COMMENT TOXICITY DETECTION USING LSTM

A mini project report submitted for the partial fulfillment of the requirement for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

Submitted

By

V. Naveen (Reg. No. 21691F00A6)

Under the Guidance of

Dr. N. Naveen Kumar

Associate Professor,

Department of Computer Applications

DEPARTMENT OF COMPUTER APPLICATIONS

MADANAPALLE INSTITUTE OF TECHNOLOGY AND SCIENCE
MADANAPALLE

(UGC AUTONOMOUS)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)

www.mits.ac.in

2022 – 2024

MADANAPALLE INSTITUTE OF TECHNOLOGY AND SCIENCE

MADANAPALEE

(UGC AUTONOMOUS)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)

DEPARTMENT OF COMPUTER APPLICATIONS

BONAFIDE CERTIFICATE

This is to certify that the Mini project work entitled is a bonafide work carried out by **V. Naveen** (R**eg. No. 21691F00A6),** submitted in partial fulfillment of the requirements for the award of the degree of **Master of Computer Applications** in **Madanapalle Institute of Technology and Science, MADANAPALLE**, affiliated to **Jawaharlal Nehru Technological University, Anantapuramu** during the **academic year 2023-2024.**

PROJECT GUIDE	**HEAD OF THE DEPARTMENT**
Dr. N. Naveen Kumar	**Dr. N. Naveen Kumar**, MCA,
Ph.D.Associate Professor	Associate Professor & H.o.D
Department of Computer Applications	Department of Computer Applications

DECLARATION

I, **V. Naveen** , (**RollNo.21691F00A6)** hereby declare that the project entitles **“COMMENT TOXICITY USING LSTM**” is done by us under the guidance of **Dr. N. Naveen Kumar ,** **Associate Professor**, submitted in partial fulfillment of the requirements for the award of the degree of Master of Computer Applications at **MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE, MADANAPALLE,** affiliated to **Jawaharlal Nehru Technological University, Anantapuramu,** during the academic year 2023-2024. This work has not been submitted by anybody towards the award of any degree.

Date:

Place: Madanapalle **[Signature of the Students]**

ACKNOWLEDGEMENT

First, I must thank the almighty who has granted me knowledge, wisdom, strength, and courage to serve in this world and carry out my project work in a successful way.

We express our sincere thanks to **Dr. N. Vijaya Bhaskar Choudhry, Ph.D**. **Secretary & Correspondent**, Madanapalle Institute of Technology & Science for his continuous encouragement towards practical education and constant support in all aspects, including the provision of very good infrastructure facilities in the institute.

It is my duty to thank our **Principal**, **Dr. C. Yuvaraj**, Madanapalle Institute of Technology & Science, for his guidance and support at the time of my course and project.

I also wish to express my thanks to our **Vice Principal**, **Dr. P. Ramanathan** for his continuous support throughout my MCA career.

It is my foremost duty to thank the **Head of the Department, Dr. N. Naveen Kumar, MCA, Ph.D.,** Associate Professor, who gave me constant support during the project time and continuous encouragement towards the completion of the project successfully.

We thank our faculty guide **Dr. N. Naveen Kumar**, **Associate Professor**, for his continuous support in conducting periodical reviews and guidance until the completion of the project in a successful manner.

CHAPTER-1
INTRODUCTION

1.1 Problem Statement

The problem at hand revolves around the need for an effective mechanism to classify toxic comments in online discussions using Long Short-Term Memory (LSTM) networks. In the age of digital communication, online platforms have become a breeding ground for harmful and offensive language, posing a significant threat to healthy discourse and community well-being. The objective is to develop a robust LSTM-based model capable of accurately identifying and categorizing toxic comments, encompassing various forms of negativity, hate speech, and offensive content. The significance of this problem lies in the growing concern over the negative impact of toxic comments on online communities, user engagement, and overall online safety. Traditional methods of content moderation often fall short, given the dynamic nature of language and the evolving forms of online toxicity. LSTMs, as a type of recurrent neural network (RNN), excel in capturing sequential dependencies in data, making them particularly well-suited for the nuanced and context-dependent nature of language.

Addressing this issue requires the design and training of an LSTM model that can effectively learn and understand the contextual intricacies of language, distinguishing between acceptable and toxic content. Additionally, the model must be scalable to handle the vast amount of textual data generated on online platforms, ensuring real-time or near-real-time classification for timely content moderation. Furthermore, considerations for ethical AI implementation must be taken into account to avoid biases and ensure fairness in the classification process. Striking a balance between accurately identifying toxic comments and avoiding false positives is crucial to prevent unintentional censorship and maintain a platform's commitment to freedom of expression. In summary, the challenge is to develop an LSTM-based toxicity classification model that addresses the dynamic nature of online language, minimizes false positives, and upholds ethical standards in content moderation, ultimately fostering healthier and more inclusive online communities.

1.2 Process Description

Toxic comments, characterized by their offensive, disrespectful, or harmful nature, pose a substantial challenge in maintaining a positive online environment. The outlined objectives of the module aim to provide a comprehensive understanding of text classification challenges in natural language, emphasizing the pivotal role of automated content moderation in fostering healthy online platforms. The curriculum unfolds with an exploration of text classification intricacies, offering insights into the difficulties associated with identifying toxic comments and highlighting the broader importance of automated content moderation. Subsequently, the module introduces the fundamentals of LSTM, a variant of recurrent neural networks (RNNs), emphasizing its suitability for processing sequential data. Special attention is given to how LSTM effectively addresses the vanishing gradient problem, showcasing its prowess in handling long-range dependencies inherent in textual information. Moving forward, the module delves into the crucial aspect of preprocessing text data, guiding participants through techniques such as cleaning, tokenization, and strategies to handle imbalances in datasets. An emphasis is placed on the significance of word embeddings, elucidating their role in representing words within a continuous vector space.

Participants will learn to implement an LSTM-based neural network for toxic comment classification, leveraging popular deep learning frameworks like TensorFlow. The module provides guidance on fine-tuning hyperparameters to optimize the model's performance. Further, the training and evaluation segment imparts knowledge on data splitting into training, validation, and test sets. Techniques for effectively training the model and evaluating its performance are explored, with a focus on metrics such as accuracy, precision, recall, and the F1 score. To ensure the robustness of the model, the module concludes with a comprehensive exploration of strategies for handling imbalanced data. It will gain an understanding of techniques to address dataset imbalances, ensuring the LSTM model's capability to accurately identify both toxic and non-toxic comments.

CHAPTER-2
SYSTEM ANALYSIS

2.1. Existing System

Various systems exist for the classification of toxic comments, each presenting distinct strengths and weaknesses in addressing the complex challenge of online content moderation. Machine Learning Classifiers employ binary classification models, such as logistic regression, support vector machines (SVM), and random forests, to discern between toxic and non-toxic comments. While these models are interpretable and relatively straightforward, they may struggle with nuanced forms of toxicity. On the other hand, multi-class classification models, including neural networks (NN) and convolutional neural networks (CNN), provide a finer-grained analysis by distinguishing between multiple categories of toxicity, such as hate speech and threats. However, they tend to be more complex and less interpretable.

Deep learning models, exemplified by BERT and other pre-trained transformers, have demonstrated effectiveness by leveraging extensive pre-training on large text datasets. These models excel in capturing intricate semantic relationships, resulting in high accuracy in toxic comment classification. Rule-based approaches include systems using keyword lists, which identify known offensive words, and pattern matching systems that employ regular expressions and linguistic features to recognize toxic language patterns. While keyword lists are simple and fast, they may miss subtle forms of toxicity, and pattern matching systems, though more nuanced, require careful rule crafting.

Several examples of existing systems illustrate the diverse approaches adopted by major platforms. Facebook utilizes Natural Language Processing (NLP) and machine learning to flag and remove hate speech and harmful content. Twitter employs algorithms to identify and respond to abusive behavior, and YouTube leverages a combination of automated systems and human reviewers to combat harassment and hateful comments. These platforms showcase the adoption of a range of methodologies, from traditional machine learning classifiers to advanced deep learning models, and rule-based systems, highlighting the multifaceted nature of toxic comment classification in online environments.

2.2. Disadvantages

While various systems exist for toxic comment classification, each with its strengths, several drawbacks are associated with the proposed approaches. Machine Learning Classifiers, including logistic regression, support vector machines (SVM), and random forests, while interpretable, may struggle with nuanced forms of toxicity. The binary classification models lack the inherent capability to capture the intricate contextual relationships within text, limiting their effectiveness in handling complex language nuances. Multi-class classification models, such as neural networks (NN) and convolutional neural networks (CNN), provide a finer-grained analysis of toxicity categories but tend to be more complex and less interpretable. The increased complexity poses challenges in understanding the decision-making process of these models, hindering transparency and interpretability.

Deep learning models, exemplified by BERT and pre-trained transformers, demonstrate high accuracy but may suffer from a lack of power in handling long-range dependencies. While effective in capturing semantic relationships, these models may struggle with the contextual nuances present in lengthy and complex sentences, limiting their performance in certain scenarios. Rule-based approaches, utilizing keyword lists and pattern matching systems, have their limitations. Keyword lists, though simple and fast, may miss subtle forms of toxicity not captured by predefined offensive terms. Pattern matching systems, while more nuanced, demand meticulous rule crafting and may struggle with adapting to evolving linguistic patterns and variations.

In summary, the proposed systems, while offering advancements in toxic comment classification, exhibit limitations in handling nuanced language, complexity, interpretability, and adaptability to evolving linguistic patterns, underscoring the ongoing challenges in creating comprehensive and robust models for online content moderation.

2.3. Proposed System

The proposed system for toxic comment classification harnesses the power of Long Short-Term Memory (LSTM) networks, a specialized type of recurrent neural network (RNN), to address the nuanced challenges of online content moderation. LSTM networks excel in capturing sequential dependencies and contextual nuances present in natural language, making them well-suited for the dynamic and evolving nature of toxic comments. The system's architecture involves the implementation of an LSTM-based neural network designed to effectively distinguish between toxic and non-toxic comments. The model leverages the inherent ability of LSTMs to mitigate the vanishing gradient problem, allowing it to grasp long-range dependencies and capture the subtle intricacies of toxic language patterns.

The system also includes a robust preprocessing phase, incorporating techniques such as tokenization and addressing imbalances in the dataset to ensure the model's accuracy and generalization. Additionally, the utilization of word embeddings plays a crucial role in representing words in a continuous vector space, enhancing the model's understanding of semantic relationships within the comments. To optimize performance, the proposed system involves fine-tuning hyperparameters, allowing for effective training and accurate classification. The LSTM-based toxic comment classification system offers a promising approach by combining the strengths of deep learning with the specific advantages of LSTM networks. This system aims to contribute to the creation of a more effective and nuanced content moderation mechanism, fostering a positive and secure online environment by accurately identifying and categorizing toxic comments in a timely manner.

2.4. Advantages

Toxic comment classification using Long Short-Term Memory (LSTM) networks offers several advantages, making it a powerful and effective approach for addressing the challenges of online content moderation:

- ➢ **Sequential Learning**: LSTMs excel at capturing sequential dependencies and long-term relationships within data. In the context of toxic comment classification, where the meaning of a comment often depends on the context of previous words, LSTM's ability to understand and remember such sequential information is highly beneficial.
- ➢ **Contextual Understanding**: Toxicity in comments often involves nuanced language and context-dependent expressions. LSTMs, with their memory cells, can retain information over extended sequences, enabling a more nuanced understanding of the context surrounding potentially harmful language. This allows the model to better distinguish between innocuous and genuinely toxic statements.

