DETECTING OFFENSIVE LANGUAGE IN TAMIL YOUTUBE COMMENTS

A PROJECT REPORT

Submitted by

JISHNU . B	**-**	**20P122**
SUWINKUMAR . T	**-**	**20P156**
TANUSH . K	**-**	**20P157**
UDHAYARAJAN . M	**-**	**20P158**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KARPAGAM COLLEGE OF ENGINEERING

COIMBATORE – 641 032

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2024

KARPAGAM COLLEGE OF ENGINEERING

COIMBATORE – 641 032

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **" DETECTING OFFENSIVE LANGUAGE IN TAMIL YOUTUBE COMMENTS "** is the bonafide work of **" JISHNU. B , SUWINKUMAR. T, TANUSH. K, UDHAYARAJAN. M "** who carried out the project work under my supervision.

SIGNATURE

Dr. D. Prabhakar, M.E, Ph.D.

HEAD OF THE DEPARTMENT

Professor

Department of CSE

Karpagam College of Engineering

Coimbatore - 32

SIGNATURE

Dr. S. Arul Antran Vijay M.Tech., Ph.D

SUPERVISOR

Associate Professor

Department of CSE

Karpagam College of Engineering

Coimbatore - 32

Certified that the candidate was examined during the viva voce examinations held on_____

Signature of the Internal Examiner with date

Signature of the External Examiner with date

DECLARATION

We hereby declare that this Project report entitled **“ DETECTING OFFENSIVE LANGUAGE IN TAMIL YOUTUBE COMMENTS ”** submitted by me for the degree of **B.E in COMPUTER SCIENCE AND ENGINEERING at KARPAGAM COLLEGE OF ENGINEERING, COIMBATORE** is the record of original work done by me under the guidance and supervision of **Dr. S. ARUL ANTRAN VIJAY M.Tech., Ph.D** at the Department of Computer Science and Engineering, Karpagam College of Engineering, Coimbatore – 641032 and has not formed the basis for the award of any degree, or diploma or titles in this institution or any other Institution of higher learning.

Date :

Place : Coimbatore

Name and Signature of the Candidate(s)

Jishnu. B

Suwinkumar. T

Tanush. K

Udhayarajan. M

ACKNOWLEDGEMENT

First and foremost praises and thanks to the almighty for her showers and blessings throughout our project work to complete it successfully. We extend our gratitude to the Management of Karpagam College of Engineering, Coimbatore for the excellent infrastructure and support facilities to undergo the project work.

We are very grateful to **Dr. V. KUMAR CHINNAIYAN**, **M.E., Ph.D.,** the Principal and **Dr. D. PRABHAKAR M.E., Ph.D.,** Head of the Department of **COMPUTER SCIENCE AND ENGINEERING** for provided the facilities, support and permission to carried out our work at our esteemed institution.

We record our Project Coordinator **Dr. R. KALA M.E., Ph.D.,** for giving inputs, encouragement for the continuous improvement during the progress and to complete this project work.

We would like to express our sincere gratitude to our Supervisor **Dr. S. ARUL ANTRAN VIJAY M.Tech., Ph.D.,** for the continuous support for our UG study, for his motivation and adequate guidance which helped us to achieve success in all our accomplishments and to complete this project work.

We also thank all the teaching faculty members and non-teaching Staff members of the department of **COMPUTER SCIENCE AND ENGINEERING**, Karpagam College of Engineering, Coimbatore for their kindness and support. I would like to our parents, family members and friends who sacrificed their time and energy to complete the project work successfully.

NAME OF THE CANDIDATE(S)

Jishnu .B

Suwinkumar .T,

Tanush .K,

Udhayarajan .M

TABLE OF CONTENTS

ABSTRACT

Online content moderation faces significant challenges in identifying offensive language across diverse linguistic environments. This study compares the performance of five advanced BERT models mBERT, BERT Base, BERT Large, DistilBERT, and RoBERTa in detecting offensive language specifically in Tamil YouTube comments. Through meticulous evaluation using metrics like accuracy, precision, recall, and F1- score, we analyze the nuanced processing capabilities of these models tailored to address the complexities of the Tamil language. Among the models assessed, mBERT emerges as a computationally efficient option due to its multilingual efficacy, while RoBERTa demonstrates impressive accuracy with fine-tuning on a Tamil-specific corpus. BERT Base exhibits solid performance, albeit with higher computational demands, and BERT Large showcases enhanced accuracy at the cost of increased resource requirements. DistilBERT presents a memory-efficient alternative, maintaining reasonable performance with a smaller model size. Nonetheless, all models face challenges inherent in processing Tamil YouTube comments, such as code-mixing and informal language usage. This comparative analysis not only aids in selecting the most suitable BERT model for Tamil language content moderation but also sheds light on the broader landscape of offensive language detection in multilingual contexts. Whisper AI serves as a pivotal component for enriching offensive language detection on Tamil Contents on YouTube. By integrating Whisper AI for audio-to-text conversion, the project augments its capabilities by extracting textual content from audio inputs. RoBERTa achieved the accuracy of 88%, indicating that it made the most correct predictions overall. BERT Large and BERT Base also performed well, with accuracies of 87% and 86%, respectively. mBERT and DistilBERT performed with accuracies of 85% and 84% respectively.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBRIVIATIONS

S. No	CLASSIFIER	SHORT NAME
1	Random Forest	RF
2	Support Vector Machine	SVM
3	Multinomial Naive Bayes	MNB
4	Decision Tree	DT
5	Light Gradient Boosting Tree	LGBM
6	Ensemble With Decision Tree	EWDT
7	Ensemble Without Decision Tree	EWODT
8	Bidirectional Encoder Representations from Transformers	BERT
9	Masked and Permuted Network	MP Net
10	Convolutional Neural Network	CNN
11	Multilingual Bidirectional Encoder Representations from Transformers	m-BERT
12	Robustly optimized Bidirectional Encoder Representations from Transformers approach	RoBERTa

CHAPTER - 1
INTRODUCTION

The advent of social media has aided in bridging both political borders and paved the path for individuals to interact with others and express themselves more readily than at any prior point in human history [2]. Through the usage of social media platforms, such as Twitter, Facebook, YouTube, Instagram, WhatsApp, Snapchat, and LinkedIn, enormous amounts of information are generated, which allow for data mining and simulation modelling. While microblogging is a relatively new communication medium in comparison with traditional media, it has garnered significant interest from users, organisations, and experts from a variety of sectors [8].

At present, the COVID-19 virus is wreaking havoc all over the world. Several studies have revealed the age distribution of users who used offensive phrases on social media during the COVID-19 pandemic, with the 18–24 and 25–34 age groups accounting for 49 percent of all users [13]. Thus, it can be said the public fear sparked by rumours and offensive comments on social media is more concerning than the virus's impact [14]. Nonetheless, the complexity of event recognition algorithms has hampered the effectiveness of most offensive language detection approaches. While the categorisation of offensive languages using social media data has remained a dynamic area of study, little attention has been paid to the creation of a data, threshold settings, and models for low-resource languages [11]. The detection of abusive language in social media sources relies on a variety of approaches from several domains, including machine learning (ML), natural language processing (NLP), data mining, content extraction and retrieval, and text mining. However, social media streams from multilingual nations such as India contain a high proportion of mixed languages; this has a detrimental effect on the effectiveness of categorisation algorithms [16].

In recent years, there have been substantial improvements in research on hate speech identification and offensive language detection [17] utilising NLP.

However, there is still a dearth of research on under-resourced languages. For instance, under-resourced languages such as Tamil, Malayalam, and Kannada lack NLP tools and datasets [18]. Recently, [19] sentiment analysis as well as language identification for Tamil and Malayalam have paved the way for further studies on Dravidian languages. Tamil, Malayalam, and Kannada are Dravidian languages spoken by approximately 220 million people in India, Singapore, and Sri Lanka [20]. It is critical to develop NLP systems, such as hate speech identification and offensive language detection, for indigenous languages, as the majority of user-generated content is in these languages. Deep learning approaches offer considerable potential in the process of classification detection. However, only a few existing studies demonstrate the value of ensemble approaches in detecting offensive language in low-resource Dravidian languages.

The rest of our work is organised as follows. Chapter 2 discusses the literature on offensive language identification, while Chapter 3 focuses on the several models and approaches for identifying offensive languages in Dravidian languages. Chapter 5 introduces the dataset used for the task at hand. Chapter 6 discusses the details of the implementation of detection methods. Chapter 7 comprises a detailed analysis on the behaviour and results of the models, which are also compared with other approaches. Finally, Chapter 8 concludes our work and discusses the potential directions for future work on offensive language identification in Dravidian languages.

The increasing number of online platforms for user-generated content enables more people to experience freedom of expression than ever before. In addition, users of these platforms have the option of being anonymous and hiding their personal identities, which can increase the chance that they will misuse these technical features. Offensive language online creates an exclusive environment and, in more severe cases, it can foster real-world violence [21]. The use of offensive language has become one of the most common problems on social networking platforms. According to a study by the Pew Research Center in 2015,

67% of people in the U.S. agree they should be able to publicly make offensive statements against minority groups [23]. Some countries have issued laws to ban hate speech on social networking platforms. For example, in 2017, Germany passed the Network Enforcement Act, a law that requires social media

In the era of social computing, the interaction between individuals becomes more striking, especially through social media platforms and chat forums. Microblogging applications opened up the chance for people worldwide to express and share their thoughts instantaneously and extensively. Driven, on one hand, by the platform's easy access and anonymity. And, on the other hand, by the user's desire to dominate debate, spread / defend opinions or argumentation, and possibly some business incentives, this offered a fertile environment to disseminate aggressive and harmful content. Despite the discrepancy in hate speech legislation from one country to another, it is usually thought to include communications of animosity or disparagement of an individual or a group on account of a group characteristic such as race, color, national origin, sex, disability, religion, or sexual orientation.

Benefiting from the variation in national hate speech legislation, the difficulty to set a limit to the constantly evolving cyberspace, the increased need of individuals and societal actors to express their opinions and counter-attacks from opponents and the delay in manual check by internet operators, the propagation of hate speech online has gained new momentum that continuously challenges both policy-makers and research community. With the development in natural language processing (NLP) technology, much research has been done concerning automatic textual hate speech detection in recent years. A couple of renowned competitions have held various events to find a better solution for automated hate speech detection. In this regard, researchers have populated large-scale datasets from multiple sources, which fueled research in the field. Many of these studies have also tackled hate speech in several non-English languages and online communities. This led to investigate and contrast various processing pipelines, including the

choice of feature set and Machine Learning (ML) methods (e.g., supervised, unsupervised, and semi-supervised), classification algorithms (e.g., Naives Bayes, Linear Regression, Convolution Neural Network (CNN), LSTM, BERT deep learning architectures, and so on).

The limitation of the automatic textual-based approach for efficient detection has been widely acknowledged, which calls for future research in this field. Besides, the variety of technology, application domain, and contextual factors require a constant up-to-date of the advance in this field in order to provide the researcher with a comprehensive and global view in the area of automatic HT detection. Extending existing survey papers in this field, this paper contributes to this goal by providing an updated systematic review of literature of automatic textual hate speech detection with a special focus on machine learning and deep learning technologies.

We frame the problem, its definition and identify methods and resources employed in HT detection. We adopted a systematic approach that critically analyses theoretical aspects and practical resources, such as datasets, methods, existing projects following PRISMA guidelines. In this regard, we have tried to answer the following research questions:

- Q1: What are the specificities among different HS branches and scopes for automatic HS detection from previous literature?
- Q2: What is the state of the deep learning technology in automatic HS detection in practice?
- Q3: What is the state of the HS datasets in practice?
- Q4: How can offensive language be detected in real-time, especially in fast-paced online environments like social media platforms?
- Q5: What are the challenges and opportunities in integrating text-based and non-textual content analysis?

- Q6: How can offensive language detection systems adapt to the constantly evolving nature of language and emerging trends in online communication?

The above-researched questions will examine barriers and scopes for the automatic hate speech detection technology. A systematic review-based approach is conducted to answer Q1 and Q2, where we will try to depict and categorize the existing technology and literature. The third research question Q3, will be answered by critically examining the scope and boundaries of the dataset identified by our literature review, highlighting the characteristics and aspects of the available resources.

This review paper is organized as follows: Chapter 1 will include a brief theoretical definition of HS. Chapter 2 details the systematic literature review document collection methodology. Chapter 6 presents the results of this literature review, including the state of deep learning technology. Chapter 5 emphasizes on the available resources (datasets and open-source projects). After that, in Chapter 8, an extensive discussion is carried out. Finally, we have highlighted future research directions and conclusions at the end of this paper.

CHAPTER - 2
LITERATURE REVIEW

Author : ***Malak Aljabri, Rachid Zagrouba, Afrah Shaahid, Fatima Alnasser, Asalah Saleh and Dorieh M. Alomari [1]***

Published Year: 2023

Summary: In this modern world, OSNs such as Twitter, Facebook, Instagram, LinkedIn have become a crucial part of each one's life. It radically impacts daily human social interactions where users and their communities are the base for online growth, commerce, and information sharing. Different social networks offer a unique value chain and target different user segments. For instance, Twitter is known for being the most famous microblogging social network for receiving rapid updates and breaking news. While Instagram usage is mainly by celebrities and businesses for marketing.

Author: ***Bharathi Raja Chakravarthi, Manoj Balaji Jagadeeshan, Vasanth Palanikumar and Ruba Priyadharshini [2]***

Published Year :2023

Summary: A thorough review of the techniques, algorithms, datasets, and tasks for offensive language detection in Dravidian languages. Novel MPNet and CNN fusion technique for offensive language detection in low-resource Dravidian languages. An extensive evaluation of benchmark datasets with positive results. The appeal of micro blogging originates from its unique characteristics, such as portability, instant messaging, and user-friendliness; these capabilities enable real-time communication with little or no content limitations.

Author: ***Md Saroar Jahan and Mourad Oussalah [3]***

Published Year: 2023

Summary: With the multiplication of social media platforms, which offer anonymity, easy access and online community formation and online debate, the issue of hate speech detection and tracking becomes a growing challenge to society, individual, policy-makers and researchers. Despite efforts for leveraging automatic techniques for automatic detection and monitoring, their performances are still far from satisfactory, which constantly calls for future research on the issue. In the era of social computing, the interaction between individuals becomes more striking, especially through social media platforms and chat forums.

Author: ***Tanjim Mahmud, Michal Ptaszynski, Juuso Eronen and Fumito Masui [4]***

Published Year: 2023

Summary: Focused on the detection of vulgar re - marks in social media posts using ML and DL classifiers, namely LR, SVM, DT, RF, MNB, simple RNN, and LSTM with various feature extraction techniques such as Count Vectorizer, TF-IDF Vectorizer, Word2vec and fast Text. We have constructed a dataset of 2,485 comments where vulgar and non-vulgar were evenly distributed. The LR with Count Vector- izer, or TF-IDF Vectorizer, as well as simple RNN with Word2vec and fast Text were an effective approach for detecting vulgar remarks.

