IDENTIFICATION OF OFFENSIVE LANGUAGE IN SOCIAL MEDIA POSTS AND COMMENTS

PROJECT PHASE 1 REPORT

Submitted by

JISHNU .B - 20P122

SUWINKUMAR .T - 20P156

TANUSH .K - 20P157

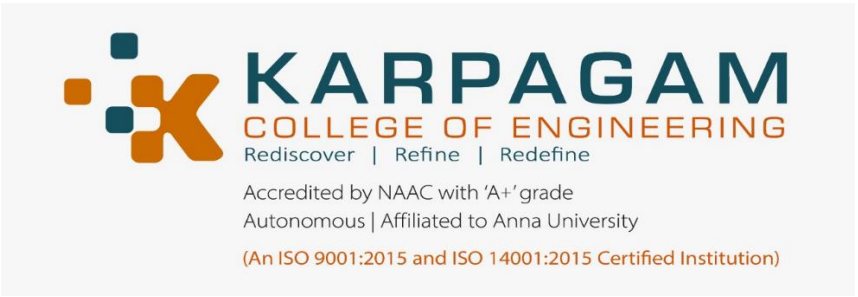
UDHAYARAJAN .M - 20P158

in partial fulfillment for the award of the degreeof

BACHELOR OF ENGINEERING

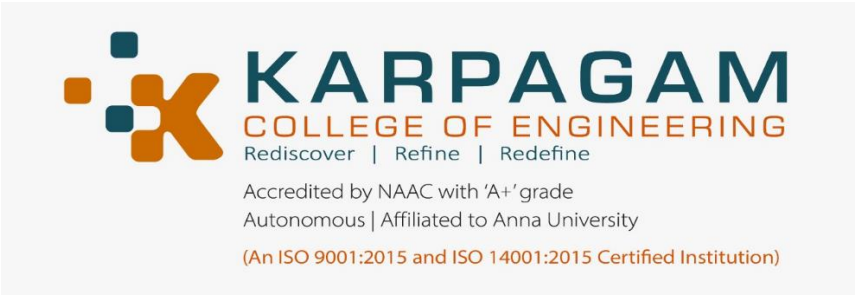
in

COMPUTER SCIENCE AND ENGINEERING



ANNA UNIVERSITY : CHENNAI 600 025

DECEMBER 2023



BONAFIDE CERTIFICATE

Certified that this project report **"IDENTIFICATION OF OFFENSIVE LANGUAGE IN SOCIAL MEDIA POSTS AND COMMENTS"** is the bonafide work of **"JISHNU.B (20P122), SUWINKUMAR.T (20P156), TANUSH.K (20P157), UDHAYARAJAN.M (20P158)"** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported here in does not form part of any other dissertation on the basis of which degree or award was conferred on an earlier occasion on this or any other candidate

SIGNATURE

Dr. S. ARUL ANTRAN VIJAY M. Tech., Ph.D.
FACULTY GUIDE

SIGNATURE

Dr. T. RAVICHANDRAN M.E., Ph.D.
HEAD OF THE DEPARTMENT

Certified that the candidate was examined in the viva-voce examination held on_

_____

……...… ……....

(Internal Examiner) (External Examiner)

ACKNOWLEDGEMENT

We express our sincere thanks to Karpagam educational and charitable trust for providing necessary facilities to bring out the project successfully. We felt greatness to record our thanks to the chairman **Dr. R. VASANTHAKUMAR, B.E., (Hons), D.Sc.** for all his support and ray of strengthening hope extended.

It is the moment of immense pride for us to reveal our profound thanks to our respected Principal, **Dr. V. KUMAR CHINNAIYAN, M.E., Ph.D.** who happens to be striving force in all our endeavors.

We express our sincere thanks to our **Dr. T. RAVICHANDRAN ME., Ph.D.** Head of the Department of Computer Science and Engineering for providing an opportunity to work on this project. His valuable suggestions helped us a lot to do this project.