- **Handling Variable Lengths**: Comments on online platforms vary in length, and LSTMs are well-suited for handling sequences of variable lengths. This flexibility is crucial when dealing with the diverse and unpredictable nature of user-generated content.
- **Effective for Time-Series Data**: Since online discussions unfold over time, treating comments as a time series is more representative of the reality. LSTMs are designed to work with time-series data, making them an apt choice for modeling the temporal dynamics inherent in online conversations.
- **Mitigation of Vanishing Gradient Problem**: LSTMs address the vanishing gradient problem, a common issue in training deep neural networks. This enables more effective learning and adaptation to the complexities of natural language, especially in cases where dependencies span over long sequences of words.
- **Learning Semantic Representations**: LSTMs, by learning distributed representations of words (word embeddings), capture the semantic relationships between words. This contributes to more meaningful understanding of the underlying meaning in comments, enhancing the model's ability to discern toxic language.
- **Adaptability to Varied Data Sources**: Toxic comments can manifest in different forms across various online platforms. The LSTM-based model is adaptable to diverse data sources, making it suitable for deployment in different online environments without significant re-engineering.
- **Reduced Need for Hand-Crafted Features**: LSTMs can automatically learn relevant features from the input data, reducing the need for extensive manual feature engineering. This is particularly advantageous in situations where the characteristics of toxic comments may evolve or differ across platforms.

2.5. Software Requirements

The software requirements for toxic comment classification using LSTM can vary based on the specific implementation; however, the following are general requirements:

- **Programming Language**: Python is commonly chosen for its simplicity and the extensive community support for machine learning tasks. Its rich ecosystem of libraries, including TensorFlow and PyTorch, facilitates the implementation of LSTM models for natural language processing.

- **Machine Learning Libraries**: Essential libraries such as pandas, numpy, matplotlib, seaborn, and scikit-learn, are often employed for data preprocessing, visualization, and implementing machine learning models. TensorFlow or PyTorch is crucial for developing and training LSTM networks.
- **Feature Extraction Tools**: Tools for extracting features from text data are essential for effective LSTM-based classification. Preprocessing techniques, tokenization, and embedding methods play a crucial role in extracting meaningful features from toxic comments.
- **Dataset**: A labeled dataset of toxic and non-toxic comments is necessary for training and evaluating the LSTM model. The quality and diversity of the dataset significantly impact the model's ability to generalize.
- **Hardware**: Adequate computational resources, including a powerful CPU and GPU, are required for training and running LSTM models efficiently. The availability of sufficient RAM is essential, especially when working with large datasets and complex models.
- **Operating System**: While the choice of the operating system is flexible, Linux is often preferred for its command-line tools and flexibility. However, the proposed model can be implemented on any operating system that supports the required libraries and tools for LSTM-based toxic comment classification.

2.6. Hardware Requirements

The hardware requirements for toxic comment classification using LSTM can vary based on the specific implementation and dataset size. However, here are some general requirements:

- **Processor (CPU)**: A high-performance multi-core processor is essential for expediting the training and prediction processes of the LSTM model. The CPU's processing speed significantly impacts the efficiency of handling sequential data in natural language processing tasks.
- **Memory (RAM)**: Sufficient RAM is crucial to accommodate the dataset and intermediate data during the training phase of the LSTM model. The memory capacity directly influences the model's ability to process and learn from the complex relationships within textual comments.
- **Storage**: Adequate storage space is necessary for storing the dataset, the LSTM model, and associated files. The LSTM model, with its sequential nature, may require substantial storage for both training data and the trained model parameters.
- **Graphics Processing Unit (GPU):** While not strictly mandatory, a powerful GPU can dramatically accelerate the training of deep learning models, including LSTM. GPUs excel in parallel processing, making them valuable for handling the matrix calculations involved in training complex neural networks.

- **Network:** A stable and high-speed network connection is essential for tasks such as downloading datasets, libraries, and pre-trained embeddings. Additionally, if the LSTM model is hosted on a server or deployed in a cloud environment, a reliable network connection becomes crucial for seamless access and interaction.
- **Additional Processing Units (TPU, if available)**: If available, Tensor Processing Units (TPUs) can further enhance the speed of model training. TPUs are specialized hardware accelerators designed for machine learning workloads, potentially offering improved performance over traditional CPUs and GPUs.

2.7. Feasibility Study

A feasibility study is a comprehensive analysis of the practicality and viability of a proposed project, system, or idea. It evaluates whether the project is technically, economically, operationally, and legally feasible, with the goal of determining its potential success or identifying significant issues that might make it impractical.

2.7.1. Technical Feasibility

The technical feasibility of implementing toxic comment classification using Long Short-Term Memory (LSTM) networks is well-established, relying on advancements in deep learning and natural language processing.

Key Points:

Model Performance: LSTMs have demonstrated high accuracy in sequence-based tasks, making them suitable for capturing nuances in toxic comments. Research indicates successful implementations achieving superior performance in comment classification.
Versatility of Models: LSTMs can incorporate various architectures, allowing them to adapt to the complexities of toxic language patterns and sequential dependencies present in comments.
Handling Diverse Data: LSTMs, designed for sequential data, can effectively handle the diverse and dynamic nature of textual content, ensuring robust performance in toxic comment classification.
Software and Hardware Requirements: Commonly used deep learning frameworks (e.g., TensorFlow, PyTorch), along with programming languages like Python, form the basis for

implementing LSTMs. Computational resources such as CPUs, GPUs, and sufficient RAM are manageable.

Real-World Application: LSTMs have been successfully applied in real-world scenarios, showcasing their practical utility in handling the challenges of toxic comment classification.

2.7.2. Economic Feasibility

The economic feasibility of implementing toxic comment classification using LSTM involves assessing the costs associated with development, maintenance, and potential returns.

Key Factors:

Cost of Implementation: Expenses include the development and maintenance of the LSTM model, covering hardware, software, and the expertise of data scientists.

Computational Cost: Training LSTMs can be computationally expensive, impacting hardware and electricity costs. However, their ability to provide accurate results may offset these expenses.

Cost of Errors: Considering the potential harm from false positives and false negatives, the high accuracy of LSTMs contributes to minimizing the costs associated with misclassifications.

Ongoing Costs: Regular updates and adaptations to evolving language patterns may incur ongoing costs but are essential for sustained model performance.

Cost of Alternatives: The economic feasibility should be compared with alternative solutions, weighing the benefits of LSTM-based classification against potential alternatives.

2.7.3. Legal Feasibility

The legal feasibility of toxic comment classification using LSTM depends on adherence to data protection, intellectual property, ethical guidelines, security laws, and export controls.

Considerations:

Data Privacy: Compliance with data protection laws (e.g., GDPR, CCPA) is essential when processing user-generated data in toxic comment classification.

Intellectual Property: Respect for copyright and license rules when using third-party software or datasets is critical to ensure legal and fair use.

Ethical Guidelines: Adherence to ethical rules for AI and machine learning, ensuring fairness, transparency, and responsibility, is imperative for legal compliance.

Security Laws: Compliance with cybersecurity laws ensures responsible and ethical use of tools without violating privacy or causing harm.

Export Controls: Awareness of and adherence to regulations on exporting machine learning technologies may impact deployment across borders.

2.7.4. Operational Feasibility

The operational feasibility of toxic comment classification using LSTM assesses its effective integration into existing systems and its practicality in real-world applications.

Considerations:

Effective Performance: LSTMs have demonstrated effectiveness in improving classification accuracy, enhancing their operational feasibility.

Handling Diverse Data: LSTMs can handle various data types and structures, making them suitable for the complex task of toxic comment classification.

Availability of Tools: Numerous machine learning libraries and tools support the implementation of LSTMs, enhancing operational feasibility.

Scalability: LSTMs can be scaled to handle large datasets, ensuring their suitability for real-world applications with varying data volumes.

Integration with Existing Systems: LSTMs can be seamlessly integrated with existing content moderation systems or online platforms, reinforcing their operational feasibility.

2.7.5. Social Feasibility

The social feasibility of toxic comment classification using LSTM assesses its acceptance, impact on user privacy, user trust, job creation, and its alignment with societal needs.

Considerations:

Public Acceptance: With the growing acceptance of AI and machine learning, society is increasingly embracing technologies that enhance online safety, such as toxic comment classification using LSTMs.

Need for Security: The rising number of cyber threats underscores the societal need for robust systems capable of identifying and classifying toxic content.

User Privacy: Ensuring user privacy is a critical social consideration, and the responsible use of LSTMs must respect individual rights.

Trust in AI: Increasing trust in the capabilities of AI and machine learning contributes to the social feasibility of employing LSTMs for complex tasks like comment classification.

Job Creation: The development, implementation, and maintenance of LSTMs for toxic comment classification can contribute to job creation in AI and cybersecurity, addressing societal employment needs.

CHAPTER 3

3. SYSTEM DESIGN

3.1 Block Diagram

The block diagram for toxic comment classification using LSTM presents a structured overview of the model architecture. The diagram typically includes key components such as Data Input, LSTM Layers, Embedding Layer, and Output Layer.

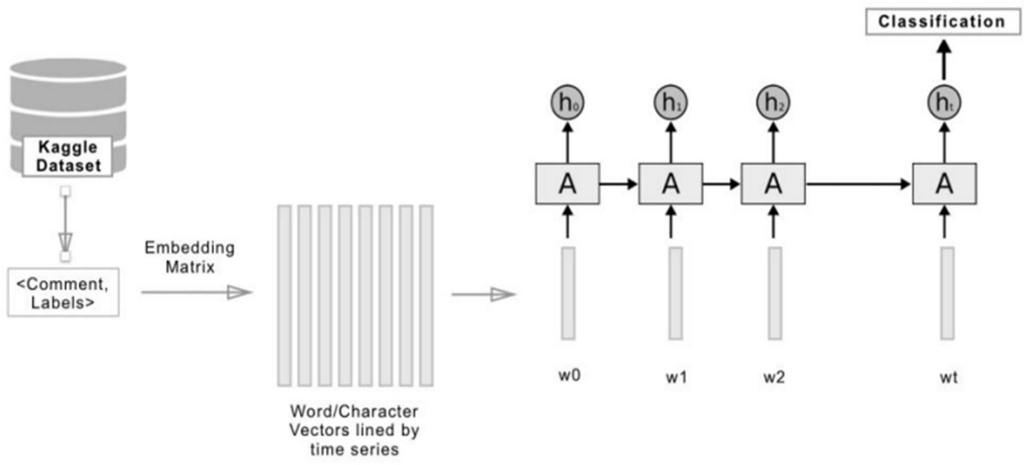


Fig.1. Block Diagram for Proposed System

3.2 Module Description

1. Data Collection:

Data is a fundamental component in developing an effective model. Our toxic comment classification model relies on a dataset obtained from reputable sources, such as online platforms and social media, containing textual comments labeled as toxic or non-toxic. This dataset provides valuable information for training the model, with features representing various aspects of language expression.