Author: ***S.V.Kogilavani, S. Malliga, K.R. Jaiabinaya, M. Malini and M.Manisha Kokila [5]***

Published Year: 2023

Summary: With the recent focus on machine learning and deep learning, this review article investigated various machine learning and deep learning models for offensive language detection and toxic comment identification, and compared the significant efforts made by researchers. This investigation provided a cognizant review on each model along with the language in which the model is developed, techniques used for classification, datasets usage.

Author : ***Hossam Elzayady, Mohamed S. Mohamed, Khaled Badran, Gouda I. Salama [13]***

Published Year: 2023

Summary: Recent studies show that social media has become an integral part of everyone's daily routine. People often use it to convey their ideas, opinions, and critiques. Consequently, the increasing use of social media has motivated malicious users to misuse online social media anonymity. Thus, these users can exploit this advantage and engage in socially unacceptable behavior. The use of inappropriate language on social media is one of the greatest societal dangers that exist today. Therefore, there is a need to monitor and evaluate social media postings using automated methods and techniques. The majority of studies that deal with offensive language classification in texts have used English datasets.

Author : ***Ayoub Ali, Alaa Y. Taqa [6]***

Published Year: 2023

Summary: Text similarity is commonly used method for detection of textual plagiarism. Pure PD is a very complex problem and only a small fraction are detected worldwide because suspicious documents have several formats and existence of different citation styles makes it impossible to extract each cited text.

After comparing the proposed algorithms' performance in TPDM based on two terms: similarity ratios and running times. Experimental results for five suspicious documents in the corpus showed that MLSPD has a better average similarity ratio of 30.3% and average running time ratio of 458.49 seconds. milliseconds.

Author : *Rishabh Jain, Andrei Barcovschi, Mariam Yiwere, Peter Corcoran, Horia Cucu [34]*

Published Year: 2023

Summary : the recent SOTA large-scale supervised Whisper models for experimental analysis over different child speech datasets. Finetuning Whisper models achieved significant

improvements in accuracy of child speech recognition. Whisper improves ASR performance for both adult and child speech, regardless of the finetuning dataset, wav2vec2 model performs better with finetune-specific datasets. the use of smaller-sized models, such as wav2vec2, would be more feasible for deployment on edge devices, which is also using 10 times less training data than Whisper.

Author : *Ashwin Rao [35]*

Published Year: 2023

Summary: Audio transcripts and sub-captions help alleviate the issues in the audio quality and enable the viewers to receive a correct interpretation of the content. For instance, Gernsbacher has shown that captions are particularly beneficial for persons watching videos in their non-native language. Although generating transcripts has been non-trivial, recent advances in speech-to-text generation have shown promising results in transcribing audio content.

Author: *Tharindu Ranasinghe, Isuri Anuradha, Damith Premasiri, Kanishka Silva, Hansi Hettiarachchi, Lasitha Uyangodage and Marcos Zampieri [14]*

Published Year: 2022

Summary: In this paper, we presented a comprehensive evaluation of Sinhala offensive language identification along with two new resources: SOLD and SemiSOLD. SOLD contains 10,500 tweets annotated at sentence-level and token-level, making it the largest manually annotated Sinhala offensive language dataset to date. SemiSOLD is a larger dataset of more than 145,000 instances an- notated with semi-supervised methods. Both these results open exciting new avenues for research on Sinhala and other low-resource languages.

Author: ***Fatimah Alkomah and Xiaogang Ma [7]***

Published Year: 2022

Summary: New datasets of hate speech from different regions with various topics of hate speech and offensive content are constantly being developed. However, many datasets are small in size whilst others lack reliability due to how the data were collected and annotated. Additionally, many datasets are small or sparse, lacking linguistic variety. Above all, the language of the content and region where the data were collected from social media make the comparison between various hate speech detection models difficult.

Author : ***Cesa Salaam, Franck Dernoncourt, Trung H. Bui, Danda B Rawat [8]***

Published Year: 2022

Summary: The human-annotated testsets for the under-resourced en-fr, en-de, en-es language combina-tions (approximately 10k). Additionally, pro-posed a synthetic code-switched data generationalgorithm for training purposes in low resourceddomains. Using this algorithm, it generated asynthetic offensive-content dataset comprised of 30k entries for en-fr, en-de, en-es language com-binations and

create two baseline models and report their results on the human-annotated test-sets. Expect this resource will enable the re-searchers to address new and exciting problems incode-switching research.

Author: ***Bedour Alrashidi, Amani Jamal, Imtiaz Khan and Ali Alkhathlan [9]***

Published Year: 2022

Summary: Firstly, we defined the abusive language and its anti-social behavior categories. Secondly, this article provides a review of the abusive content automatic detection approaches and tasks. In brief, we discussed three research questions to investigate, understand and analyze the existing works in this area. Accordingly, after a comprehensive review we propose a new taxonomy that covers five different aspects and related tasks for the abusive content automatic detection problem. The proposed taxonomy includes, namely, the data and resources, categories and annotation types, pre-processing and feature representation, models and approaches, and the evaluation metrics.

Author : ***Md Saroar Jahan, Mainul Haque, Nabil Arhab, Mourad Oussalah [10]***

Published Year: 2022

Summary: The majority of this research has concentrated on English, although one notices the emergence of multilingual detection tools such as multilingual-BERT (mBERT). However, there is a lack of hate speech datasets compared to English, and a multilingual pre-trained model often contains fewer tokens for other languages. This paper attempts to contribute to hate speech identification in Finnish by constructing a new hate speech dataset that is collected from a popular forum (Suomi24). Furthermore, we have experimented with FinBERT pre-trained model performance for Finnish hate speech detection compared to state-of-the-art mBERT and other practices.

Author: ***Md Saroar Jahan, and Mourad Oussalah [11]***

Published Year: 2021

Summary: The findings indicate that initially SVM algorithm and various types of TF-IDF features were the most widely used. However, after the advancement in deep-learning technology, a rapid change in the hate speech analysis methods was observed. The research community preferred to use different kinds of word embedding with CNN and RNN architectures. From 2017 to 2021, several comparative studies have shown the merits of deep-learning models including CNN, RNN using word2Vec, GloVe, FastText, among other embedding as compared to traditional machine learning models such as SVM, LR, NB, and RF models.

Author: ***Salim Sazzed [12]***

Published Year: 2021

Summary: With the surge of user-generated content online, the detection of vulgar or abusive language has become a subject of utmost importance. While there have been few works in hate speech or abusive content analysis in Bengali, to the best of our knowledge, this is the first attempt to thoroughly analyze the existence of vulgarity in Bengali social media content.

Author: ***Moin Khan, Khurram Shahzad, Kamran Malik [15]***

Published Year: 2021

Summary: To annotate the candidate corpus, employed an iterative approach to develop concreted sets of guidelines. The first set of guidelines can be used to distinguish between offensive and neutral tweets; the second set of guidelines

separates simple sentences from complex sentences. The third set of guidelines were developed for simple sentences to differentiate between offensive and hate-speech sentences. The final set of guidelines were developed for complex sentences to differentiate between offensive and hate- speech sentences. The guidelines were used to annotate the candidate corpus in order to generate the hate-speech corpus.

Author：***Charangan Vasantharajan and Uthayasanker Thayasivam [16]***

Published Year: 2021

Summary: Offensive Language detection in social media platforms has been an active field of research over the past years. In non-native English-speaking countries, social media users mostly use a code-mixed form of text in their posts/comments. This poses several challenges for offensive content identification tasks and considering the low resources available for the Tamil language, the task becomes much more challenging. The current study presents extensive experiments using multiple deep learning and transfers learning models to detect offensive content on YouTube. transformer networks like BERT, DistilBERT, and XLM-RoBERTa. The experimental results showed that ULMFiT is the best model for this task.

Author : ***Dorothee Beermann, Laurent Besacier, Sakriani Sakti and Claudia Soria [17]***

Published Year: 2020

Summary: Understanding the sentiment of a comment from a video or an image is an essential task in many applications. Sentiment analysis of a text can be useful for various decision-making processes. One such application is to analyse the popular sentiments of videos on social media based on viewer comments.

However, comments from social media do not follow strict rules of grammar, and they contain mixing of more than one language, often written in non-native scripts.

Author: Hossam Elzayady, Mohamed S. Mohamed, Khaled M. Badran, Gouda I. Salama *[18]*

Published Year: 2022

Summary: Posting offensive or abusive content on social media have been a serious concern in recent years. This has created a lot of problems because of the huge popularity and usage of social media sites like Facebook and Twitter. The main motivation lies in the fact that our model will automate and accelerate the detection of the posted offensive content so as to facilitate the relevant actions and moderation of these offensive posts. The final comparative analysis concluded that the best model came out to be Bidirectional Encoder Representation from Transformer (BERT).

Author : *Fabio Poletto, Valerio Basile, Manuela Sanguinetti, Cristina Bosco & Viviana Patti [19]*

Published Year: 2020

Summary: The high number of resources and benchmark corpora for many different languages developed in a very narrow time span, from 2016 onward, confirms the growing interest of the community around abusive language in social media and HS detection in particular. Being the subject in a yet recent stage, it suffers from several weaknesses, related to both the specific targets and nuances of HS and the nature of the classification task at large, that represent an obstacle toward reaching optimal results.

Author : ***Rahul Pradhan, Ankur Chaturvedi, Aprna Tripathi, Dilip Kumar Sharma [20]***

Published Year: 2020

Summary : The work done recently in this field of automatic detection of offensive language and the research goes from using tf-idf to popular classifiers such as Naïve Bayes, Support vector machine (SVM), Logistic Regression, then research work goes to variant of these classifiers such as Linear SVM, Logistic Regression with L2, from here researchers further explore ensemble classifiers using the combination of these classifier by decomposing the task into sub tasks, then lastly usage of deep learning and we found many researcher using approaches such as LSTM, CNN and RNN.

Author : ***Alec Radford Jong Wook Kim Tao Xu Greg Brockman Christine McLeavey Ilya Sutskever [33]***

Published Year: 2019

Summary : The unsupervised pre-training has improved the quality of audio encoders dramatically, the lack of an equivalently high-quality pre-trained decoder, combined with a recommended protocol of dataset-specific finetuning, is a crucial weakness which limits their usefulness and robustness.

CHAPTER 3

SYSTEM ANALYSIS

3.1 PROBLEM STATEMENT:

The task at hand involves tackling the pervasive issue of offensive language in Tamil YouTube content through a series of comprehensive problem statements. Firstly, developing a robust machine learning or deep learning model is crucial for accurately identifying offensive language within Tamil YouTube comments. The evaluation of this model's performance on a diverse dataset is essential, considering factors like accuracy, precision, recall, and F1-score, to ensure its effectiveness. Moreover, handling the intricacies of informal language, slang, and contextual nuances inherent in Tamil comments poses a significant challenge that demands exploration. In parallel, the development of a real-time detection system capable of swiftly identifying offensive language in Tamil YouTube comments is imperative. Integrating this system with YouTube's API enables continuous monitoring, facilitating timely intervention to maintain a positive online environment. Scalability and efficiency are paramount to handle the substantial influx of comments efficiently. Expanding the scope to include audio content, the creation of an automatic transcription system for Tamil YouTube audio clips enables text-based offensive language detection. Overcoming challenges such as accent variation, background noise, and speech recognition errors in Tamil audio requires innovative techniques and robust algorithms. A further advancement involves the integration of text and audio analysis in a multimodal offensive language detection system. Extending offensive language detection beyond Tamil to other languages commonly spoken in Tamil YouTube videos is essential for comprehensive content moderation. Finally, implementing a user feedback mechanism empowers viewers to actively participate in content moderation. Algorithms analyzing reported content prioritize detection efforts, fostering a collaborative approach to maintain a safe and inclusive online community.

3.2 EXISTING APPROACHES:

1. **Lexicon-based approaches:** These methods rely on predefined lists of offensive words, phrases, or patterns. Offensive language is flagged based on the presence of such terms in the text. While simple and computationally efficient, lexicon-based approaches may struggle with context-dependent or creative language usage [7].

2. **Machine Learning (ML) approaches:** ML models, particularly supervised learning algorithms like Support Vector Machines (SVM), Random Forests, or neural networks, are trained on labeled datasets containing examples of offensive and non-offensive language. These models learn to classify text based on various features such as word n-grams, part-of-speech tags, or semantic representations. Deep learning architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown promising results in capturing complex patterns in text data [1].

3. **Hybrid approaches:** Combining lexicon-based and machine learning methods can leverage the strengths of both approaches. For instance, lexicon-based features can be used alongside machine learning models to enhance performance, especially in cases where training data is limited [9].

4. **Contextual approaches:** Understanding the context of language use is crucial for accurately detecting offensive content. Contextual features such as user interactions, historical behavior, or conversation dynamics can provide valuable insights into the intent behind the language. Context-aware models, which consider not only the text but also the broader context in which it appears, are increasingly being explored for offensive language detection [16].

5. **Ensemble methods:** Combining multiple models, each trained using different algorithms or features, can often lead to better performance than individual models.

Ensemble methods like bagging, boosting, or stacking can help mitigate the weaknesses of individual classifiers and improve overall accuracy [21].

6. **Naive Bayes Classifier:** Naive Bayes classifiers are probabilistic models based on Bayes' theorem with the assumption of independence between features. Despite their simplicity, Naive Bayes classifiers can perform reasonably well in low-resource scenarios, especially when combined with simple text representations like BoW. They are computationally efficient and require relatively small amounts of training data [4].

7. **Ensemble of Weak Classifiers:** In low-resource settings, combining multiple weak classifiers into an ensemble can improve overall performance. Techniques like bagging or boosting can be employed to aggregate predictions from multiple simple models, mitigating the limitations of individual classifiers. By leveraging the diversity of weak learners, ensemble methods can achieve competitive performance with minimal computational overhead [28].

3.2.1 Limitations of Existing Approaches:

Existing models for the above approaches have made significant advancements, but they still have several limitations:

Contextual Understanding: Models often struggle to understand the nuances of language and context. Offensive content can be highly context-dependent, and a model may misinterpret a benign statement as offensive or fail to detect offensive language when it's used in a subtle or disguised manner.

Multimodal Content: Many social media posts contain not only text but also images, videos, and audio. Existing models primarily focus on text-based analysis, which means they might miss offensive content presented in other media formats.

Evolving Language and Trends: Offensive language and trends on social media are constantly evolving. Models trained on historical data may not be up-to-date with the latest terminology and forms of online harassment or offensive content.

False Positives and Negatives: Models often generate false positives and false negatives. Achieving the right balance between these two is a challenging task.

Language Variability: Offensive content can vary significantly across languages and cultures. Models trained on one language or cultural context may not perform well in others.

