwa, “Detecting clusters of fake accounts in online social networks,” in Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security. ACM, Conference Proceedings, pp. 91– 101.

[10] T. Tuna, E. Akbas, A. Aksoy, M. A. Canbaz, B. Gonen, and R. Aygun, “User characterization for online social networks,” Social Network Analysis and Mining, vol. 6, no. 1, p. 104, 2016.

[11] F. Benvenuto, G. Magno, T. Rodrigues and V. Almedia,” Detecting Spammers on twitters,”. Email Anti-Spam, 2010, vol. 6.p. 12.

[12] K. Lee, and S. Webb, ” Uncovering social honeypots + machine learning,” in Proc. Int. ACM SIGIR Conf. Res. Develop. Inform. Retrieval, 2010, pp. 435–442.

[13] S. Lee and J. Kim, “WarningBird: Detecting suspicious URLs in twitter stream,” in NDSS Symposium, 2012.

[14] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, “Design and evaluation of a real-time URL spam filtering service,” in Proc. IEEE Symp. Security Privacy, 2011, pp. 447–462.

[15] M. Egele , Gianuluca Stringhini , C. Kruegel, and G. Vigna, “COMPA: Detecting compromised accounts on social networks,” presented at the Network and Distributed SystemSecurity Symp., SanDiego, CA, USA Feb. 2013.

[16] W. B. Cavnar and J. M. Trenkle, “N-gram-based text categorization,” in Proc. 3rd Annu. Symp. Document Anal. Inform. Retrieval, 1994, pp. 161–175.

[17] J. C. Platt, “Fast training of support vector machines using sequential minimal optimization,” in Advances in Kernel Methods—Support Vector Learning. Cambridge, MA, USA: MIT Press, 1998, pp. 185– 208.

[18] H. M. Saleem, K.P. Dillon, S. Benesch, and D. Ruths, “A Web of Hate: Tackling Hateful Speech in Online Social Spaces,” 2017, [Online] . Available: http://arxiv.org/abs/1709.10159.

[19] M. Duggan, “Online harassment 2017,” Pew Res., pp.1–85,2017 doi:202.419.4372.

[20] M. A. Walker, Anand, J. E. F. Tree, R. Abbott, and J. King, “A corpus for research on deliberation and debate,” P roc. 8th Int. Conf. Lang. Resour. Eval. Lr. 2012, pp. 812–817, 2012.

[21] J. Cheng, C. Danescu-Niculescu-Mizil, and J. Leskovec, “ Antisocial behaviour in online discussion communities,” P roc. 9th Int. Conf. Web Soc. Media, ICWSM 2015, pp. 61–70, 2015.

[22] B. Mat hew et al., “Thou shalt not hate: Countering online hate speech,” Proc. 13th Int. Conf. Web Soc. Media, ICWSM 2019, no. August, pp. 369–380, 2019.

[23] C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang, “Abusive language detection in online user content,” 25th Int World Wide Web Conf. WWW 2016, pp. 145–153, 2016, Doi: 10.1145/2872427.2883062.

[24] E.K. Ikonomakis, S. Kot siantis, and V. T. Ampakas, “Text Classification Using Machine Learning Techniques,” no. August, 2005.

[25] M.R. Murt y, J.V.. Murthy, and P. Reddy P.V.G.D, “Text Document Classification based on Least Square Support Vector Machines with Singular Value Decomposition,” Int. J. Comput. Appl., vol. 27, no. 7, pp. 21–26, 2011, Doi:10.5120/3312-4540.

[26] E. Wulczyn, N. Thain, and L. Dixon, “Ex machina: Personal attacks seen at scale,” 26th Int. World Wide Web Conf. WWW 2017, pp. 1391–1399, 2017, Doi: 10.1145/3038912.3052591.