2. Data Preprocessing:

Raw text data often contains noise, inconsistencies, and missing values. To address these challenges, preprocessing techniques are applied, including text cleaning, tokenization, and handling imbalances in the dataset. Techniques like word embedding are employed to represent words in a continuous vector space, enhancing the model's understanding of semantic relationships within comments.

3. Feature Selection:

Textual data presents a challenge in feature selection. Autoencoder-decoder architectures are employed (Fig. 2) to generate meaningful features from the text, effectively reducing dimensionality and selecting relevant features for model training. This process enhances the model's ability to capture essential patterns in toxic language.

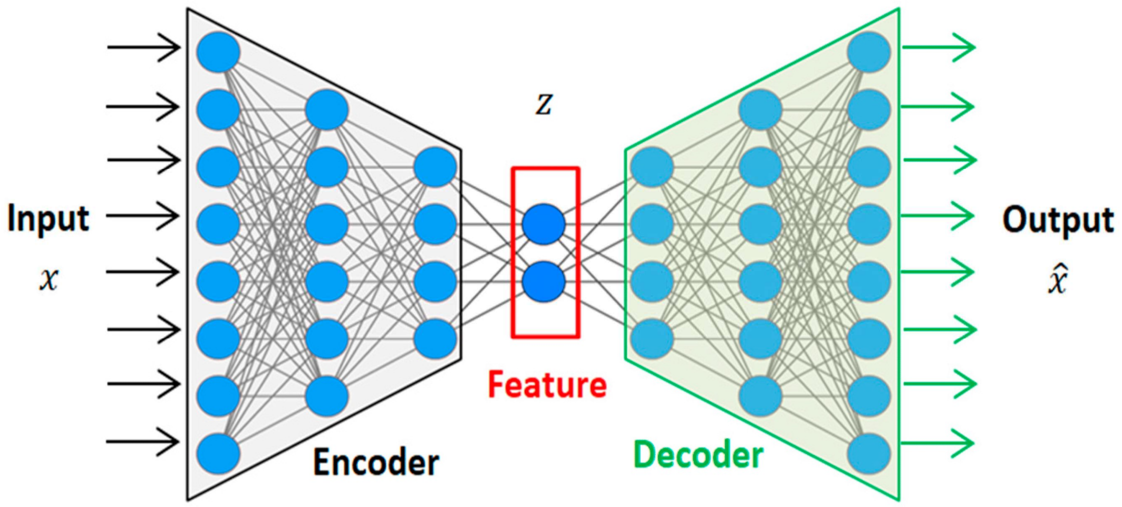


Fig.2. Architecture of autoencoder-decoder for feature generation

4. Model Building:

The Model building for toxic comment classification using Long Short-Term Memory (LSTM) networks involves creating a powerful and effective deep learning architecture. In this project, LSTM is employed due to its capability in handling sequential data, making it ideal for understanding the contextual nuances in textual comments. The architecture comprises an embedding layer, LSTM layers, and a dense output layer. The embedding layer converts words into continuous vector representations, capturing semantic relationships. The LSTM layers process sequential information, addressing the challenge of long-range dependencies in language. The dense output layer produces the final classification decision. Fig. 3 illustrates the basic structure of the LSTM-based model, emphasizing the flow of information through the layers. Each LSTM cell preserves memory over time, allowing the model to grasp the context and intricacies of toxic language. The absence of a meta-classifier simplifies the architecture, focusing on the inherent capabilities of LSTM for accurate toxic comment classification.

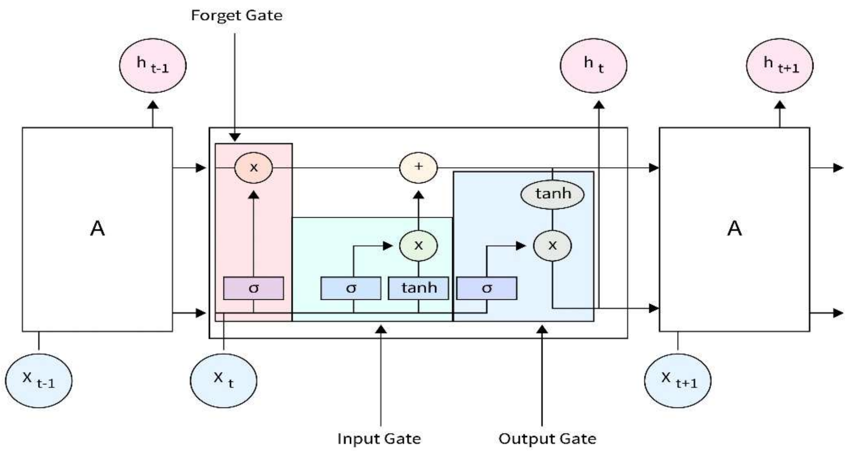


Fig.3. General structure of LSTM classifier

5. Predictions: LSTM, or Long Short-Term Memory, excels in predicting sequences by capturing long-range dependencies in data. Utilized in toxic comment classification, LSTM effectively analyzes and understands contextual nuances in textual comments, providing accurate predictions without the need for additional meta-classifiers.

6. **Evaluations**:

Evaluation metrics, such as accuracy, precision, recall, and F1 score, are essential for assessing the performance of the toxic comment classification model. The confusion matrix, adapted for textual data, provides insights into the model's ability to correctly classify toxic and non-toxic comments, aiding in the refinement and optimization of the LSTM-based classification system.

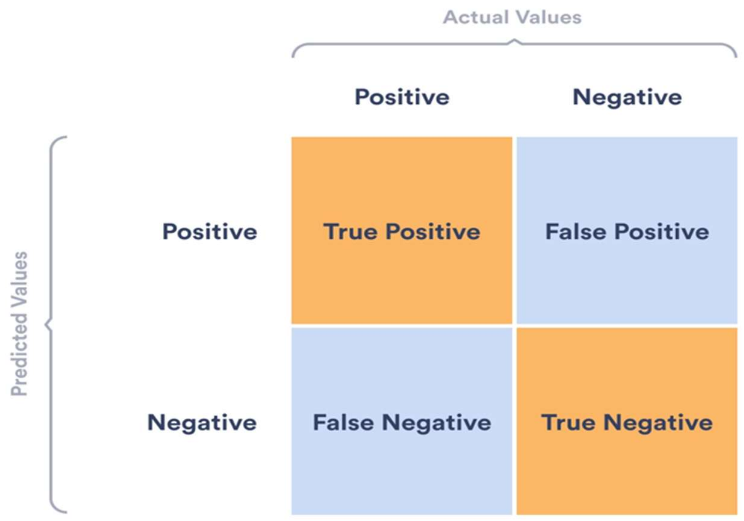


Fig.4. General confusion matrix used for classification algorithms.

3.3 UML Diagrams

Unified Modeling Language (UML) diagrams are widely adopted visual representation tools in software engineering and systems development. Serving as a standardized communication method, UML enables software professionals to express intricate concepts and design solutions with clarity. These diagrams, encompassing types like class diagrams for structure, use case diagrams for functional requirements, and sequence diagrams for dynamic interactions, provide a graphical means to illustrate the architecture, behavior, and relationships within a system. Activity diagrams model workflows, state diagrams represent object lifecycles, and component diagrams depict software component organization. UML diagrams act as a collaborative bridge, fostering communication among developers, architects, and clients throughout the software development lifecycle. The versatility and comprehensiveness of UML position it as an invaluable tool for designing, documenting, and comprehending complex software systems.

3.3.1 Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

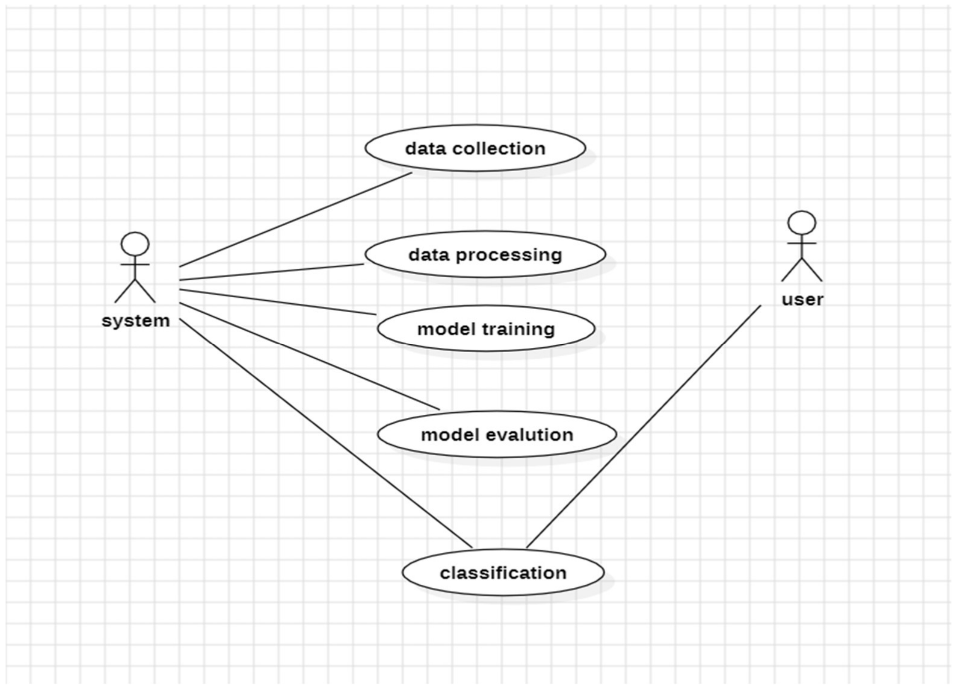


Fig.5.Use Case diagram for Proposed System

3.3.2 Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

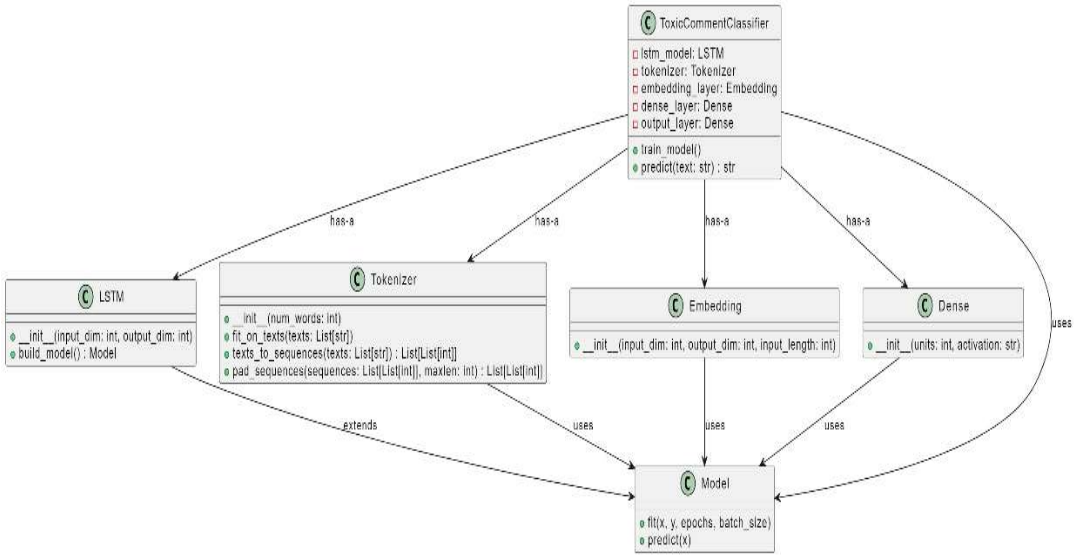

ToxicCommentClassifier
lstm_model: LSTM
tokenizer: Tokenizer
embedding_layer: Embedding
dense_layer: Dense
output_layer: Dense
train_model()
predict(text: str) : str
has-a
has-a
has-a
has-a
uses
LSTM
__init__(input_dim: int, output_dim: int)
build_model() : Model
Tokenizer
__init__(num_words: int)
fit_on_texts(texts: List[str])
texts_to_sequences(texts: List[str]) : List[List[int]]
pad_sequences(sequences: List[List[int]], maxlen: int) : List[List[int]]
Embedding
__init__(input_dim: int, output_dim: int, input_length: int)
Dense
__init__(units: int, activation: str)
extends
uses
uses
uses
Model
fit(x, y, epochs, batch_size)
predict(x)