Adversarial Attacks: Malicious users may intentionally try to fool offensive content detection systems, and models can be vulnerable to adversarial attacks, where slight modifications to the text can bypass the system's filters.

3.3 PROPOSED SYSTEM:

Tamil, renowned for its diverse dialects and colloquialisms, presents a unique challenge in content moderation, especially in the realm of hate speech detection. The dynamic nature of language usage on social media platforms further complicates this task. To address this challenge, we propose a model that heavily emphasizes audio processing, particularly in detecting hate speech within audio files, leveraging the power of Whisper AI. Whisper AI, an automatic speech recognition (ASR) system trained on a vast corpus of multilingual data, holds promise in deciphering nuanced speech patterns inherent to Tamil.

The methodology involves adapting and fine-tuning BERT models to comprehend Tamil subtitles, an area that has been underrepresented in natural language processing (NLP) research. Specifically, the study explores the efficacy of mBERT, BERT base, BERT large, DistilBERT, and RoBERTa models in accurately pinpointing offensive language in Tamil. This assessment takes into account the language's unique linguistic characteristics and cultural context.

The integration of Whisper AI augments our model's capability to analyze hate speech within audio files. With its extensive training on diverse linguistic data, Whisper AI provides valuable insights into the subtle nuances of spoken Tamil, further enhancing the accuracy of our hate speech detection system.

By addressing the crucial challenges of hate speech detection in Tamil, our research aims to contribute significantly to the advancement of more inclusive and efficient NLP tools for content moderation.

PROPOSED SOLUTION ARCHITECTURE:

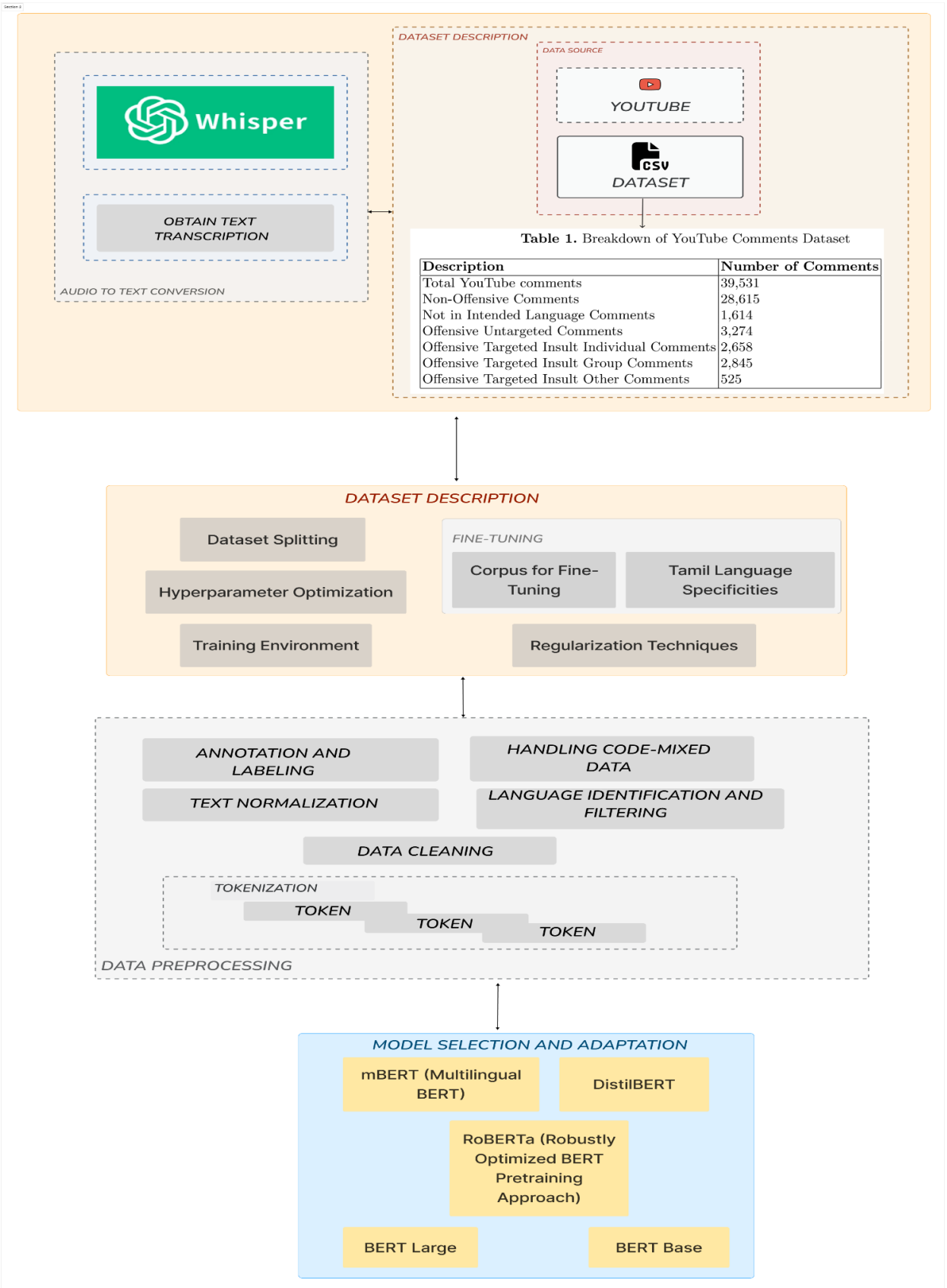

Fig No 3.2 System Architecture

Advantages of Our Approach:

1. **Enhanced Accuracy:** The evaluation of multiple BERT models, including RoBERTa fine-tuned for Tamil, ensures superior accuracy in identifying offensive language, thereby enhancing the overall effectiveness of the detection process.

2. **Precision in Language Detection:** The findings provide valuable insights into the precision of each BERT model, enabling more precise identification of offensive language in Tamil comments on YouTube, thereby reducing false positives.

3. **Adaptability to Resource Constraints:** With models like DistilBERT offering a balance between performance and resource utilization, the approach caters to resource-constrained environments, making offensive language detection accessible across different computational setups.

4. **Multilingual Effectiveness:** mBERT's effectiveness in identifying offensive language in Tamil demonstrates its multilingual capabilities, making it adaptable to various languages and linguistic nuances, thus broadening the scope of application beyond a single language.

5. **Robust Performance:** Despite computational demands, BERT Large demonstrates robust performance, particularly in terms of accuracy and recall, making it suitable for applications prioritizing precision and comprehensive detection of offensive language.

6. **Context Enrichment:** Integration of Whisper AI for audio-to-text conversion enriches the offensive language detection process by providing additional context from audio inputs, thereby improving the accuracy and contextual understanding of offensive language in comments.

CHAPTER 4

REQUIREMENTS

4.1 HARDWARE SPECIFICATIONS

Processor - CPU 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz

RAM - 16 GB DDR4@3200MHz

Storage - 512 GB M.2 NvMe SSD

GPU - RTX 3040

4.2 SOFTWARE SPECIFICATIONS

OS - Windows 8, 10 or 11

Chrome - Version 118.0.6045.106 (Official Build) (64-bit)

Python – 3.11.5

- absl-py-2.0.0
- anyio-4.0.0
- argon2-cffi-23.1.0
- argon2-cffi-bindings 21.2.0
- arrow-1.3.0
- asgiref-3.7.2
- asttokens-2.4.0
- astunparse-1.6.3
- async-lru-2.0.4
- attrs-23.1.0
- Babel-2.13.0
- backcall-0.2.0
- beautifulsoup4-4.12.2
- bleach-6.1.0
- cachetools-5.3.1
- certifi-2023.7.22
- defusedxml-0.7.1
- Django-4.2.6
- django-emoji-2.2.2
- emoji-2.8.0
- exceptiongroup-1.1.3
- executing-2.0.0
- fastjsonschema-2.18.1
- fasttext-0.9.2
- flatbuffers-23.5.26
- fqdn-1.5.1
- gast-0.5.4
- google-auth-2.23.3
- google-auth-oauthlib-1.0.0
- google-pasta-0.2.0
- grpcio-1.59.

CHAPTER 5

MODULE DESCRIPTION

5.1. DATASET PREPARATION AND PREPROCESSING

5.1.1 DATASET:

Online media, for example, Twitter, Facebook or YouTube, contain quickly changing data produced by millions of users that can drastically alter the reputation of an individual or an association. This raises the significance of programmed extraction of sentiments and offensive language used in online social media. YouTube is one of the popular social media platforms in the Indian subcontinent because of the wide range of content available from the platform such as songs, tutorials, product reviews, trailers and so on. YouTube allows users to create content and other users to comment on the content. It allows for more user-generated content in under-resourced languages. Hence, we chose YouTube to extract comments to create our dataset. We chose movie trailers as the topic to collect data because movies are quite popular among the Tamil speaking populace. This increases the chance of getting varied views on one topic. Figure 1 shows the overview of the steps involved in creating our dataset.

We compiled the comments from different film trailers of Tamil, Kannada, and Malayalam languages from YouTube in the year 2019. The comments were gathered using YouTube Comment Scraper tool. We utilized these comments to make the datasets for sentiment analysis and offensive language identification with manual annotations. We intended to collect comments that contain code-mixing at various levels of the text, with enough representation for each sentiment and offensive language classes in all three languages. It was a challenging task to extract the necessary text that suited our intent from the comment section, which was further complicated by the presence of remarks in other non-target languages.

DATASET	NUMBER OF COMMENTS
Data Collected	39,531
Non - Offensive Comments	28,615
Not in Intended Language Comments	1,614
Offensive Untargeted Comments	3,274
Offensive Targeted Insult Individual Comments	2,658
Offensive Targeted Insult Group Comments	2,845
Offensive Targeted Insult Other Comments	525

Table 5.1.1 tamil offensive full train Dataset

DATASET	NUMBER OF COMMENTS
Data Collected	1014
Non - Offensive Comments	328
Offensive Untargeted Comments	256
Offensive Targeted Insult Individual Comments	183
Offensive Targeted Insult Group Comments	140
Offensive Targeted Insult Other Comments	108

Table 5.1.2 tamil audio test Dataset

Code - Switching Type	Example	Translation
No Code - Mixing Only Tamil (Written in Tamil Script)	இது மாதிரி ஒரு படத்தை தான் இத்தனை வருஷமா எதிர்பார்த்து கொண்டிருந்தேன் டிரெய்லர் பார்க்கும்போதே மனசுக்கு அவ்வளவு சந்தோசமா இருக்கு	How long I have been Waiting for this kind of Movie! Feels joyful just Watching this Trailer
Inter - Sentential Code - Mixing Mix of English and Tamil (Tamil Written only in Tamil Script)	ரெட்டியார் சமுகம் சார்பாக படம் வெற்றி பெற வாழ்த்துக்கள்… Mohan G All the Very Best we all behind you.	On behalf of Reddiyar Community, I am Wishing the Best for this Movie. Mohan G All the Very Best. We all behind you.
Only Tamil (Written in Latin Script)	Inga niraya perukku illatha kedda palakkam enkidda irukku. vassol mannan VIJAY anna.	'I have a bad habit which is not found here in others'. Brother Vijay is the king of blockbusters.
Code - Switching at morphological level (Written in both Tamil and Latin Script)	ஓ விஜய் படத்துக்கு இப்படிதான் viewers sekkurangala	Oh. So this is how you gather more viewers for Vijay's movie ?
Intra - Sentential mix of English and Tamil (Written in Latin Script Only)	Patti thotti engum pattaya kelaputha ne va thalaiva I am waiting	Rocking Performance that will be a hit among every type of audience. Come on, my star. I am Waiting

Code - Switching Type	Example	Translation
Inter - Sentential and Intra - Sentential Mix (Tamil Written in Both Tamil Script and Latin Script)	இந்த படத்த வர விட கூடாது... இந்த படத்த திரையரங்கில் ஓட விட கூடாதுனு எவனாது தடை பன்னா.. . Theatre la vera endha padam odunaalum screen கிழியும்	If anybody imposes a ban that this movie should not be released, that it should not be allowed to run on theaters, then the screens will be torn if any other movie is released.

Table 5.1.3: Tamil Comments Sample Data

5.1.2 DATA PREPROCESSING:

Given the complexity of the Tamil language, the preprocessing steps included tokenization, normalization, and removal of irrelevant characters. Special attention was paid to retaining the linguistic nuances critical for understanding the context and sentiment of the comments. The preprocessing of the Tamil YouTube comments dataset involved several key steps to ensure its readiness for analysis:

1. **Data Cleaning:** The initial step involved the removal of irrelevant content, including URLs, hashtags, and user mentions, using Regular Expressions and custom scripts. Additionally, non-Tamil characters and symbols were filtered out to maintain linguistic purity and focus.

2. **Language Identification and Filtering:** To ensure linguistic coherence, language detection tools were applied to filter out comments written in languages other than Tamil. This step helped maintain the integrity of the dataset and avoid noise from non-Tamil content.

3. **Text Normalization:** Standardizing the text format was crucial for reducing complexity and facilitating subsequent analysis. Techniques such as lowercasing and normalization of colloquial terms were applied to ensure consistency and uniformity across the dataset.

4. **Tokenization:** Given the unique morphological characteristics of Tamil, a custom tokenizer was employed to segment the text into tokens. This tokenization process addressed the language's intricate structure and facilitated further linguistic analysis and feature extraction.

5. **Handling Code-Mixed Data:** Recognizing the prevalence of code-mixed comments, particularly those blending Tamil with English, specific techniques were implemented to process such instances while preserving linguistic authenticity. Strategies such as transliteration normalization and language specific tokenization were employed to accurately represent code-mixed text.

6. **Annotation and Labeling:** As mentioned earlier, each comment underwent manual annotation by linguistic experts to determine its offensive or non-offensive nature. Comments were evaluated based on explicit language, tone, context, and cultural sensitivity, ensuring comprehensive labeling and annotation. Inter-annotator agreement was assessed to validate the consistency and reliability of annotations, enhancing the credibility of the ground truth dataset.

These steps collectively refined the dataset, making it suitable for effective natural language processing and analysis.

5.2. MODEL DESCRIPTION

In this study, we chose five BERT models for evaluation: mBERT, BERT Base, BERT Large, DistilBERT, and RoBERTa. To ensure their effectiveness in handling the Tamil language, each model underwent a process of adaptation. This adaptation encompassed fine-tuning the models using a Tamil-specific corpus, thereby

enhancing their ability to capture the distinctive syntactic and semantic features of the language. We chose five distinct models, each offering unique capabilities and strengths in natural language processing:

M-BERT (Multilingual BERT): The mBERT (Multilingual BERT) model is a variant of the BERT architecture that has been trained on a diverse corpus of text data from multiple languages. BERT itself stands for Bidirectional Encoder Representations from Transformers, a powerful neural network model introduced by Google in 2018.