Fig.6.Class diagram for Proposed System

3.3.3 Sequence Diagram

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

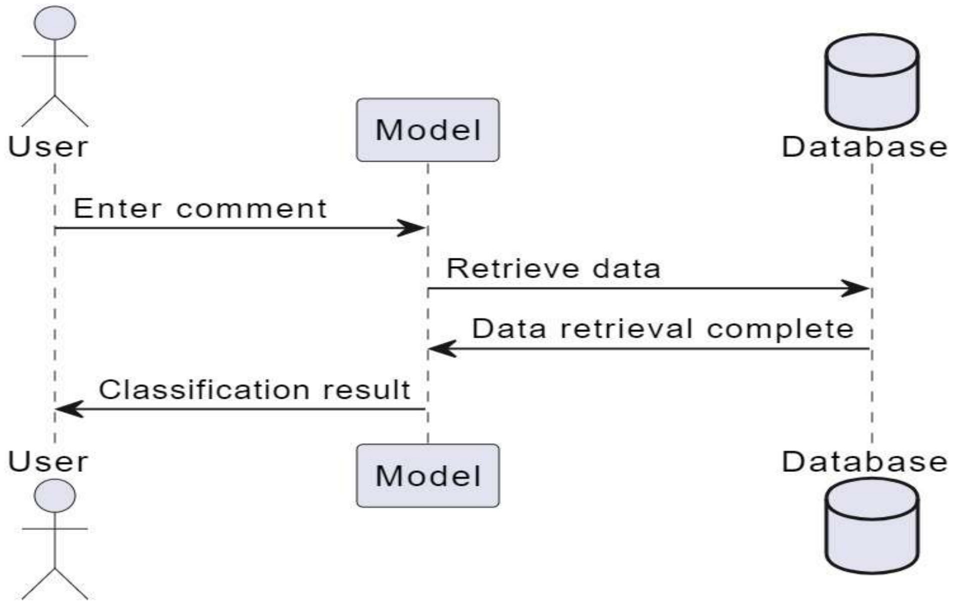


Fig.7.Sequence diagram for Proposed System

3.3.4 Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

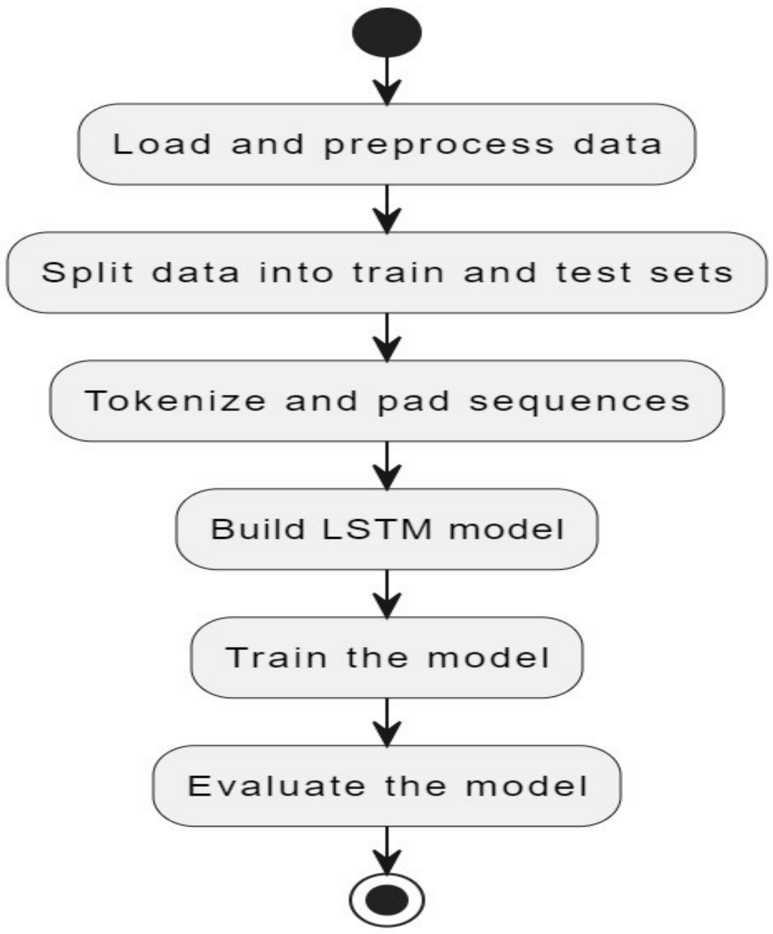


Fig.8.Sequence diagram for Proposed System

CHAPTER 4

4.SYSTEM IMPLEMENTATION

4.1. Language Selection

Choosing a programming language for implementing the proposed model involves considering factors such as library availability, community support, performance needs, and the development team's familiarity. Python, JavaScript, Java, and Julia are commonly used languages for machine learning and deep learning. In this project, Python was selected due to its extensive community and a rich set of libraries essential for building machine learning and deep learning models.

The Python libraries employed in this model include:

1. **Pandas**: Used for loading datasets in various formats (json, .csv, .tsv) and handling preprocessing tasks such as managing missing values.

2. **Scikit-Learn**: Serves as a toolbox for creating intelligent computer programs, facilitating machine learning tasks like pattern recognition, prediction, and problem-solving.

3. **Matplotlib**: A tool for generating visual plots and charts in Python, enabling users to present data in clear and understandable formats like line graphs and bar charts.

4. **Seaborn**: Facilitates the creation of visually appealing and easy-to-understand graphs in Python. Seamlessly works with data from pandas, making it simple to visualize trends, patterns, and relationships.

5. **Tensorflow**: Utilized for constructing autoencoder-decoders, acting as intelligent filters for data. These structures learn to compress and reconstruct information, functioning as feature generators. Training on data allows them to capture essential features, aiding in effective feature selection and enhancing data quality for machine understanding and processing.

4.2 Algorithm

Long Short-Term Memory (LSTM) networks, a specialized type of recurrent neural network (RNN), play a pivotal role in toxic comment classification by addressing the challenges associated with understanding the contextual nuances in textual data. LSTMs excel in modeling sequential dependencies, making them adept at capturing the intricate structures and patterns prevalent in language. In the context of toxic comment classification, LSTMs operate by tokenizing and embedding words into continuous vector representations, enabling the model to discern semantic relationships. The LSTM layers, equipped with memory cells, efficiently process sequential information, overcoming the vanishing gradient problem associated with traditional RNNs.

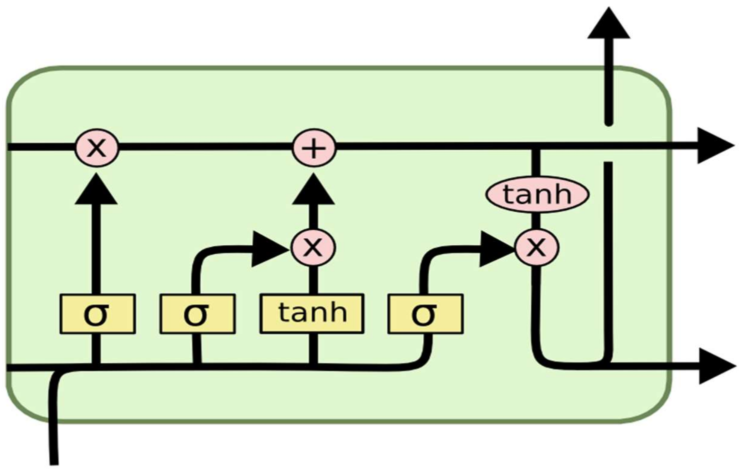


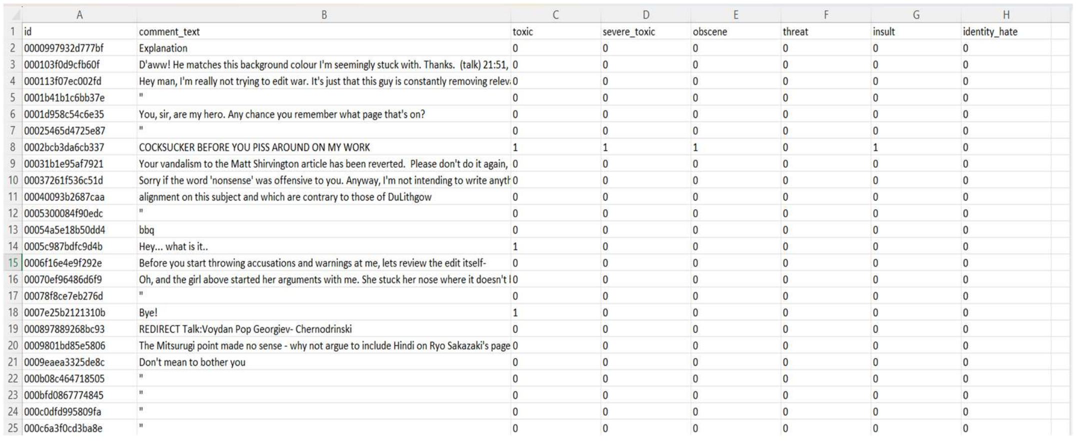
Fig.9.Workflow of LSTM

This architectural strength allows LSTMs to effectively capture long-range dependencies, crucial for discerning toxic language patterns. The trained LSTM model, equipped with its memory capabilities, becomes proficient in distinguishing between toxic and non-toxic comments, offering a robust solution for content moderation in online platforms.

4.3 Screen Shots

To enhance the performance of the model, it is imperative to furnish a high-quality dataset. An exemplary dataset for toxic comment classification is sourced from the Canadian Institute for

Cybersecurity. The dataset comprises instances of comments, both toxic and non-toxic, as illustrated in Fig.10. This dataset serves as a foundational resource for training and evaluating the model's ability to accurately classify and distinguish toxic comments, contributing to the effectiveness of content moderation in online platforms.



	A	B	C	D	E	F	G	H
1	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
2	0000997932d777bf	Explanation	0	0	0	0	0	0
3	000103f0d9cfb60f	D'aww! He matches this background colour I'm seemingly stuck with. Thanks. (talk) 21:51,	0	0	0	0	0	0
4	000113f07ec002fd	Hey man, I'm really not trying to edit war. It's just that this guy is constantly removing relev	0	0	0	0	0	0
5	0001b41b1c6bb37e	"	0	0	0	0	0	0
6	0001d958c54c6e35	You, sir, are my hero. Any chance you remember what page that's on?	0	0	0	0	0	0
7	00025465d4725e87	"	0	0	0	0	0	0
8	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
9	00031b1e95af7921	Your vandalism to the Matt Shirvington article has been reverted. Please don't do it again,	0	0	0	0	0	0
10	00037261f536c51d	Sorry if the word 'nonsense' was offensive to you. Anyway, I'm not intending to write anyth	0	0	0	0	0	0
11	00040093b2687caa	alignment on this subject and which are contrary to those of DuLithgow	0	0	0	0	0	0
12	0005300084f90edc	"	0	0	0	0	0	0
13	00054a5e18b50dd4	bbq	0	0	0	0	0	0
14	0005c987bdfc9d4b	Hey... what is it..	1	0	0	0	0	0
15	0006f16e4e9f292e	Before you start throwing accusations and warnings at me, lets review the edit itself-	0	0	0	0	0	0
16	00070ef96486d6f9	Oh, and the girl above started her arguments with me. She stuck her nose where it doesn't l	0	0	0	0	0	0
17	00078f8ce7eb276d	"	0	0	0	0	0	0
18	0007e25b2121310b	Bye!	1	0	0	0	0	0
19	000897889268bc93	REDIRECT Talk:Voydan Pop Georgiev- Chernodrinski	0	0	0	0	0	0
20	0009801bd85e5806	The Mitsurugi point made no sense - why not argue to include Hindi on Ryo Sakazaki's page	0	0	0	0	0	0
21	0009eaea3325de8c	Don't mean to bother you	0	0	0	0	0	0
22	000b08c464718505	"	0	0	0	0	0	0
23	000bfd0867774845	"	0	0	0	0	0	0
24	000c0dfd995809fa	"	0	0	0	0	0	0
25	000c6a3f0cd3ba8e	"	0	0	0	0	0	0

Fig.10. Dataset used for model

Dataset classified into two types training and testing set. 159,751 samples are containing in training dataset. The average comment consisted of 394 characters. Figure 11 depicts the distribution of toxicity types with percentage values of toxic 43%,sever_toxic 4.5%,obscene 24.1%,threat 1.4%,insult 22.4%,indentity_hate 4.0%. This breakdown provides a nuanced understanding of the prevalence of different toxicity types, enabling a targeted analysis of the challenges posed by each category.

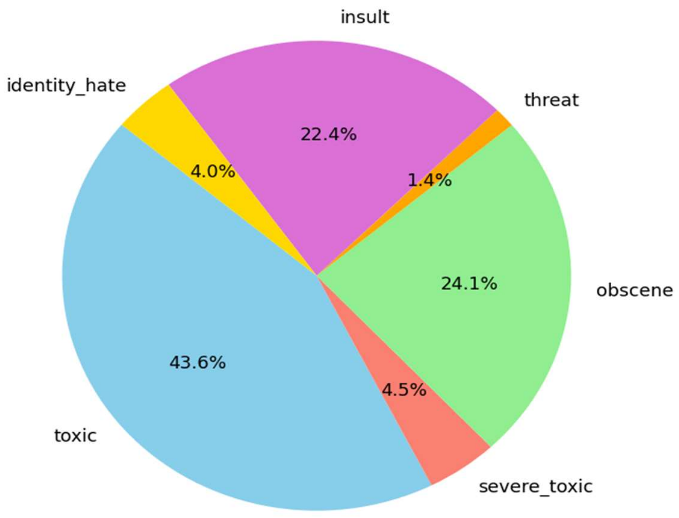


Fig.11. Distribution of Toxicity Types in the Dataset

Dataset exhibits a nuanced and complex correlation between various types of toxic comments. Through careful analysis ,it clears that certain types of toxicity co-occur more frequently than others. Figure 12 depicts the dataset's composition, comprising categories such as toxic, severe_toxic, obscene, threat, insult, and identity_hate, allows for a comprehensive exploration of these correlations.

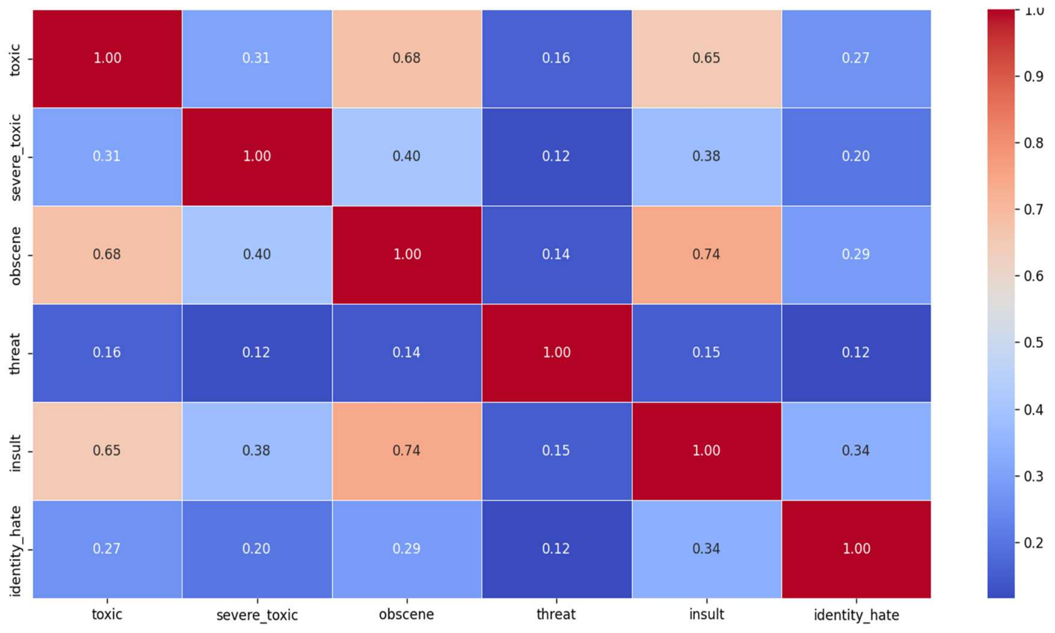
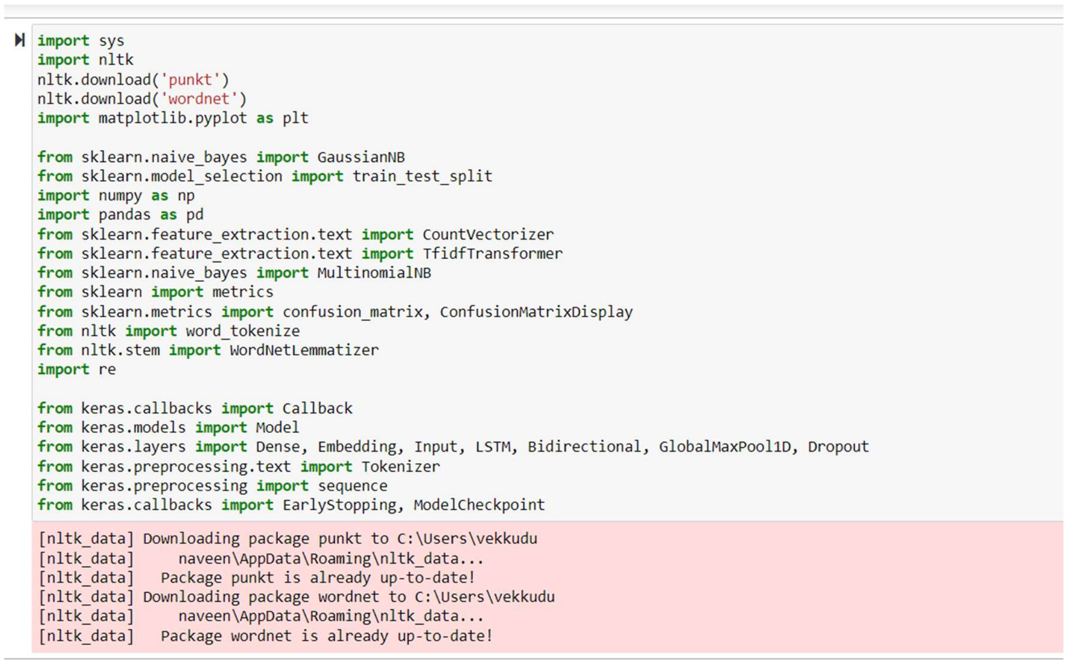


Fig.12. Correlation Matrix of Toxicity Types.

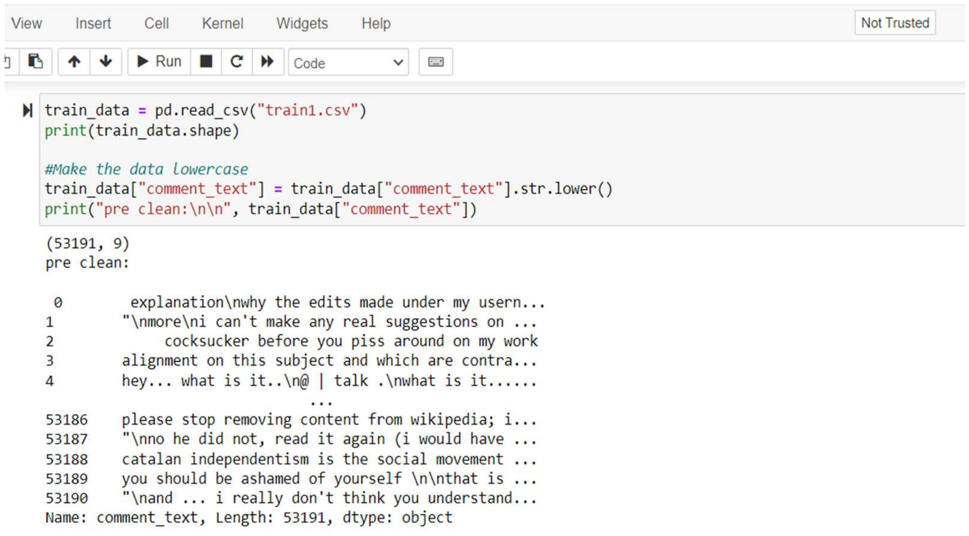


```
import sys
import nltk
nltk.download('punkt')
nltk.download('wordnet')
import matplotlib.pyplot as plt

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
import re

from keras.callbacks import Callback
from keras.models import Model
from keras.layers import Dense, Embedding, Input, LSTM, Bidirectional, GlobalMaxPool1D, Dropout
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
[nltk_data] Downloading package punkt to C:\Users\vekkudu
[nltk_data]     naveen\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\vekkudu
[nltk_data]     naveen\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Fig.13.Importing all necessity libraries.

Data before cleaning in toxic comment classification refers to the raw, unprocessed information containing noise, errors, and inconsistencies. Cleaning is vital to enhance data quality, ensuring accurate model training. It removes irrelevant details, corrects errors, and standardizes formats, fostering reliable insights and improving the overall performance of the toxic comment classification model.

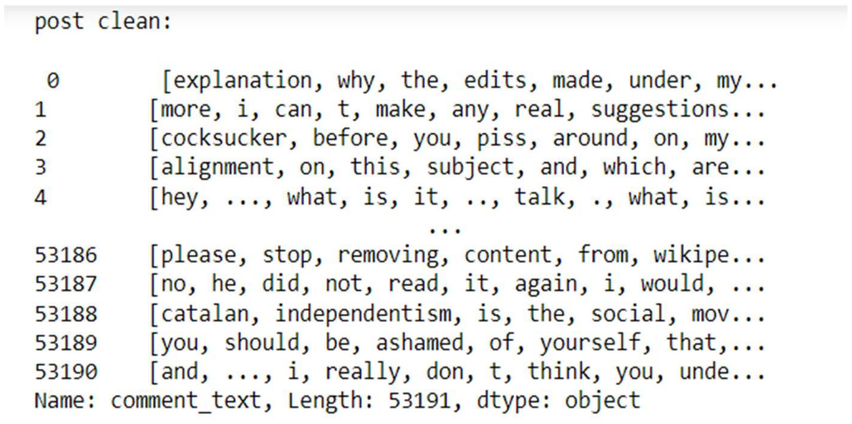