The key innovation of mBERT lies in its ability to capture multilingual context and semantics. By pre-training on a mixture of languages, mBERT learns to represent words and sentences in a shared vector space, enabling it to understand and generate text across various languages. This shared representation allows for transfer learning, where the knowledge gained from one language can be applied to tasks in other languages, even if those languages were not seen during training.

mBERT has proven to be particularly effective in tasks such as cross-lingual document classification, cross-lingual sentence retrieval, and cross-lingual named entity recognition, among others. Its versatility and ability to handle multiple languages make it a valuable tool for multilingual natural language processing applications.

Pseudo Code :

Step 1: Import necessary libraries:

- Import the required libraries from the Transformers library:
  - BertTokenizer
  - BertForSequenceClassification

Step 2: Load the tokenizer:

- Use the ‘BertTokenizer.from_pretrained()’ method to load the tokenizer for BERT Base:

  - Pass the string ‘bert-base-multilingual-cased’ as the argument.

Step 3: Load the pre-trained BERT Base model:

- Use the ‘BertForSequenceClassification.from_pretrained()’ method to load the pre-trained BERT Base model for sequence classification:

  - Pass the string ‘bert-base-multilingual-cased’ as the first argument.

  - Specify the number of labels for your classification task using the ‘num_labels’ parameter.

Step 4: Prepare your data:

- Tokenize your input text data using the loaded tokenizer.

- Convert your tokenized text data into input features that BERT can process.

Step 5: Training:

- Define your training loop or use the provided training function.

- Iterate over your training dataset:

  - Pass the input features and labels to the BERT model.

  - Compute the loss based on the model's predictions and the ground truth labels.

  - Update the model's parameters using backpropagation.

Step 6: Evaluation:

- Evaluate the performance of your trained model on a validation dataset or using cross-validation.

- Compute evaluation metrics such as accuracy, precision, recall, and F1-score.

Step 7: Inference:

- Use the trained model to make predictions on new unseen data.

- Tokenize the input text using the same tokenizer used during training.

- Pass the tokenized input through the trained model to obtain predictions.

BERT Base: BERT, which stands for Bidirectional Encoder Representations from Transformers, is a state-of-the-art pre-trained deep learning model developed by Google AI researchers. It is designed to understand and generate natural language text. Unlike traditional models that process text in a unidirectional manner, BERT is bidirectional, meaning it can capture the context of a word by looking at both the words that come before and after it in a sentence. This bidirectional understanding enables BERT to grasp nuances and dependencies within text more effectively.

BERT is built on the transformer architecture, which has proven to be highly effective in various NLP tasks. The transformer architecture relies on self-attention mechanisms to weigh the importance of different words in a sentence, allowing the model to focus on relevant information while processing text.

The BERT base model is the smaller version of BERT, containing 12 transformer layers (compared to the larger BERT large model, which has 24 layers). Despite its smaller size, the BERT base model still consists of hundreds of millions of parameters and has been pre-trained on large corpora of text data, such as Wikipedia and the BookCorpus dataset, using unsupervised learning techniques.

After pre-training, BERT can be fine-tuned on specific downstream NLP tasks with labeled data, such as sentiment analysis, named entity recognition, text

classification, question answering, and more. Fine-tuning involves adjusting the model's parameters to better fit the characteristics of the task at hand, thereby achieving state-of-the-art performance on various NLP benchmarks. Due to its versatility and effectiveness, BERT has become a cornerstone in modern NLP research and applications.

Pseudo Code :

Step 1: Import necessary libraries:

- Import the required libraries from the Transformers library:
  - BertTokenizer
  - BertForSequenceClassification

Step 2: Load the tokenizer:

- Use the 'BertTokenizer.from_pretrained()' method to load the tokenizer for BERT Base:
  - Pass the string 'bert-base-cased' as the argument.

Step 3: Load the pre-trained BERT Base model:

- Use the 'BertForSequenceClassification.from_pretrained()' method to load the pre-trained BERT Base model for sequence classification:
  - Pass the string 'bert-base-cased' as the first argument.
  - Specify the number of labels for your classification task using the 'num_labels' parameter.

Step 4: Prepare your data:

- Tokenize your input text data using the loaded tokenizer.

- Convert your tokenized text data into input features that BERT Base can process.

Step 5: Training:

- Define your training loop or use the provided training function.

- Iterate over your training dataset:

  - Pass the input features and labels to the BERT Base model.

  - Compute the loss based on the model's predictions and the ground truth labels.

  - Update the model's parameters using backpropagation.

Step 6: Evaluation:

- Evaluate the performance of your trained model on a validation dataset or using cross-validation.

- Compute evaluation metrics such as accuracy, precision, recall, and F1-score.

Step 7: Inference:

- Use the trained model to make predictions on new unseen data.

- Tokenize the input text using the same tokenizer used during training.

- Pass the tokenized input through the trained model to obtain predictions.

BERT Large: BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art natural language processing (NLP) model developed by researchers at Google. Unlike previous models that only looked at text data in one direction (either left-to-right or right-to-left), BERT is bidirectional, meaning it considers the entire context of a word by looking at both preceding and following words simultaneously. This bidirectional approach allows BERT to capture more

comprehensive contextual information, resulting in better understanding and representation of language.

The "Large" variant of BERT refers to a version of the model that has a larger architecture, with more parameters and computational resources compared to the base version. BERT Large has 24 transformer layers (compared to 12 layers in the base version) and 340 million parameters, making it capable of capturing even more complex relationships and nuances in text data.

BERT Large has been pre-trained on a massive corpus of text data using a masked language modeling (MLM) objective and next sentence prediction (NSP) task. This pre-training process allows BERT to learn rich, generalized representations of language that can be fine-tuned for a wide range of downstream NLP tasks, such as sentiment analysis, named entity recognition, question answering, and more. Due to its impressive performance and versatility, BERT has become a cornerstone of many NLP applications and research projects.

Pseudo Code :

Step 1: Import necessary libraries:

- Import the required libraries from the Transformers library:
    - BertTokenizer
    - BertForSequenceClassification

Step 2: Load the tokenizer:

- Use the ‘BertTokenizer.from_pretrained()’ method to load the tokenizer for BERT Large:
    - Pass the string ‘bert-large-cased’ as the argument.

Step 3: Load the pre-trained BERT Large model:

- Use the ‘BertForSequenceClassification.from_pretrained()’ method to load the pre-trained BERT Large model for sequence classification:

  - Pass the string ‘bert-large-cased’ as the first argument.

  - Specify the number of labels for your classification task using the ‘num_labels’ parameter.

Step 4: Prepare your data:

- Tokenize your input text data using the loaded tokenizer.

- Convert your tokenized text data into input features that BERT Large can process.

Step 5: Training:

- Define your training loop or use the provided training function.

- Iterate over your training dataset:

  - Pass the input features and labels to the BERT Large model.

  - Compute the loss based on the model's predictions and the ground truth labels.

  - Update the model's parameters using backpropagation.

Step 6: Evaluation:

- Evaluate the performance of your trained model on a validation dataset or using cross-validation.

- Compute evaluation metrics such as accuracy, precision, recall, and F1-score.

Step 7: Inference:

- Use the trained model to make predictions on new unseen data.

- Tokenize the input text using the same tokenizer used during training.

- Pass the tokenized input through the trained model to obtain predictions.

DistilBERT: DistilBERT, short for "Distill BERT," is a derivative of the BERT (Bidirectional Encoder Representations from Transformers) model, a powerful language representation model based on the Transformer architecture.

DistilBERT is designed to be a smaller and faster alternative to BERT while still maintaining a significant portion of its performance. It achieves this through a process called knowledge distillation, where a larger and more complex model (in this case, the original BERT model) teaches a smaller model (DistilBERT) by transferring its knowledge during training. This results in a distilled model that is computationally more efficient and can be deployed in environments with limited resources, such as mobile devices or edge computing devices.

Despite its reduced size, DistilBERT retains much of the original BERT model's capability for understanding natural language, making it suitable for a wide range of natural language processing tasks such as text classification, sentiment analysis, question answering, and more. It has been widely adopted in both research and industry for its balance of performance and efficiency.

Pseudo Code:

Step 1: Import necessary libraries:

- Import the required libraries from the Transformers library:

  - DistilBertTokenizer

  - DistilBertForSequenceClassification

Step 2: Load the tokenizer:

- Use the ‘DistilBertTokenizer.from_pretrained()’ method to load the tokenizer for DistilBERT:

- Pass the string ‘distilbert-base-multilingual-cased’ as the argument.

Step 3: Load the pre-trained DistilBERT model:

- Use the ‘DistilBertForSequenceClassification.from_pretrained()’ method to load the pre-trained DistilBERT model for sequence classification:

- Pass the string ‘distilbert-base-multilingual-cased’ as the first argument.

- Specify the number of labels for your classification task using the ‘num_labels’ parameter.

Step 4: Prepare your data:

- Tokenize your input text data using the loaded tokenizer.

- Convert your tokenized text data into input features that DistilBERT can process.

Step 5: Training:

- Define your training loop or use the provided training function.

- Iterate over your training dataset:

- Pass the input features and labels to the DistilBERT model.

- Compute the loss based on the model's predictions and the ground truth labels.

- Update the model's parameters using backpropagation.

Step 6: Evaluation:

- Evaluate the performance of your trained model on a validation dataset or using cross-validation.

- Compute evaluation metrics such as accuracy, precision, recall, and F1-score.

Step 7: Inference:

- Use the trained model to make predictions on new unseen data.

- Tokenize the input text using the same tokenizer used during training.

- Pass the tokenized input through the trained model to obtain predictions.

RoBERTa RoBERTa, short for "Robustly optimized BERT approach," represents a significant advancement in natural language processing (NLP) models. Developed by Facebook AI, it builds upon the foundational architecture of BERT (Bidirectional Encoder Representations from Transformers) while introducing several key improvements. One of its notable enhancements is the adoption of dynamic masking during pretraining, which allows for varied token masking in each training iteration, thereby enriching the model's understanding of contextual relationships.

Additionally, RoBERTa utilizes larger batch sizes and undergoes longer training periods compared to BERT, enabling it to capture more intricate linguistic patterns and nuances. Unlike BERT, RoBERTa discards the Next Sentence Prediction (NSP) task during pretraining, simplifying the training process and yielding better performance. Moreover, it benefits from being trained on a larger and more diverse dataset, encompassing a broader range of textual sources. These optimizations collectively render RoBERTa more robust and capable across various NLP tasks, positioning it as a leading model in the field.

Pseudo Code:

Step 1: Import necessary libraries:

- Import the required libraries from the Transformers library:
  - RobertaTokenizer
  - RobertaForSequenceClassification

Step 2: Load the tokenizer:

- Use the ‘RobertaTokenizer.from_pretrained()’ method to load the tokenizer for RoBERTa:

  - Pass the string 'roberta-base' as the argument.

Step 3: Load the pre-trained RoBERTa model:

- Use the ‘RobertaForSequenceClassification.from_pretrained()’ method to load the pre-trained RoBERTa model for sequence classification:

  - Pass the string ‘roberta-base’ as the first argument.

  - Specify the number of labels for your classification task using the ‘num_labels’ parameter.

Step 4: Prepare your data:

- Tokenize your input text data using the loaded tokenizer.

- Convert your tokenized text data into input features that RoBERTa can process.

Step 5: Training:

- Define your training loop or use the provided training function.

- Iterate over your training dataset:

  - Pass the input features and labels to the RoBERTa model.

  - Compute the loss based on the model's predictions and the ground truth labels.

  - Update the model's parameters using backpropagation.

Step 6: Evaluation:

- Evaluate the performance of your trained model on a validation dataset or using cross-validation.

- Compute evaluation metrics such as accuracy, precision, recall, and F1-score.

Step 7: Inference:

- Use the trained model to make predictions on new unseen data.

- Tokenize the input text using the same tokenizer used during training.

- Pass the tokenized input through the trained model to obtain predictions.

Whisper AI serves as a pivotal component for enriching offensive language detection on Tamil Contents on YouTube. By integrating Whisper AI for audio-to-text conversion, the project augments its capabilities by extracting textual content from audio inputs. This integration enables the system to analyze both textual and auditory data, providing a more comprehensive understanding of comments and audio's. Whisper AI's advanced algorithms accurately transcribe spoken words into text, enabling the system to incorporate audio cues alongside textual content for more precise offensive language detection.

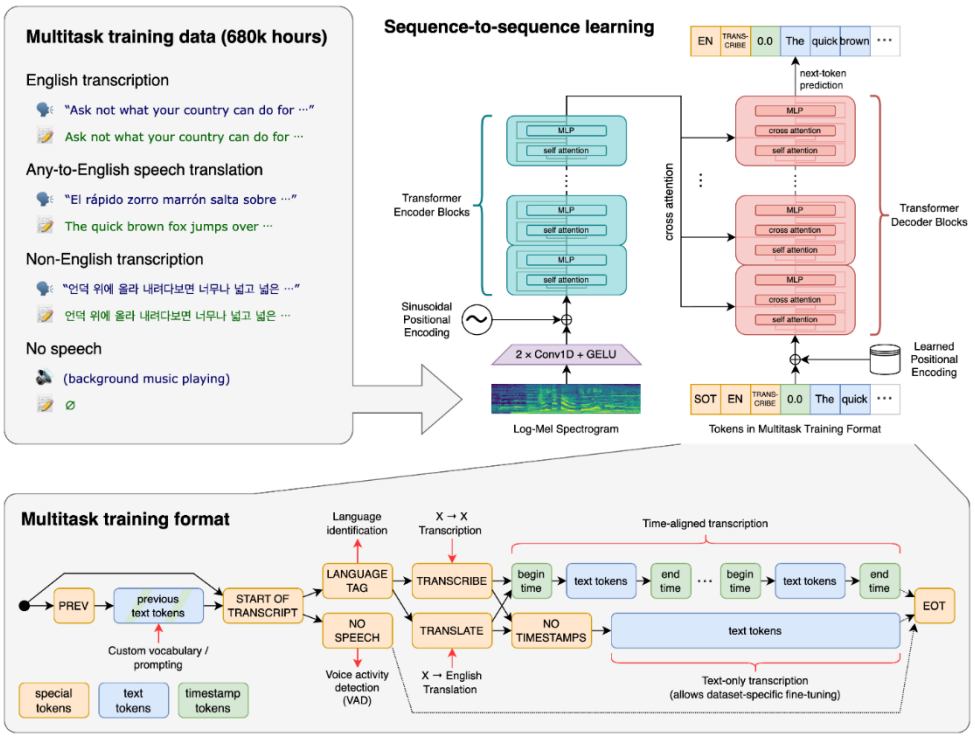

Fig No 5.2.1 Whisper Ai Architecture

5.3. TRAINING AND TESTING

A data set used for machine learning should be partitioned into two subsets - training and test sets.

Training set – It is used to train a model and define its optimal parameters it has to learn from data.