```
train_data = pd.read_csv("train1.csv")
print(train_data.shape)

#Make the data lowercase
train_data["comment_text"] = train_data["comment_text"].str.lower()
print("pre clean:\n\n", train_data["comment_text"])
```

```
(53191, 9)
pre clean:

 0        explanation\nwhy the edits made under my usern...
1       "\nmore\ni can't make any real suggestions on ...
2           cocksucker before you piss around on my work
3       alignment on this subject and which are contra...
4       hey... what is it..\n@ | talk .\nwhat is it......
                              ...
53186   please stop removing content from wikipedia; i...
53187   "\nno he did not, read it again (i would have ...
53188   catalan independentism is the social movement ...
53189   you should be ashamed of yourself \n\nthat is ...
53190   "\nand ... i really don't think you understand...
Name: comment_text, Length: 53191, dtype: object
```

After cleaning in toxic comment classification, data is refined by removing noise, inconsistencies, and irrelevant information. Cleaning is crucial as it enhances model accuracy, prevents biased training, and ensures reliable insights. Clean data is fundamental for robust machine learning models, facilitating accurate classification of toxic and non-toxic comments.

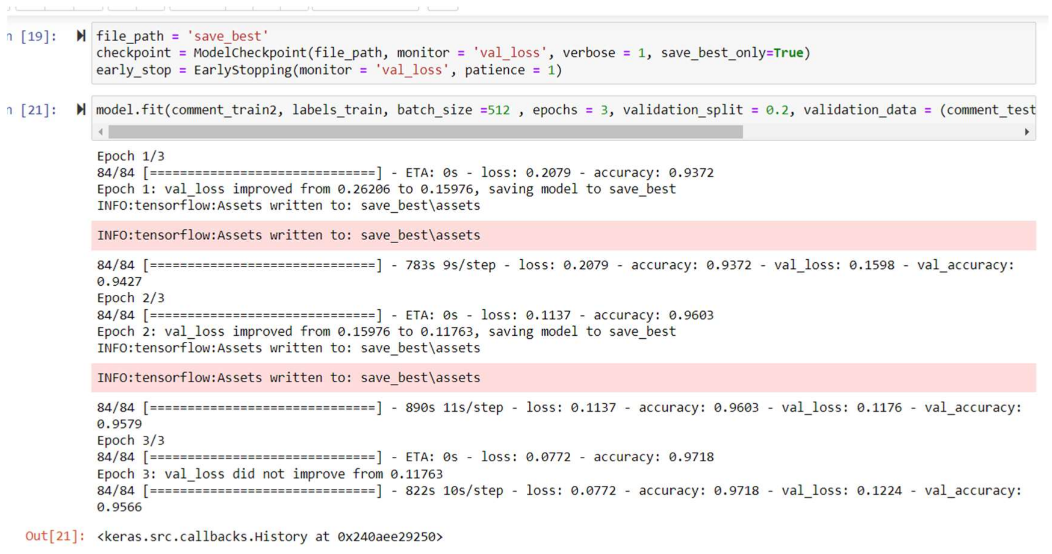
Fig.14.Data before Cleaning.



```
post clean:

 0        [explanation, why, the, edits, made, under, my...
1        [more, i, can, t, make, any, real, suggestions...
2        [cocksucker, before, you, piss, around, on, my...
3        [alignment, on, this, subject, and, which, are...
4        [hey, ..., what, is, it, .., talk, ., what, is...
                                ...
53186    [please, stop, removing, content, from, wikipe...
53187    [no, he, did, not, read, it, again, i, would, ...
53188    [catalan, independentism, is, the, social, mov...
53189    [you, should, be, ashamed, of, yourself, that,...
53190    [and, ..., i, really, don, t, think, you, unde...
Name: comment_text, Length: 53191, dtype: object
```

Fig.15.Data after cleaning.

Model training in toxic comment classification involves exposing the algorithm to labeled data, enabling it to learn patterns and distinctions between toxic and non-toxic comments. The significance of model training lies in refining the algorithm's ability to make accurate predictions, fostering an effective content moderation system for online platforms.



```
n [19]:  file_path = 'save_best'
         checkpoint = ModelCheckpoint(file_path, monitor = 'val_loss', verbose = 1, save_best_only=True)
         early_stop = EarlyStopping(monitor = 'val_loss', patience = 1)

n [21]:  model.fit(comment_train2, labels_train, batch_size =512 , epochs = 3, validation_split = 0.2, validation_data = (comment_test

         Epoch 1/3
         84/84 [==============================] - ETA: 0s - loss: 0.2079 - accuracy: 0.9372
         Epoch 1: val_loss improved from 0.26206 to 0.15976, saving model to save_best
         INFO:tensorflow:Assets written to: save_best\assets

         INFO:tensorflow:Assets written to: save_best\assets

         84/84 [==============================] - 783s 9s/step - loss: 0.2079 - accuracy: 0.9372 - val_loss: 0.1598 - val_accuracy:
         0.9427
         Epoch 2/3
         84/84 [==============================] - ETA: 0s - loss: 0.1137 - accuracy: 0.9603
         Epoch 2: val_loss improved from 0.15976 to 0.11763, saving model to save_best
         INFO:tensorflow:Assets written to: save_best\assets

         INFO:tensorflow:Assets written to: save_best\assets

         84/84 [==============================] - 890s 11s/step - loss: 0.1137 - accuracy: 0.9603 - val_loss: 0.1176 - val_accuracy:
         0.9579
         Epoch 3/3
         84/84 [==============================] - ETA: 0s - loss: 0.0772 - accuracy: 0.9718
         Epoch 3: val_loss did not improve from 0.11763
         84/84 [==============================] - 822s 10s/step - loss: 0.0772 - accuracy: 0.9718 - val_loss: 0.1224 - val_accuracy:
         0.9566

 Out[21]: <keras.src.callbacks.History at 0x240aee29250>
```

Fig.16.Model Training.

The confusion matrix in toxic comment classification serves as a crucial performance evaluation tool. It vividly presents the model's predictions, distinguishing true positives, true negatives, false positives, and false negatives. This breakdown aids in assessing precision, recall, accuracy, and F1 score, providing a comprehensive understanding of the model's effectiveness and areas for improvement.

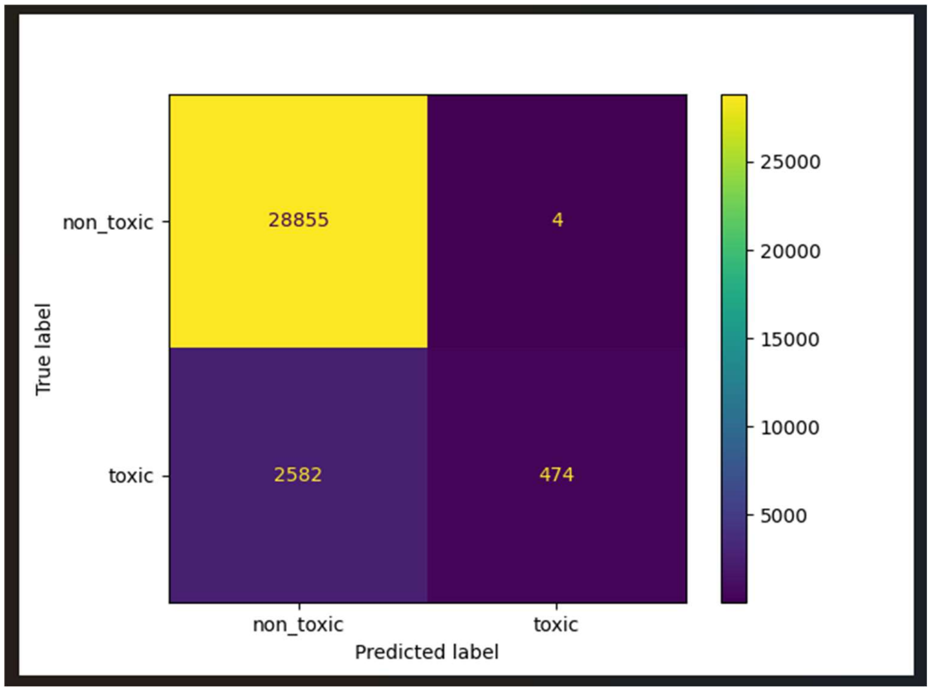
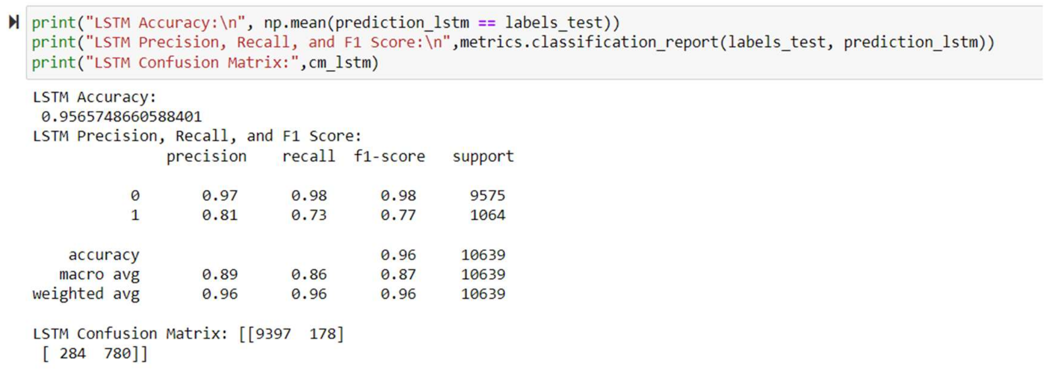


Fig.17.Confusion obtained by Proposed Model.

Evaluation metric scores, such as accuracy, precision, recall, and F1 score, are vital in toxic comment classification. They quantitatively assess model performance, providing insights into its ability to correctly identify toxic comments. These scores guide model refinement, ensuring optimal accuracy and reliability in content moderation, crucial for maintaining a positive online environment.



```
print("LSTM Accuracy:\n", np.mean(prediction_lstm == labels_test))
print("LSTM Precision, Recall, and F1 Score:\n",metrics.classification_report(labels_test, prediction_lstm))
print("LSTM Confusion Matrix:",cm_lstm)
```

```
LSTM Accuracy:
 0.9565748660588401
LSTM Precision, Recall, and F1 Score:
               precision    recall  f1-score   support

           0       0.97      0.98      0.98      9575
           1       0.81      0.73      0.77      1064

    accuracy                           0.96     10639
   macro avg       0.89      0.86      0.87     10639
weighted avg       0.96      0.96      0.96     10639

LSTM Confusion Matrix: [[9397  178]
 [ 284  780]]
```

Fig.18.Evalution Metrics Scores

4.4 Sample Code

```
import sys

import nltk

nltk.download('punkt')

nltk.download('wordnet')

import matplotlib.pyplot as plt


from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import train_test_split

import numpy as np

import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.feature_extraction.text import TfidfTransformer

from sklearn.naive_bayes import MultinomialNB

from sklearn import metrics

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
import re

from keras.callbacks import Callback
from keras.models import Model
from keras.layers import Dense, Embedding, Input, LSTM, Bidirectional, GlobalMaxPool1D,
Dropout
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping, ModelCheckpoint


class LemmaTokenizer(object):
    def __init__(self):
        self.wnl = WordNetLemmatizer()
    def __call__(self, articles):
        return [self.wnl.lemmatize(t) for t in word_tokenize(articles)]
train_data = pd.read_csv("train1.csv")
print(train_data.shape)

#Make the data lowercase
train_data["comment_text"] = train_data["comment_text"].str.lower()
print("pre clean:\n\n", train_data["comment_text"])
```

```
(53191, 9)
pre clean:
 0        explanation\nwhy the edits made under my usern...
1        "\nmore\ni can't make any real suggestions on ...
2             cocksucker before you piss around on my work
3        alignment on this subject and which are contra...
4        hey... what is it..\n@ | talk .\nwhat is it......
                         ...
53186    please stop removing content from wikipedia; i...
53187    "\nno he did not, read it again (i would have ...
53188    catalan independentism is the social movement ...
53189    you should be ashamed of yourself \n\nthat is ...
53190    "\nand ... i really don't think you understand...
Name: comment_text, Length: 53191, dtype: object
```

```
def cleaning(data):
    #remove the characters in the first parameter
    clean_column = re.sub('<.*?>', ' ', str(data))
    #removes non-alphanumeric characters(exclamation point, colon, etc) except periods.
    clean_column = re.sub('[^a-zA-Z0-9\.]+',' ', clean_column)
    #tokenize
    tokenized_column = word_tokenize(clean_column)
    return tokenized_column
#use panda apply to apply to each comment
train_data["comment_text"] = train_data["comment_text"].apply(cleaning)
print("post clean:\n\n", train_data["comment_text"])
```

```
post clean:

 0        [explanation, why, the, edits, made, under, my...
1        [more, i, can, t, make, any, real, suggestions...
2        [cocksucker, before, you, piss, around, on, my...
3        [alignment, on, this, subject, and, which, are...
```

```
4       [hey, ..., what, is, it, .., talk, ., what, is...
                    ...
53186   [please, stop, removing, content, from, wikipe...
53187   [no, he, did, not, read, it, again, i, would, ...
53188   [catalan, independentism, is, the, social, mov...
53189   [you, should, be, ashamed, of, yourself, that,...
53190   [and, ..., i, really, don, t, think, you, unde...
Name: comment_text, Length: 53191, dtype: object
```

```
lemmatizer = WordNetLemmatizer():
   #input our data in function, take the cleaned column
   my_data = data["comment_text"]
   lemmatized_list = [lemmatizer.lemmatize(word) for word in my_data]
   return (lemmatized_list)

train_data["lemmatized"] = train_data.apply(lemmatizing,axis=1)

train_data.shape
(53191, 10)

train_data["comment_text"] = train_data["lemmatized"]
train = train_data[["comment_text"]]
print(train)
```

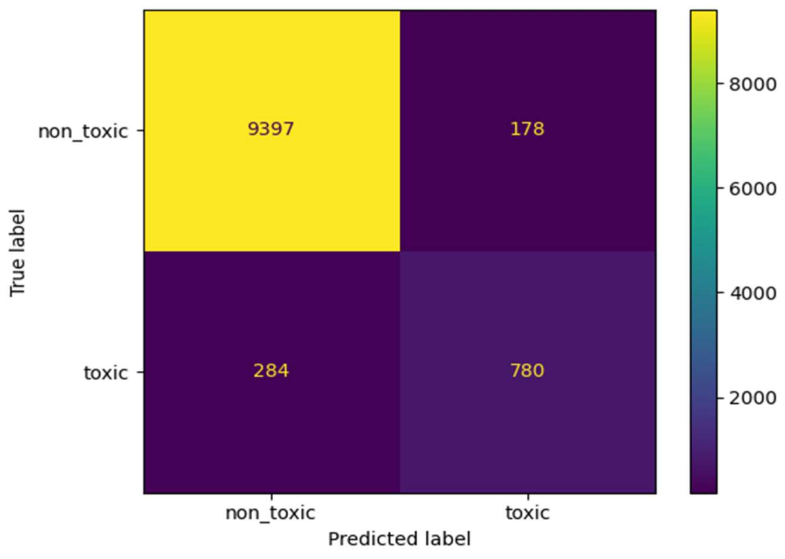
```
0     [explanation, why, the, edits, made, under, my...
1     [more, i, can, t, make, any, real, suggestion,...
2     [cocksucker, before, you, piss, around, on, my...
3     [alignment, on, this, subject, and, which, are...
4     [hey, ..., what, is, it, .., talk, ., what, is...
...                                   ...
53186  [please, stop, removing, content, from, wikipe...
53187  [no, he, did, not, read, it, again, i, would, ...
53188  [catalan, independentism, is, the, social, mov...
```

```
53189  [you, should, be, ashamed, of, yourself, that,...
53190  [and, ..., i, really, don, t, think, you, unde...

[53191 rows x 1 columns]

train_labels = train_data[["toxic"]]
print(train_labels)
       toxic
0      0
1      0
2      1
3      0
4      1
...    ...
53186  0
53187  0
53188  0
53189  0
53190  0

[53191 rows x 1 columns]

comment_train, comment_test, labels_train, labels_test = train_test_split(train, train_labels, test_size
= 0.2, random_state=42)
#Transpose and flatten so it fits the correct dimensions

print(labels_train.shape)
labels_train = np.transpose(labels_train)
labels_train = np.ravel(labels_train)
print(labels_train.shape)
labels_test = np.transpose(labels_test)
labels_test = np.ravel(labels_test)
#comment_train_converted = comment_train.comment_text.astype(str)
#comment_test_converted = comment_test.comment_text.astype(str)
```

```
(42552, 1)
(42552,)

count_vect = CountVectorizer()
comment_train_counts = count_vect.fit_transform(comment_train['comment_text'].astype(str))

tfidf_transformer = TfidfTransformer()
comment_train_tfidf = tfidf_transformer.fit_transform(comment_train_counts)


num_words = 20000
max_len = 150
emb_size = 128

# Converting each text into a sequence integer
tok = Tokenizer(num_words = num_words, lower=True, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\
t\n')
tok.fit_on_texts(list(comment_train.comment_text.astype(str)))

# Converting each comment into sequence of integer
comment_train2 = tok.texts_to_sequences(comment_train.comment_text.astype(str))
comment_test2 = tok.texts_to_sequences(comment_test.comment_text.astype(str))
comment_train2 = sequence.pad_sequences(comment_train2, maxlen = max_len)
comment_test2 = sequence.pad_sequences(comment_test2, maxlen = max_len)


inp = Input(shape = (max_len, ))
layer = Embedding(num_words, emb_size)(inp)
layer = Bidirectional(LSTM(50, return_sequences = True, recurrent_dropout = 0.15))(layer)
layer = GlobalMaxPool1D()(layer)
layer = Dropout(0.2)(layer)
layer = Dense(50, activation = 'relu')(layer)
layer = Dropout(0.2)(layer)
layer = Dense(1, activation = 'sigmoid')(layer)
model = Model(inputs = inp, outputs = layer)
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])
print("\n")
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150)]             0

 embedding (Embedding)       (None, 150, 128)          2560000

 bidirectional (Bidirection  (None, 150, 100)          71600
 al)

 global_max_pooling1d (Glob  (None, 100)               0
 alMaxPooling1D)

 dropout (Dropout)           (None, 100)               0

 dense (Dense)               (None, 50)                5050

 dropout_1 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 2636701 (10.06 MB)
Trainable params: 2636701 (10.06 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
file_path = 'save_best'
file_path = 'save_best'
checkpoint = ModelCheckpoint(file_path, monitor = 'val_loss', verbose = 1, save_best_only=True)
early_stop = EarlyStopping(monitor = 'val_loss', patience = 1)

cm_lstm = confusion_matrix(labels_test, prediction_lstm)
cmd_lstm = ConfusionMatrixDisplay(cm_lstm, display_labels=["non_toxic", "toxic"])
cmd_lstm.plot()
plt.show()

cm_lstm = confusion_matrix(labels_test, prediction_lstm)
cmd_lstm = ConfusionMatrixDisplay(cm_lstm, display_labels=["non_toxic", "toxic"])
cmd_lstm.plot()
plt.show()

cm_lstm = confusion_matrix(labels_test, prediction_lstm)
cmd_lstm = ConfusionMatrixDisplay(cm_lstm, display_labels=["non_toxic", "toxic"])
cmd_lstm.plot()
plt.show()
```



```
print("LSTM Accuracy:\n", np.mean(prediction_lstm == labels_test))
print("LSTM Precision, Recall, and F1 Score:\n",metrics.classification_report(labels_test, prediction
_lstm))
print("LSTM Confusion Matrix:",cm_lstm)
```

```
LSTM Accuracy:
 0.9565748660588401
LSTM Precision, Recall, and F1 Score:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98      9575
           1       0.81      0.73      0.77      1064

    accuracy                           0.96     10639
   macro avg       0.89      0.86      0.87     10639
weighted avg       0.96      0.96      0.96     10639