DATASET	NUMBER OF COMMENTS
Trained Dataset	35,139
Not offensive Comments	25,425
Not in Intended Language Comments	1,454
Offensive Untargeted Comments	2,906
Offensive Targeted Insult Individual Comments	2,343
Offensive Targeted Insult Group Comments	2,557
Offensive Targeted Insult Other Comments	454

Table 5.3.1: tamil train Dataset

Test set – It is needed for an evaluation of the trained model and its capability for generalization. It means a model's ability to identify patterns in new unseen data after having been trained over a training data. It's crucial to use different subsets for training and testing to avoid model over-fitting.

DATASET	NUMBER OF COMMENTS
Tested Dataset	4,392
Not offensive Comments	3,190
Not in Intended Language Comments	160
Offensive Untargeted Comments	368
Offensive Targeted Insult Individual Comments	315
Offensive Targeted Insult Group Comments	288
Offensive Targeted Insult Other Comments	71

Table 5.3.2: tamil test Dataset

DATASET	NUMBER OF COMMENTS
Data Collected	1143
Non - Offensive Comments	328
Not in Intended Language Comments	129
Offensive Untargeted Comments	256
Offensive Targeted Insult Individual Comments	183
Offensive Targeted Insult Group Comments	140
Offensive Targeted Insult Other Comments	108

Table 5.3.3 : tamil audio test Dataset

5.4. RESULT ANALYSIS:

The last phase of our methodology encompasses a comparative assessment of the performance of each BERT model. This evaluation centers on their ability to accurately identify offensive language, alongside considerations of computational

efficiency and adaptability to new data. Through this methodological approach, our study aims to provide an in-depth understanding of how different BERT models perform in the context of Tamil language processing and offer insights into the most effective models for this specific application.

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Specificity**
mBERT	0.85	0.88	0.80	0.84	0.90
BERT Base	0.86	0.89	0.82	0.85	0.91
BERTLarge	0.87	0.90	0.84	0.87	0.92
DistilBERT	0.84	0.87	0.79	0.83	0.89
RoBERTa	0.88	0.91	0.86	0.88	0.93

Table 5.4.1: Performance Comparison of BERT Models

In this work, we used metrics such as accuracy, precision, recall, F1 score, and specificity. These metrics provide a comprehensive understanding of each model's ability to correctly identify offensive language in Tamil YouTube comments.

1. **Accuracy:** RoBERTa achieved the accuracy of 88%, indicating that it made the most correct predictions overall. BERT Large and BERT Base also performed well, with accuracies of 87% and 86%, respectively. mBERT and DistilBERT had slightly lower accuracy values but still demonstrated good performance.

2. **Precision:** RoBERTa had the highest precision of 91%, implying that it made fewer false positive predictions when identifying offensive language. BERT Large and BERT Base also showed high precision values of 90% and 89%, respectively. mBERT and DistilBERT exhibited slightly lower precision but remained effective in identifying offensive language.

3. **Recall:** RoBERTa achieved the highest recall of 86%, indicating its ability to identify a significant proportion of actual offensive comments. BERT Large and BERT Base also demonstrated good recall values of 84% and 82%, respectively.

mBERT and DistilBERT had lower recall values but were still effective in capturing offensive content.

4. **F1-Score:** RoBERTa achieved the highest F1-Score of 88%, striking a balance between precision and recall. BERT Large and BERT Base also had strong F1-Scores of 87% and 85%, respectively. mBERT and DistilBERT had slightly lower F1-Scores but remained competitive.

5. **Specificity:** RoBERTa exhibited the highest specificity of 93%, indicating its ability to correctly identify non-offensive comments. BERT Large and BERT Base also demonstrated strong specificity values of 92% and 91%, respectively. mBERT and DistilBERT had slightly lower specificity values but still effectively identified non-offensive content.

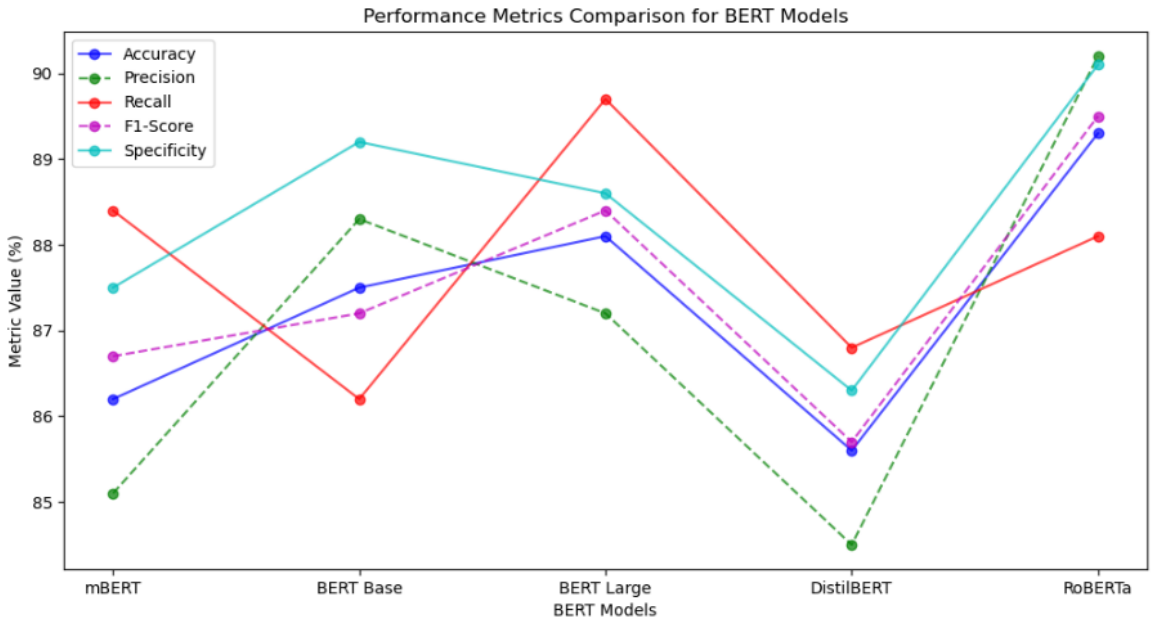

Fig No 5.4.1 Performance Metrics Comparison for BERT Models

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 SOURCE CODE:

Front End: Next js

Landing Page:

```
import styles from "./LandingPage.module.scss";
import {
  Button,
  Input,
  Snippet,
  Table,
  TableBody,
  TableCell,
  TableColumn,
  TableHeader,
  TableRow,
} from "@nextui-org/react";
import cx from "classnames";
```

```
import { Icon } from "@iconify/react";
import { ChangeEvent, useEffect, useState } from "react";
import axios from "axios";


interface DataResponse {
label: string;
  score: number;
}
const initialState: DataResponse[] = [
  {
    label: "",
    score: 0,
  },
];


export const LandingPage = () => {
  const [inputText, setInputText] = useState("");
  const [data, setData] = useState(initialState);
```

```
const handleChange = (event: ChangeEvent<HTMLInputElement>) => {
  setInputText(event.target.value);
};


// api call function
const fetchData = async (value: string) => {
  try {
    const response = await axios.get("http://127.0.0.1:5000/find/" + value);
    return response.data;
  } catch (error) {
    throw error;
  }
};
const handleSearch = () => {
  fetchData(inputText)
    .then((result) => setData(result))
    .catch((error) => console.error("API call error", error));
};
return (
```

```
<div>
  <div className={styles.TitleText}>
    Hate and Offensive Language Detection
  </div>
  <div className="flex items-center mt-5">
    <div className="container mx-auto px-44">
      <Input
        classNames={{
          label: "active:text-black text-white",
          inputWrapper: "bg-purple-100 ",
          clearButton: "text-black",
        }}
        isClearable
        className={cx("")}
        type="email"
        labelPlacement="outside"
        label="Enter text"
        size="lg"
        onChange={handleChange}
```

```
startContent={<Icon icon="mingcute:search-line" height="30" />}
endContent={
            <Button color="primary" onClick={handleSearch}>
              Search
            </Button>
          }
        />
      </div>
    </div>
    <div className="flex items-center mt-5">
      <div className="container mx-auto px-44 text-center">
        {data.map((item, index) => (
          <div key={index}>
            <span className="text-white text-xl mr-2">Score and Value </span>
            <Snippet hideSymbol={true} color="primary" variant="solid">
              {item.label + " " + item.score}
            </Snippet>
          </div>
```

```
        ))}
      </div>
    </div>
    <div className="flex items-center mt-5">
      <div className="container mx-auto px-44 text-center">
        <Table hideHeader aria-label="Example static collection table">

<TableHeader>
          <TableColumn>NAME</TableColumn>
<TableColumn>ROLE</TableColumn>
        </TableHeader>
        <TableBody>
          <TableRow key="1">
            <TableCell>Off_target_other</TableCell>
            <TableCell>Offensive to target_other</TableCell>
          </TableRow>
          <TableRow key="2">
            <TableCell>Not_offensive</TableCell>
            <TableCell>The text is not offensive</TableCell>
```

```
            </TableRow>
            <TableRow key="3">
              <TableCell>Off_target_group</TableCell>
              <TableCell>Offensive to specific group</TableCell>
            </TableRow>
            <TableRow key="4">
              <TableCell>Off_target_ind</TableCell>
              <TableCell>Offensive to specific individual</TableCell>
            </TableRow>
            <TableRow key="5">
              <TableCell>Profanity</TableCell>
              <TableCell>Vulgar offensive words</TableCell>
            </TableRow>
            <TableRow key="6">
              <TableCell>Not_in_intended_language</TableCell>
<TableCell>Not trained this language</TableCell>
</TableRow>
          </TableBody>
        </Table>
```

```
        </div>
      </div>
    </div>
  );
};
```

layout.tsx :

```
import "./globals.css";
import type { Metadata } from "next";
import { Inter } from "next/font/google";
import cx from "classnames";

const inter = Inter({ subsets: ["latin"] });

export const metadata: Metadata = {
  title: "Hate and Offensive Language Detection",
  description: "Generated by create next app",
};
```

```
export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body className={cx(inter.className, "h-screen backgroundBody")}>
        {children}
      </body>
    </html>
  );
}
```

page.tsx :

```
"use client";
import { Button, NextUIProvider } from "@nextui-org/react";
import { LandingPage } from "@/Modules/LandingPage";
```

```
export default function Home() {
  return (
    <NextUIProvider>
      <LandingPage />
    </NextUIProvider>
  );
}
```

Back End : Python

```
import pandas as pd
import numpy as np


# Reading CSV from link
def read_csv_from_link(url):
    path = 'https://drive.google.com/uc?export=download&id=' + url.split('/')[-2]
    df = pd.read_csv(path, delimiter="\t", error_bad_lines=False, header=None)
    return df

# Loading All Data
tamil_train =
read_csv_from_link('https://drive.google.com/file/d/15auwrFAlq52JJ61u7eSfnhT9rZtI5sjk/view?usp=sharing')
tamil_dev = read_csv_from_link('https://drive.google.com/file/d/1Jme-
```

```
Oftjm7OgfMNLKQs1mO_cnsQmznRI/view?usp=sharing')
tamil_test =
read_csv_from_link('https://drive.google.com/file/d/10RHrqXvIKMdnvN_tVJa_FAm41z
aeC8WN/view?usp=sharing')


# Tamil Preprocess
tamil_train = tamil_train.iloc[:, 0:2]
tamil_train = tamil_train.rename(columns={0: "text", 1: "label"})
tamil_dev = tamil_dev.iloc[:, 0:2]
tamil_dev = tamil_dev.rename(columns={0: "text", 1: "label"})

# Stats
tamil_train['label'] = pd.Categorical(tamil_train.label)
tamil_dev['label'] = pd.Categorical(tamil_dev.label)
print(tamil_train['label'].value_counts())
# Change Device - CPU/GPU-0/GPU-1
torch.cuda.set_device(0)
device = 'cuda'
device = device if torch.cuda.is_available() else 'cpu'

import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from tqdm import tqdm
import os
from sklearn.metrics import classification_report, f1_score
from torch.utils.data import Dataset

# Dataset
```

```
class tamil_Offensive_Dataset(Dataset):
    def __init__(self, encodings, labels, bpe=False):
        self.encodings = encodings
        self.labels = labels
        self.is_bpe_tokenized = bpe

    def __getitem__(self, idx):
        if not self.is_bpe_tokenized:
            item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        else:
            item = {
                'input_ids': torch.LongTensor(self.encodings[idx].ids),
                'attention_mask': torch.LongTensor(self.encodings[idx].attention_mask)
            }
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

# list of random seeds to try
r_seeds = [5, 10, 15, 23, 45, 52, 100, 150, 210, 500]
from transformers import BertTokenizer, BertForSequenceClassification

tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')
model = BertForSequenceClassification.from_pretrained('bert-base-multilingual-cased',
num_labels=6)
model_name = 'Mbert_base_cased_tamil'
```

```
from transformers import AdamW

optimizer = AdamW(model.parameters(), lr=1e-5)

label_mapping = {
    'Not_offensive': 0,
    'not-lTami': 1,
    'Offensive_Targeted_Insult_Other': 2,
    'Offensive_Targeted_Insult_Group': 3,
    'Offensive_Untargetede': 4,
    'Offensive_Targeted_Insult_Individual': 5
}

# Collecting Text and Labels
train_batch_sentences = list(tamil_train['text'])
train_batch_labels = [label_mapping[x] for x in tamil_train['label']]
dev_batch_sentences = list(tamil_dev['text'])
dev_batch_labels = [label_mapping[x] for x in tamil_dev['label']]

# Defining Datasets
train_dataset = tamil_Offensive_Dataset(train_encodings, train_labels, bpe=False)
dev_dataset = tamil_Offensive_Dataset(dev_encodings, dev_labels, bpe=False)

device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
model.to(device)
best_val_f1 = 0
count = 0

# Dataloaders
```

```
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
dev_loader = DataLoader(dev_dataset, batch_size=16, shuffle=False)
for i in r_seeds:
    random.seed(i)
    np.random.seed(i)
    torch.manual_seed(i)

    for epoch in range(100):
        train_preds = []
        train_labels = []
        total_train_loss = 0
        model.train()

print("==========================================================")
        print("Epoch {}".format(epoch))
        print("Train")
        for batch in tqdm(train_loader):
            optimizer.zero_grad()
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
            if loss_weighted:
                loss = loss_function(outputs[1], labels)
            else:
                loss = outputs[0]
            loss.backward()
            optimizer.step()
```

```
    for logits in outputs[1].detach().cpu().numpy():
        train_preds.append(np.argmax(logits))
    for logits in labels.cpu().numpy():
        train_labels.append(logits)
    total_train_loss += loss.item() / len(train_loader)

print("Dev")
dev_preds = []
model.eval()
total_val_loss = 0
with torch.set_grad_enabled(False):
    for batch in tqdm(dev_loader):
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        if loss_weighted:
            loss = loss_function(outputs[1], labels)
        else:
            loss = outputs[0]
        total_val_loss += loss.item() / len(dev_loader)

        for logits in outputs[1].cpu().numpy():
            dev_preds.append(np.argmax(logits))

y_true = dev_batch_labels
y_pred = dev_preds
target_names = label_mapping.keys()
train_report = classification_report(train_labels, train_preds,
```

```
target_names=target_names)
        report = classification_report(y_true, y_pred, target_names=target_names)
        val_f1 = f1_score(y_true, y_pred, average='macro')

        # Save Best Model
        if val_f1 > best_val_f1:
            PATH = '../../finetuned_models/' + model_name + '_seed_' + i + '.pth'
            torch.save(model.state_dict(), PATH)
            model.save_pretrained(os.path.join('../../finetuned_berts/', (model_name + '_seed_'
+ i)))
            best_val_f1 = val_f1
            count = 0
        else:
            count += 1


        print(train_report)
        print(report)
        print("Epoch {}, Train Loss = {}, Val Loss = {}, Val F1 = {}, Best Val f1 = {},
stagnant = {}".format(epoch, total_train_loss, total_val_loss, val_f1, best_val_f1, count))
        if count == 5:
            print("No increase for 5 epochs, Stopping ...")
            break


import Library
import pandas as pd
import numpy as np
import fasttext
import fasttext.util
```

```
from keras.layers import Input, Dense, Embedding, Conv2D, MaxPool2D
from keras.layers import Reshape, Flatten, Dropout, Concatenate
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adam
from keras.models import Model
from sklearn.model_selection import train_test_split
from keras import layers
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.preprocessing.text import Tokenizer, one_hot
from keras.preprocessing.sequence import pad_sequences
from sklearn.metrics import classification_report, f1_score
from keras.layers.pooling import MaxPooling2D
```