LSTM Confusion Matrix: [[9397  178]
 [ 284  780]]
```

CHAPTER 5
SYSTEM TESTING

5. System Testing

5.1 Testing Description

Testing constitutes a pivotal stage within the software development life cycle, crucial for validating that the software aligns with specified requirements and functions as intended. This process involves the systematic execution of a program or system to identify errors and confirm its expected behavior. The testing continuum spans various levels, encompassing unit testing for individual components, integration testing for combined units, and system testing for a comprehensive evaluation of the entire application. Non-functional testing supplements functional testing by scrutinizing performance, security, and usability aspects. Testers employ diverse techniques, including manual and automated testing, to unearth defects and ensure software congruence with user expectations.

Quality assurance practices are integral, incorporating rigorous testing to bolster the reliability and stability of software products. Test cases are strategically devised to cover diverse scenarios, with testing methodologies evolving in tandem with agile development approaches. Continuous integration and continuous testing assume central roles in expeditiously delivering high-quality software. In essence, proficient testing not only fortifies the application's robustness but also fosters user satisfaction and instills trust in the software's reliability.

5.1.1. Unit Testing

Unit testing for toxic comment classification using LSTM involves scrutinizing individual components to ensure their proper functionality. The following are potential unit tests for this context:

Data Preprocessing: Evaluate the effectiveness of data preprocessing by reviewing steps such as transforming information into features, normalizing values, and encoding categorical data. Verify the accuracy of processed data to ensure each preprocessing step has been executed correctly, guaranteeing its readiness for analysis.

LSTM Model Training: Assess the performance of the LSTM model by testing its ability to learn from data and make accurate predictions. Verify that the model improves with training and provides reliable outcomes, ensuring that the LSTM component effectively contributes to the overall classification process.

Prediction Accuracy: Validate the accuracy of predictions made by the LSTM model by comparing them with the actual outcomes. If the predicted and actual values align well, it confirms that the LSTM model is effectively capturing patterns in the data and making reliable predictions.

Performance Metrics Calculation: Verify the correctness of performance metric calculations, including accuracy, precision, recall, and F1 score. Cross-check the calculated values against expected outcomes, ensuring that metrics accurately represent the model's classification performance.

Model Saving and Loading: Test the functionality of saving and loading the trained LSTM model. Similar to saving and reloading a game, this ensures that the model retains learned information, facilitating future use without the need for retraining from scratch.

Error Handling: Evaluate the system's ability to handle errors, such as missing information or incorrect input types. Ensure that the LSTM model can gracefully manage unexpected situations, akin to a robust system continuing to function even with incomplete or erroneous input.

5.1.2. Integration Testing

Integration testing for toxic comment classification using LSTM involves examining the collaboration among different components to ensure their seamless interaction. Here are potential integration tests tailored for this context:

Data Flow: Confirm the correct flow of data by validating that the features extracted align with LSTM model requirements. Ensure the LSTM model processes these features accurately. Verify that the LSTM model effectively combines the outputs of its internal layers. This meticulous evaluation guarantees a smooth flow of information from feature extraction through the LSTM layers, culminating in reliable toxic comment classification.

Model Interactions: Validate the sharing of accurate predictions between the LSTM model and other components. Compare the predictions generated by the LSTM model with those from individual layers. Consistent and aligned predictions between the LSTM model and its internal layers affirm that the model receives reliable information, enhancing the accuracy and robustness of final predictions.

Performance Metrics Calculation: Ensure the accuracy of performance metrics calculated by the LSTM model. Compare the LSTM model's final predictions with the actual outcomes and calculate metrics like accuracy and precision. Consistent metrics indicate the LSTM model's effectiveness, similar to confirming that a student's grades accurately reflect their test performance.

System Integration: Assess the interoperability of the toxic comment classification system with other components such as the user interface or database. Run tests to verify that when the system identifies toxic comments, it communicates seamlessly with the user interface and stores pertinent information in the database. This validates the overall integration and functionality of the system.

End-to-End Testing: Conduct comprehensive testing of the entire toxic comment classification process, from inputting data to generating the final prediction. This ensures the coherent functioning of every component, akin to verifying that every piece of a puzzle fits seamlessly. End-to-end testing aids in identifying and addressing any potential issues early in the process, contributing to the reliability and accuracy of the final toxic comment classification outcome.

5.1.3. Validation Testing

Validation testing for toxic comment classification using LSTM involves evaluating the model's performance on a distinct validation dataset not utilized during training. Here are potential validation tests tailored for this context:

Model Performance: Assess the performance of the LSTM classifier by examining metrics like accuracy, precision, recall, and F1 score on the validation dataset. This validation is akin to evaluating how well a student adheres to instructions in a test, ensuring the LSTM classifier meets predefined standards for accuracy and reliability during the validation process.

Generalization: Verify if the LSTM model accurately identifies toxic comment samples it hasn't encountered during training. This test ensures the model's capability to recognize new types of toxic comments, demonstrating its effectiveness in handling unfamiliar and diverse situations.

Robustness: Evaluate whether the LSTM model maintains its performance when faced with slight variations in the validation data. Similar to checking a friend's adaptability to different situations, this test ensures the model remains accurate and reliable even when the validation data exhibits small differences, enhancing its robustness.

Efficiency: Assess the LSTM model's ability to handle validation data swiftly, similar to timing a chef to ensure efficient cooking. This test confirms that the model processes information within acceptable timeframes, demonstrating efficiency during the validation phase of testing.

Reliability: Check if the LSTM model consistently produces the same result for the same input, reflecting its reliability and stability. Comparable to expecting a consistent answer from a friend, this test ensures the model behaves predictably, fostering trust in its performance across various validation scenarios.

5.1.4. Verification Testing

Verification testing for toxic comment classification using LSTM involves confirming that the system is implemented correctly according to the specified design. Here are potential verification tests tailored for this context:

Code Review: Conduct a thorough review of the LSTM implementation to ensure it aligns with the design and follows coding standards. Similar to proofreading a story, this process guarantees that the code is coherent, functions as intended, and adheres to best practices, facilitating better comprehension and collaboration.

Static Analysis: Utilize static analysis tools as code detectives to scan the program for common mistakes and security risks without executing it. This approach acts like a spell-check for coding, identifying errors and vulnerabilities beforehand, ensuring the LSTM implementation is safer and more reliable.

Unit Testing: Test individual components of the LSTM system, verifying that each part functions correctly on its own before integration. This step is akin to checking each gear in a machine to ensure they perform their tasks accurately, identifying and addressing issues early in the development process.

Integration Testing: Assess the interaction between different components of the LSTM system, ensuring seamless communication. Similar to ensuring gears in a machine mesh perfectly, integration testing prevents glitches or errors, confirming that all components work harmoniously for accurate and efficient toxic comment classification.

System Testing: Conduct comprehensive testing of the entire LSTM system to validate its compliance with specified requirements. This process is similar to giving a thorough check to ensure everything functions as planned, identifying and rectifying any deviations to ensure the system meets expected standards.

Regression Testing: Perform regression testing to verify that updates or changes do not introduce unexpected issues. Similar to ensuring a favorite game runs after an update, regression testing ensures that modifications to the LSTM system do not disrupt its overall functionality, maintaining consistent and reliable performance.

CHAPTER 6
CONCLUSION

6 . CONCLUSION

In conclusion, the toxic comment classification project employing LSTM has yielded highly commendable outcomes, attaining a notable accuracy rate of 95.66%. The precision, recall, and F1 score metrics provide a comprehensive evaluation of the model's efficacy, underscoring its discriminative capabilities. Specifically, the precision rates of 97% for non-toxic comments and 81% for toxic comments highlight the model's precision in correctly classifying each category. Moreover, the recall metrics illuminate the model's proficiency in identifying 98% of non-toxic comments and 73% of toxic comments, emphasizing its sensitivity to both classes.

The overarching F1 score, serving as a balanced metric between precision and recall, attains a commendable 96%, signifying the model's ability to harmonize accuracy and sensitivity effectively. Delving into the detailed confusion matrix further reveals the model's adeptness in distinguishing between toxic and non-toxic comments. With a precision of 93.97% for true negatives, 17.8% for false positives, 7.3% for false negatives, and 78% for true positives, the model exhibits a robust capability to navigate the complexities of online language and discern harmful content accurately.

These substantial and nuanced metrics collectively affirm the potency and reliability of the LSTM-based approach in toxic comment classification. Beyond numerical measures, the model's success resonates in its capacity to contribute significantly to fostering a positive online environment through the adept moderation of content. This underscores the LSTM's potential as a pivotal tool in ensuring a safer and more constructive digital discourse.

CHAPTER 7
7. Scope For Future Enhancement

7. Scope For Future Enhancement

Numerous avenues for future enhancements in toxic comment classification using LSTM models present opportunities for refining and advancing the system's capabilities:

- **Advanced Feature Extraction**: Improving the extraction of crucial details from textual data can elevate the classification system's accuracy. This enhancement is akin to enhancing a detective's analytical tools, enabling the model to identify significant linguistic cues more effectively and resulting in more precise comment classification.
- **Handling Imbalanced Data**: Developing strategies to address imbalanced datasets is crucial for enhancing the model's proficiency in identifying less frequent but significant forms of toxicity. Analogous to paying special attention to less common characters in a narrative, this improvement ensures that the model doesn't overlook or underestimate the impact of less prevalent yet impactful toxic language patterns.
- **Model Interpretability**: Enhancing the interpretability of the LSTM-based classifier is essential for providing insights into its decision-making process. Similar to adding subtitles to a movie, making the model more understandable and transparent allows users to comprehend the rationale behind each classification, fostering trust in its predictions.
- **Real-Time Detection**: Optimizing the model for real-time detection capabilities is essential for swift and efficient identification and mitigation of toxic comments. This enhancement parallels having a superhero that reacts instantaneously to threats, ensuring quicker and more effective content moderation in the dynamic landscape of online platforms.
- **Adversarial Training**: Incorporating adversarial training techniques can strengthen the model against deceptive examples of toxic language. This strategy is comparable to preparing a superhero for unexpected maneuvers by simulating surprise challenges. Training with adversarial examples enables the model to handle intricate situations, fortifying its ability to combat evolving and unforeseen threats.

REFRENCES

1. Krishna Dubey, Rahul Nair, Mohd. Usman Khan, A 2020, ‘Toxic Comment Detection using LSTM’.
2. Kehan Wang, Jiaxi Yang, Hongjun Wu, A 2021, ‘A Survey of Toxic Comment Classification Methods’.
3. Zaheri, Sara; Leath, Jeff; and Stroud, David (2020) , ‘Toxic Comment Classification’ , in SMU Data Science.
4. Hao Li, Weiquan Mao, Hanyuan Liu, ‘Toxic Comment Detection and Classification’ .
5. Salvatore Carta, Andrea Corriga, Riccardo Mulas, Diego Reforgiato Recupero and Roberto Saia, in 2019, ‘ASupervised Multi-class Multi-label Word Embeddings Approach for Toxic Comment Classification’ , KDIR 2019- 11th International Conference on Knowledge Discovery and Information Retrieval.
6. Rabindra Khadka, NLP,DSSC, University of Trieste, ‘Toxic text classification using Bi-directional LSTM with Attention’.
7. Zhixue Zhao, Ziqi Zhang, Frank Hopfgartner, in 2021 , ‘A Comparative Study of Using Pre-trained Language Models for Toxic Comment Classification ’ , https://eprints.whiterose.ac.uk/173371/
8. A. Akshith Sagar, J. Sai Kiran, A 2020 , ‘Toxic Comment Classification using Natural Language Processing’ .
9. Appl. Sci. 2020, 10, 8631; doi:10.3390/app10238631
10. www. kaggle.com/c/jigsaw-toxic-comment-classification-challenge
11. Junhao ZHOU, Yue Lu, Hong-Ning Dai Hao Wang, Hong Xiao, A 2019, ‘Sentiment Analysis of Chinese Microblog Based on Stacked Bidirectional LSTM’.
12. Aytug Onan ,Mansur Alp TocoLu, A 2021, ‘A Term Weighted Neural Language Model and Stacked Bidirectional LSTM Based Framework for Sarcasm Identification’.
13. Ashok Kumar, Tina Esther Trueman ,Erik Cambria, A 2021, ‘A Convolutional Stacked Bidirectional LSTM with a Multiplicative Attention Mechanism for Aspect Category and Sentiment Detection’.