Data Import and Preprocess

```
# Reading CSV from link
def read_csv_from_link(url):
    path = 'https://drive.google.com/uc?export=download&id=' + url.split('/')[-2]
    df = pd.read_csv(path, delimiter="\t", error_bad_lines=False, header=None)
    return df

# Loading All Data
tamil_train =
read_csv_from_link('https://drive.google.com/file/d/15auwrFAlq52JJ61u7eSfnhT9rZtI5sjk/view?usp=sharing')
tamil_dev = read_csv_from_link('https://drive.google.com/file/d/1Jme-Oftjm7OgfMNLKQs1mO_cnsQmznRI/view?usp=sharing')
```

```
tamil_test =
read_csv_from_link('https://drive.google.com/file/d/10RHrqXvIKMdnvN_tVJa_FAm41zaeC8WN/view?usp=sharing')

# Tamil Preprocess
tamil_train = tamil_train.iloc[:, 0:2]
tamil_train = tamil_train.rename(columns={0: "text", 1: "label"})
tamil_dev = tamil_dev.iloc[:, 0:2]
tamil_dev = tamil_dev.rename(columns={0: "text", 1: "label"})

# Stats
tamil_train['label'] = pd.Categorical(tamil_train.label)
tamil_dev['label'] = pd.Categorical(tamil_dev.label)
print(tamil_train['label'].value_counts())
Training
Fasttext
import emoji
characters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 't', 'u', 'v',
            'w', 'x', 'y', 'z']
def convert_emoticons(text):
    for emot in EMOTICONS:
        text = re.sub(u'(' + emot + ')', "_".join(EMOTICONS[emot].replace(",", "").split()), text)
    return text


def preprocess(text):
    text = emoji.demojize(text, delimiters=("", ""))
    # text = convert_emoticons(text)
    res = text.lower()
```

```
    res = res.replace('_', ' ')
    res = res.replace('.', ' ')
    res = res.replace(',', ' ')
    res = res.strip()
    words = res.split()
    for i, word in enumerate(words):
        if (word[0] in characters):
            if (len(word) < 3): continue
            while words[i][-1] == words[i][-2]:
                if (len(words[i]) < 2): break
                words[i] = words[i][:-1]
                if (len(words[i]) < 2): break
    sen = " ".join(words)
    return sen
train_text = []
for key, value in tamil_train['text'].iteritems():
    train_text.append(preprocess(value))

dev_text = []
for key, value in tamil_dev['text'].iteritems():
    dev_text.append(preprocess(value))
tamil_train['text'] = pd.DataFrame(train_text)
tamil_dev['text'] = pd.DataFrame(dev_text)

corpus = []
for i, sen in enumerate(tamil_train['text']):
    if (tamil_train[label][i] == 'not-Tamil '):
        continue
    if i == 0: continue
```

```
    corpus.append(preprocess(tamil_train['text'][i]))
for i, sen in enumerate(tamil_dev['text']):
    if (tamil_train[label][i] == 'not-Tamil '):
        continue
    if i == 0: continue
    corpus.append(preprocess(tamil_dev['text'][i]))

with open("corpus.txt", "w") as output:
    output.write(str(corpus))

# Train unsupervised skipgram model
unsuper_model = fasttext.train_unsupervised('/home/punyajoy/corpus.txt', "skipgram",
dim=300)
Train and Test set
# function to build vocabulary and inverse vocabulary dictionary
def build_vocab(sentences):
    # Build vocabulary
    word_counts = Counter(itertools.chain(*sentences))
    # Mapping from index to word
    vocabulary_inv = [x[0] for x in word_counts.most_common()]
    vocabulary_inv = list(sorted(vocabulary_inv))
    # Mapping from word to index
    vocabulary = {x: i for i, x in enumerate(vocabulary_inv)}
    return [vocabulary, vocabulary_inv]

# Prepare X_train by replacing text with fasttext embeddings
def build_input_data(sentences, labels):
        x = np.array([np.array([unsuper_model.get_word_vector(word) for word in
sentence]) for sentence in sentences])
```

```
    y = np.array(labels)
    return [x, y]

# padding sentence for uniform input size
def pad_sentences(sentences, padding_word="<PAD/>"):
        sequence_length = 15
    padded_sentences = []
    for i in range(len(sentences)):
        sentence = sentences[i].strip()
        sentence = sentence.split(" ")
        if (len(sentence) > sequence_length):
            sentence = sentence[0:15]
            padded_sentences.append(sentence)
        else:
            num_padding = sequence_length - len(sentence)
            new_sentence = sentence + [padding_word] * num_padding
            padded_sentences.append(new_sentence)
    return padded_sentences
def load_data(train_text, label):
        # Load and preprocess data
    sentences_padded = pad_sentences(train_text)
    # vocabulary, vocabulary_inv = build_vocab(sentences_padded)
    x, y = build_input_data(sentences_padded, label)
    print(type(x))
    return [x, y]

# Loading train set
x_train, y_train = load_data(tamil_train["text"], tamil_train["label"])
# Encoding labels
```

```
x_train = np.asarray(x_train)
coded = dict({'Not_offensive': 0, 'Offensive_Targeted_Insult_Group': 1,
              'Offensive_Targeted_Insult_Individual': 2,
              'Offensive_Untargetede': 3,
              'not-Tamil': 4,
              'Offensive_Targeted_Insult_Other': 5})
for i, j in enumerate(y_train):
    y_train[i] = coded[j]

x_train = x_train.reshape(x_train.shape[0], 15, 300, 1)
from keras.utils import to_categorical

y_train = to_categorical(y_train)

# Loading dev set
x_dev, y_dev = load_data(tamil_dev["text"], tamil_dev["label"])
for i, j in enumerate(y_dev):
    y_dev[i] = coded[j]

x_dev = x_dev.reshape(x_dev.shape[0], 15, 300, 1)
y_dev = to_categorical(y_dev)

# Loading test set

x_test, y_test = load_data(tamil_test[0], tamil_test[1])
x_test = x_test.reshape(x_test.shape[0], 15, 300, 1)
for i, j in enumerate(y_test):
    y_test[i] = coded[j]
```

```
y_test = to_categorical(y_test)


np.save('../sentence_embeddings/cnn_emb_dev_128_tamil.npy', x_dev_dense_cnn)

np.save('../sentence_embeddings/cnn_emb_train_128_tamil.npy', x_train_dense_cnn)

np.save('../sentence_embeddings/cnn_emb_test_128_tamil.npy', x_test_dense_cnn)
    else:
        cnt += 1
    # loop break condition
    if (cnt >= 7):
        print("NO increase for 5 itr, breaking....")

break
```

6.2 OUTPUT:

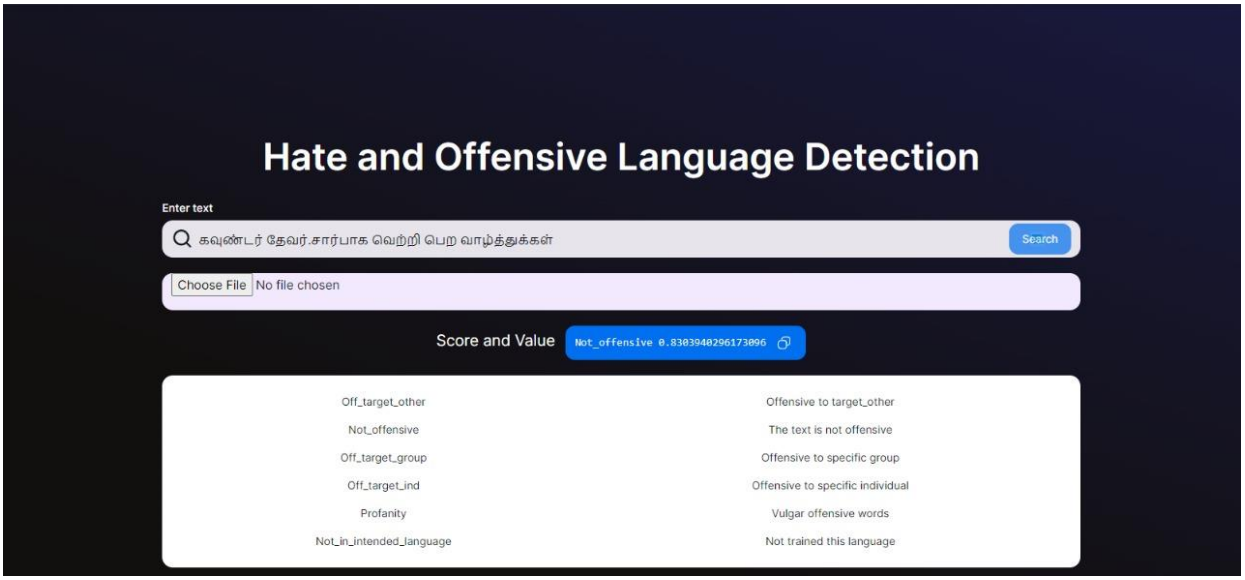

Fig No 6.2.1: Prediction for text using mBERT Model

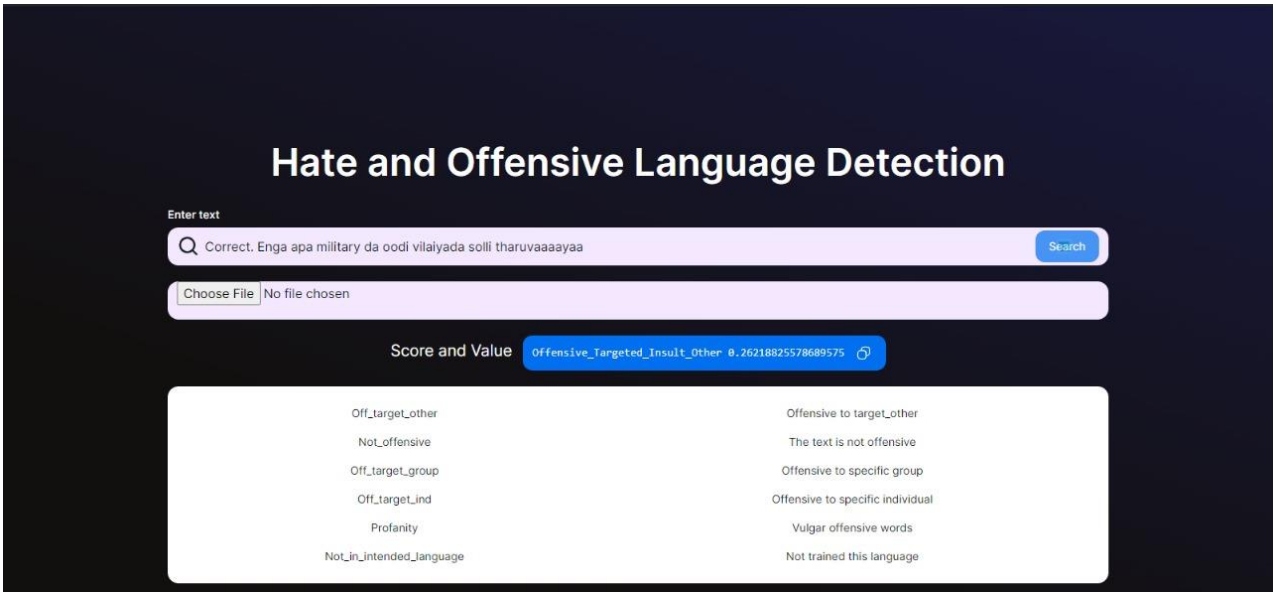

Fig No 6.2.2: Prediction for text using BERT Base Model

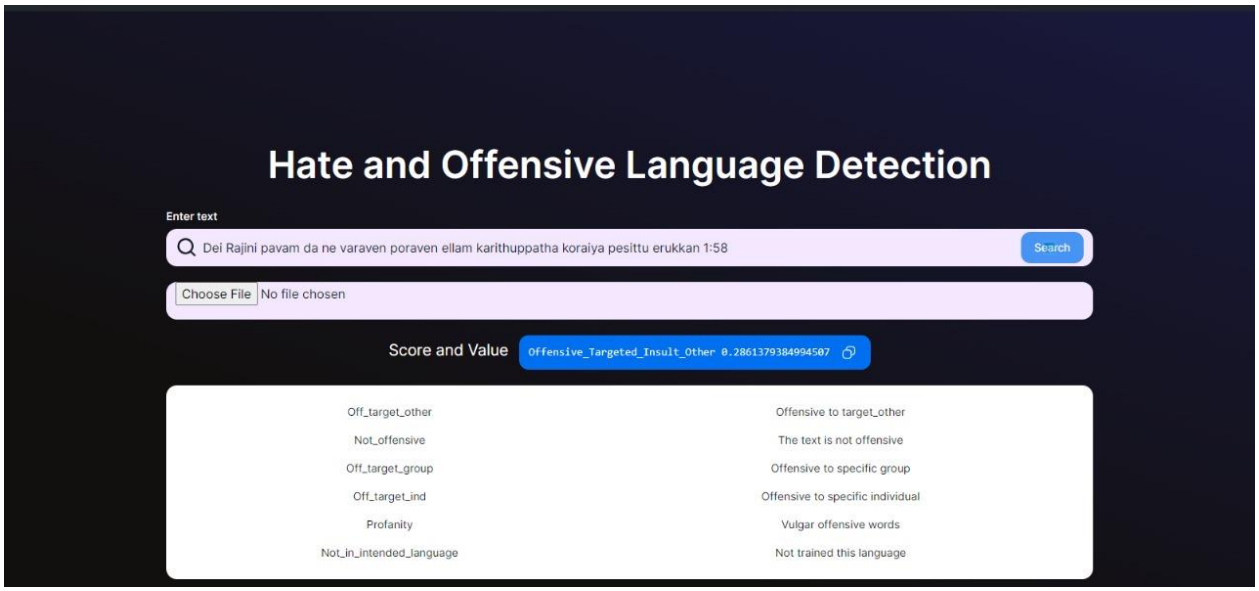

Fig No 6.2.3: Prediction for text using BERT Large Model

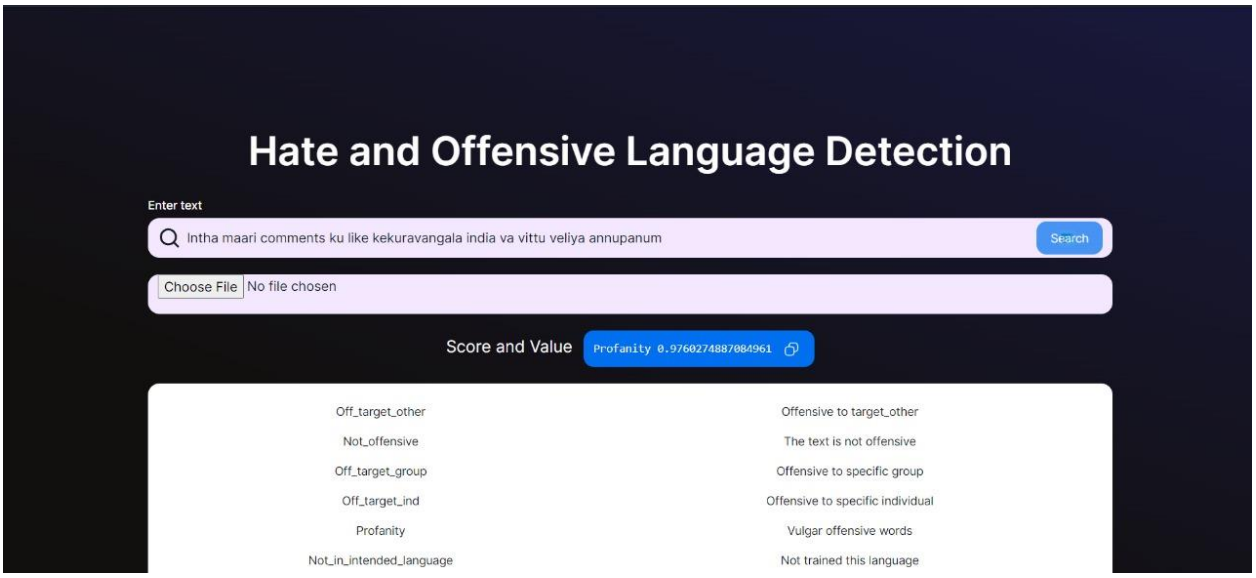

Fig No 6.2.4 : Prediction for text using DistilBERT Model

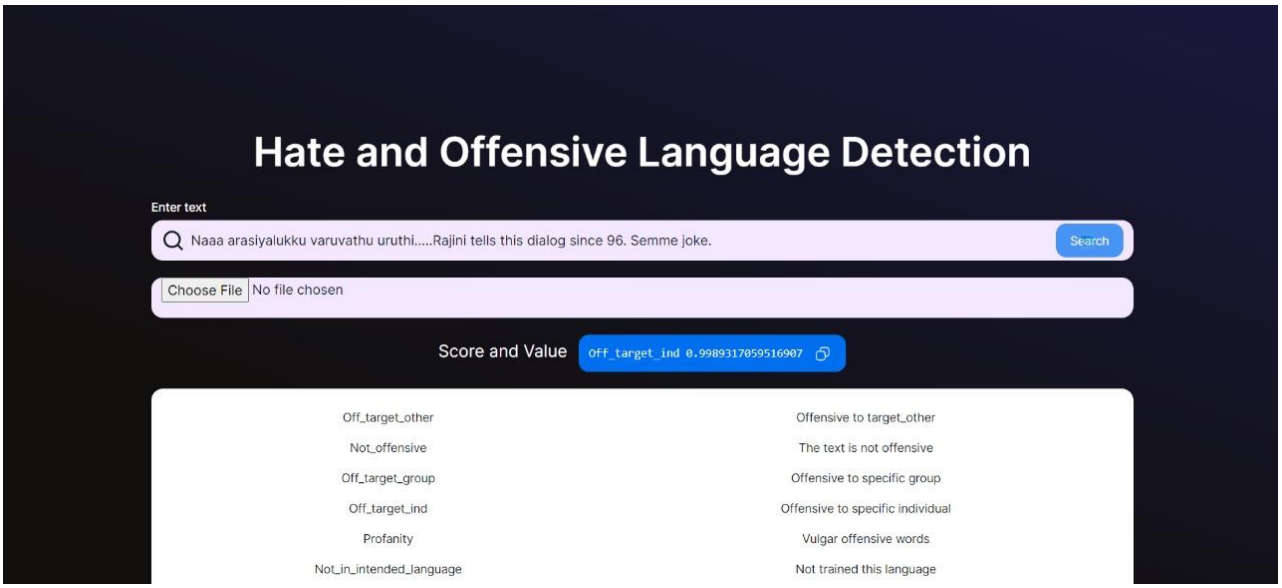

Fig No 6.2.5 : Prediction for text using RoBERTa Model

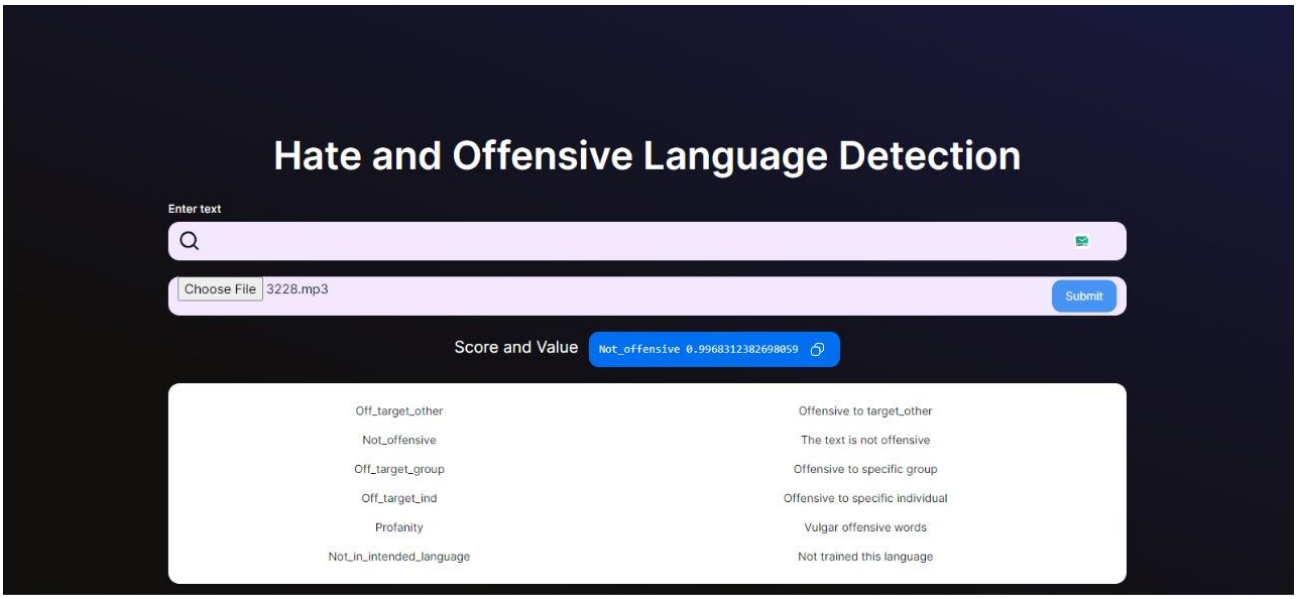

Fig No 6.2.6 : Prediction for Audio using mBERT Model

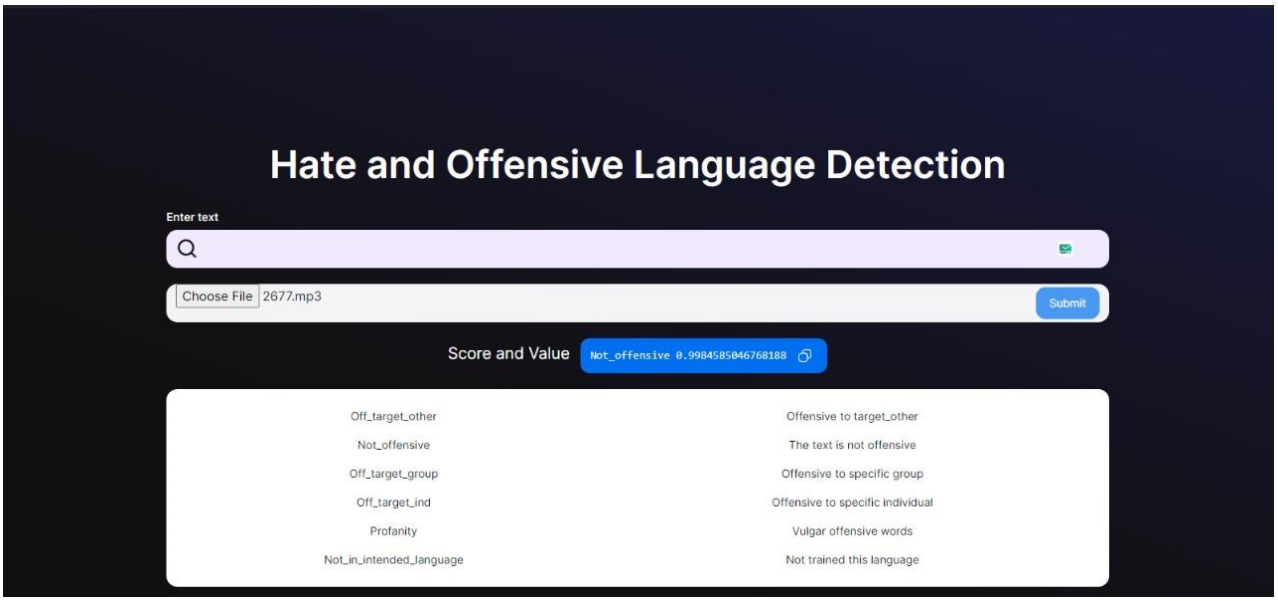

Fig No 6.2.7 : Prediction for Audio using BERT Base Model

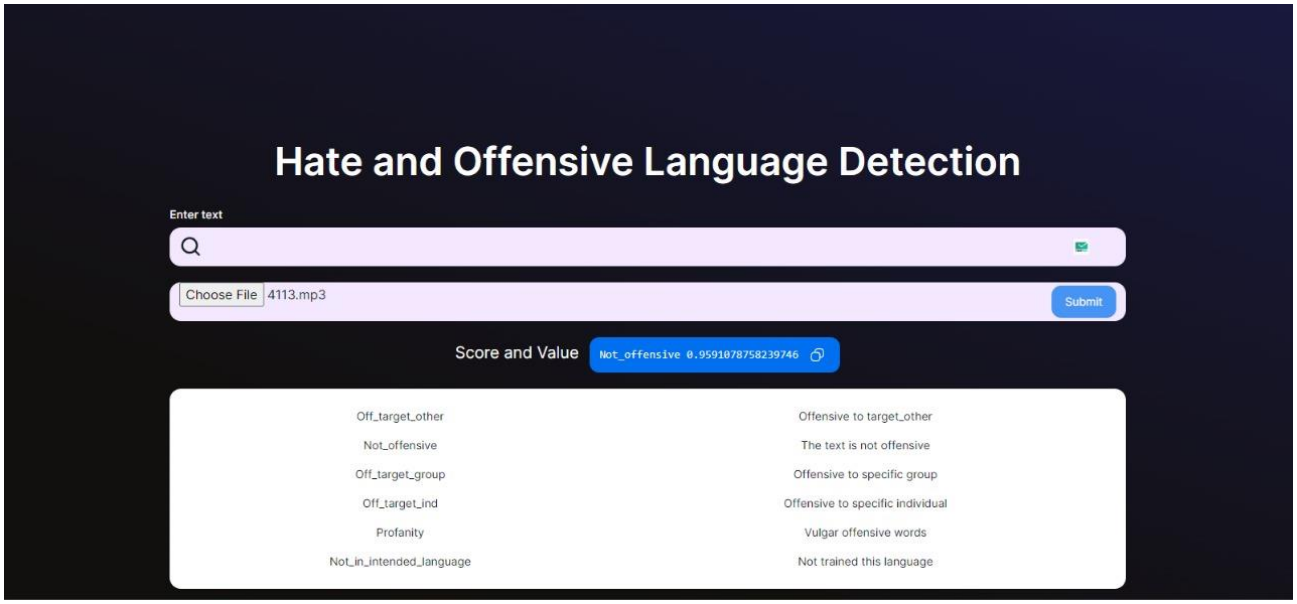

Fig No 6.2.8 : Prediction for Audio using BERT Large Model

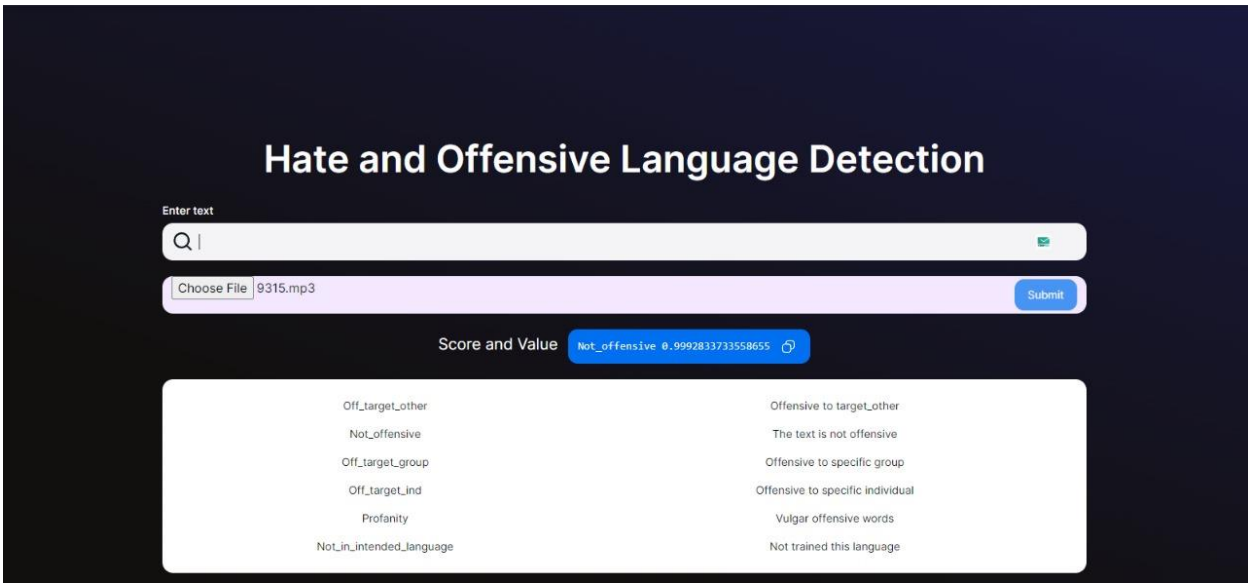

Fig No 6.2.9 : Prediction for Audio using DistilBERT Model

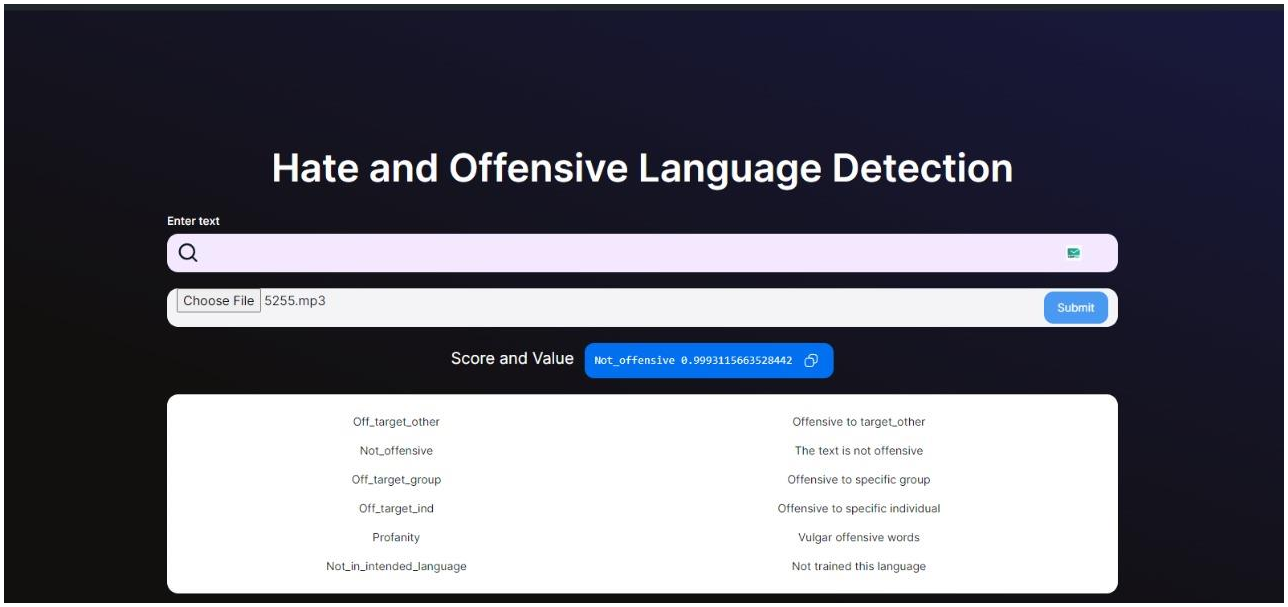

Fig No 6.2.10 : Prediction for Audio using RoBERTa Model

CHAPTER 7

CONCLUSION

This project conducted a comprehensive evaluation of five different BERT models—mBERT, BERT Base, BERT Large, DistilBERT, and RoBERTa—in conjunction with Whisper AI for audio-to-text conversion. Notably, RoBERTa, fine-tuned for Tamil, exhibited superior performance, achieving the highest accuracy and precision in identifying offensive language. BERT Large demonstrated enhanced accuracy and recall despite its computational demands, making it suitable for applications prioritizing accuracy. DistilBERT, renowned for its memory efficiency, offered a balanced solution between performance and resource utilization, ideal for resource-constrained environments. Integrating Whisper AI for audio-to-text conversion provided additional context, enriching the offensive language detection process. Furthermore, mBERT showcased its multilingual effectiveness by effectively identifying offensive Tamil language, highlighting its adaptability. Lastly, BERT Base, though resource-intensive, delivered robust performance, positioning itself as a viable option for well-resourced applications. This comprehensive approach underscores the importance of leveraging both text and audio inputs for more accurate and context-aware offensive language detection on Tamil YouTube.

CHAPTER 8

FUTURE WORKS

In future works, we aim to expand the scope of our research to include other South Indian languages such as Telugu, Kannada, and Malayalam, extending the applicability of offensive language detection across a broader linguistic landscape. By incorporating datasets and fine-tuning techniques tailored to these languages, we aspire to develop models that are proficient in identifying offensive language nuances specific to each linguistic context, thereby enhancing the inclusivity and effectiveness of our detection framework.

Additionally, we envision integrating advanced image recognition and text extraction techniques to enhance the capabilities of our offensive language detection system. Specifically, we plan to explore the integration of computer vision algorithms to extract text from memes and other multimedia content posted on social media platforms. By incorporating this multimodal approach, we aim to leverage both textual and visual cues to identify offensive content more comprehensively. This integration will involve preprocessing steps to extract text from images, followed by feeding the extracted text into our existing model for offensive language detection. Through this approach, we anticipate improving the accuracy and granularity of our detection system, enabling it to effectively flag offensive content across various mediums and formats on social media platforms.

CHAPTER 9

REFERENCES

1. Malak Aljabri, Rachid Zagrouba, Afrah Shaahid, Fatima Alnasser, Asalah Saleh and Dorieh M. Alomari of the shared task on Machine learning-based social media bot detection: a comprehensive literature review. https://link. springer.com /article/10.1007/s13278-022-01020-5. Accessed 05 Jan 2023.

2. Bharathi Raja Chakravarthi, Manoj Balaji Jagadeeshan, Vasanth Palanikumar and Ruba Priyadharshini of the shared task on Offensive language identification in dravidian languages using MPNet and CNN. https://doi.org/10.1016/ j.jjimei.2022. 100151. Accessed 30 Apr 2023.

3. Md Saroar Jahan and Mourad Oussalah of the shared task on A systematic review of hate speech automatic detection using natural language processing. https://doi.org/10.1016/j.neu com.2023.126232. Accessed 14 August 2023.

4. Tanjim Mahmud, Michal Ptaszynski, Juuso Eronen and Fumito Masui of the shared task on Cyberbullying detection for low-resource languages and dialects: Review of the state of the art. https://doi.org/10.1016/j.ipm.2023.103454. Accessed 17 Sept 2023.

5. S.V.Kogilavani, S. Malliga, K.R. Jaiabinaya, M. Malini and M.Manisha Kokila of the shared task on Characterization and mechanical properties of offensive language taxonomy and detection techniques. https://doi.org/10.1016/j.matpr. 2021.04.102. Accessed 18 May 2023.

6. =Ayoub Ali, Alaa Y. Taqa of the shared task on Designing and Implementing Intelligent Textual Plagiarism Detection Models. https://www.researchgate.net /figure/Several-plagiarism classifications_fig1_371695547. Accessed 23 June 2023.

7. Fatimah Alkomah and Xiaogang Ma of the shared task on A Literature Review of Textual Hate Speech Detection Methods and Datasets. https://doi.org/10.3390/info13060273. Accessed 26 May 2022.

8. Cesa Salaam, Franck Dernoncourt, Trung H. Bui, Danda B Rawat of the shared task on Offensive Content Detection Via Synthetic Code-switched Text. https://www.researchgate.net/publication/364647997_Offensive_Content_Detection_Via_Synthetic_Code-switched_Text. Accessed 17 Oct 2022.

9. Bedour Alrashidi, Amani Jamal, Imtiaz Khan and Ali Alkhathlan of the shared task on A review on abusive content automatic detection: approaches, challenges and opportunities. 10.7717/peerj-cs.1142. Accessed 02 Nov 2022.

10. Md Saroar Jahan, Mainul Haque, Nabil Arhab, Mourad Oussalah of the shared task on BERT for Abusive Language Detection in Bengali. https://aclanthology.org/2022.restup-1.2. Accessed 25 Jun 2022.

11. Md Saroar Jahan, and Mourad Oussalah of the shared task on A systematic review of Hate Speech automatic detection using Natural Language Processing. https://doi.org/10.48550/ arXiv.2106.00742. Accessed 22 May 2021.

12. Salim Sazzed of the shared task on Identifying vulgarity in Bengali social media textual content. 10.7717/peerj-cs.665. Accessed 19 Oct 2021.

13. Hossam Elzayady, Mohamed S. Mohamed, Khaled Badran, Gouda I. Salama of the shared task on A hybrid approach based on personality traits for hate speech detection in Arabic social media. 10.11591/ijece.v13i2.pp1979-1988. Accessed 2 Apr 2023.

14. Tharindu Ranasinghe, Isuri Anuradha, Damith Premasiri, Kanishka Silva, Hansi Hettiarachchi, Lasitha Uyangodage and Marcos Zampieri of the shared task on SOLD: Sinhala Offensive Language Dataset. https://doi.org/10.48550/arXiv.2

212. 00851 Accessed 01 Dec 2022.

15. Moin Khan, Khurram Shahzad, Kamran Malik of the shared task on Hate Speech Detection in Roman Urdu. https://doi.org/10.48550/arXiv.2108.02830. Accessed 05 Aug 2021.

16. Charangan Vasantharajan and Uthayasanker Thayasivam of the shared task on Towards Offensive Language Identification for Tamil Code-Mixed YouTube Comments and Posts. https://doi.org/10.48550/arXiv.2108.10939. Accessed 24 Aug 2021.

17. Dorothee Beermann, Laurent Besacier, Sakriani Sakti and Claudia Soria of the shared task on Proceedings of the 1st Joint Workshop on Spoken Language Technologies for Under-resourced languages (SLTU) and Collaboration and Computing for Under-Resourced Languages(CCURL). https://aclanthology.org/ 2020. sltu-1.0. Accessed 17 May 2020.

18. Hossam Elzayady, Mohamed S. Mohamed, Khaled M. Badran, Gouda I. Salama of the shared task on Detecting Arabic textual threats in social media using artificial intelligence: An overview. 10.11591/ijeecs.v25.i3.pp1712-1722. Accessed 3 Mar 2022.

19. Fabio Poletto, Valerio Basile, Manuela Sanguinetti, Cristina Bosco & Viviana Patti of the shared task on Resources and benchmark corpora for hate speech detection: a systematic review. https://link.springer.com/article/10.100 7/s10579-020-09502-8. Accessed 30 Sep 2020.

20. Rahul Pradhan, Ankur Chaturvedi, Aprna Tripathi, Dilip Kumar Sharma of the shared task on A Review on Offensive Language Detection. 10.1007/978-981-15-0694-941. Accessed 23 Jan 2022.

21. Edosomwan, S., Prakasan, S. K., Kouame, D., Watson, J., & Seymour, T.

(2011). The History of Social Media and Its Impact on Business. Management, 16, 79-91 of the shared task on Literature Review on the Factors Influencing the Usage of Social Media among Entrepreneurs in Malaysia. https://www.scirp.org/%28S%28lz5mqp453edsnp55rrgjct55%29%29/reference/referencespapers.aspx?referenceid=3162688. Accessed 27 Jan 2022.

22. Ye, Dai, an Dong, & Wang, 2021 of the shared task on Multi-view ensemble learning method for microblog sentiment classification. https://doi.org/10.10 16/j.eswa.2020.113987. Accessed 15 Mar 2021.

23. Lyu, Chen, Wang, & Luo, 2020 of the shared task on Sense and Sensibility: Characterizing Social Media Users Regarding the Use of Controversial Terms for COVID-19. https://doi.org/10.48550/arXiv.2004.06307. Accessed 24 Apr 2020.

24. Depoux et al of the shared task on The pandemic of social media panic travels faster than the COVID-19 outbreak. https://pubmed.ncbi.nlm.nih.gov/32125413/. Accessed 18 May 2020.

25. Ravikiran & Annamalai of the shared task on DOSA: Dravidian Code-Mixed Offensive Span Identification Dataset. https://aclanthology.org/2021.Dravidianlangtech-1.2. Accessed 20 Apr 2021.

26. Jose, Chakravarthi, Suryawanshi, Sherly, & McCrae, 2020 of the shared task on A Sentiment Analysis Dataset for Code-Mixed Malayalam-English. https://aclanthology.org/2020.sltu-1.25. Accessed 23 May 2020.

27. Mandl, Modha, Kumar M, Chakravarthi, 2020, Zampieri, Nakov, Rosenthal, Atanasova, Karadzhov, Mubarak, Derczynski, Pitenis, Çöltekin, 2020 of the shared task on SemEval-2020 Task 12: Multilingual Offensive Language Identification in Social Media (OffensEval 2020). http://dx.doi.org/10.18653/v1/2020.semeval-1.188. Accessed 13 Dec 2020.

28. Thavareesan, Mahesan, 2019, Thavareesan, Mahesan, 2020, Thavareesan, Mahesan, 2020 of the shared task on Sentiment Analysis in Tamil Texts: A Study on Machine Learning Techniques and Feature Representation. 10.1109/ICIIS47346.2019.9063341. Accessed 28 Dec 2019.

29. Chakravarthi et al., 2021c of the shared task on Findings of the Shared Task on Hope Speech Detection for Equality, Diversity, and Inclusion. https://aclanthology.org/2021.ltedi-1.8. Accessed 27 Apr 2021.

30. Krishnamurti of the shared task on The Dravidian Languages. www.cambridge.org/9780521771115. Accessed 21 Dec 2003.

31. Sapetal of the shared task on The Risk of Racial Bias in Hate Speech Detection. http://dx.doi.org/10.18653/v1/P19-1163. Accessed 17 Jul 2019.

32. Laub of the shared task on Hate Speech on Social Media: Global Comparisons. https://www.cfr.org/backgrounder/hate-speech-social-media-global-comparisons. Accessed 7 Jun 2019.

33. Alec Radford Jong Wook Kim Tao Xu Greg Brockman Christine McLeavey Ilya Sutskever of the shared task on Robust Speech Recognition via Large-Scale Weak Supervision. https://cdn.openai.com/papers/whisper.pdf. Accessed 29 November 2019.

34. Rishabh Jain, Andrei Barcovschi, Mariam Yiwere, Peter Corcoran, Horia Cucu of the shared task on Adaptation of Whisper models to child speech recognition. https://doi.org/10.48550/arXiv.2307.13008. Accessed on 24 Jul 2023.

35. Ashwin Rao of the shared task on Transcribing Educational Videos Using Whisper. https://doi.org/10.48550/arXiv.2307.03200. Accessed on 4 Jul 2023.