A word of thanks would not be sufficient for the work of our project guide **Dr. S. ARUL ANTRAN VIJAY M. Tech., Ph.D.** Department of Computer Science and Engineering whose efforts and inspiration lead us through every trying circumstance.

We would also like to recollect the courage and enthusiasm that was inculcated in us by our project co - ordinator, **Dr. R. KALA ME., Ph.D.** Department of Computer Science and Engineering for valuable guidance and support through the tenure of our project.

We deeply express our gratitude to all the members of the faculty of the Department of Computer Science and Engineering for the encouragement, whichwe received throughout the semester.

TABLE OF CONTENTS

ABSTRACT

The rapid proliferation of online content has given rise to an urgent need for automated systems capable of identifying offensive language in comments and posts across various digital platforms. This approach presents a comprehensive approach to address this challenge. We propose a multi-faceted model that combines natural language processing techniques, machine learning algorithms, and deep neural networks to accurately detect offensive language. Our approach begins with data pre-processing, including tokenization, stemming, and lemmatization, to enhance the model's understanding of text. We employ a wide range of linguistic features, such as word embeddings and syntactic analysis, to capture the nuances of offensive language usage. Additionally, we curate a large and diverse dataset of offensive and non-offensive text to train and fine-tune our model.

To tackle the dynamic nature of offensive language, we incorporate continuous learning techniques that adapt the model over time to evolving language trends and user behaviors. This approach ensures the model's long-term effectiveness in identifying offensive content. Evaluation of our approach against benchmark datasets demonstrates its superior performance in terms of precision, recall, and F1-score. Furthermore, we provide insights into the interpretability of the model's decisions, addressing concerns related to transparency and accountability. Overall, our comprehensive approach offers a robust solution for identifying offensive language in comments and posts, promoting safer and more inclusive online communities.

LIST OF FIGURES

LIST OF TABLES

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

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 [21] (Edosomwan, Prakasan, Kouame, Watson, & Seymour, 2011). 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 [22] (Ye, Dai, an Dong, & Wang, 2021).

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 [23] (Lyu, Chen, Wang, & Luo, 2020). 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 [24] (Depoux et al., 2020). 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 [25] (Ravikiran & Annamalai, 2021). 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 [26] (Jose, Chakravarthi, Suryawanshi, Sherly, & McCrae, 2020).

In recent years, there have been substantial improvements in research on hate speech identification and offensive language detection [27] (Mandl, Modha, Kumar M, Chakravarthi, 2020, Zampieri, Nakov, Rosenthal, Atanasova, Karadzhov, Mubarak, Derczynski, Pitenis, Çöltekin, 2020) 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 [28] (Thavareesan, Mahesan, 2019, Thavareesan, Mahesan, 2020, Thavareesan, Mahesan, 2020). Recently, [29] (Chakravarthi et al., 2021c) 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 [30] (Krishnamurti, 2003). 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. Section 2 discusses the literature on offensive language identification, while Section 3 introduces the dataset used for the task at hand. Section 4 focuses on the several models and approaches for identifying offensive languages in Dravidian languages. Section 4.9 discusses the details of the implementation of detection methods. Section 5 comprises a detailed analysis on the behaviour and results of the models, which are also compared with other approaches. Finally, Section 6 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 [31] [Sapetal. 2019]. 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 [32] [Laub 2019]. 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?
- Q7: What are the challenges in scaling up real-time detection to handle large volumes of user-generated content?

  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: section 2 will include a brief theoretical definition of HS. Section 4 examines the previously identified review papers of HS detection. Section 5 details the systematic literature review document collection methodology. Section 6 presents the results of this literature review, including the state of deep learning technology. Section 7 emphasizes on the available resources (datasets and open-source projects). After that, in section 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, various performance measures like precision and recall.

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: ***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.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING APPROACHES:

1. **Keyword-Based Filtering**: This is one of the simplest methods, where a predefined list of offensive words and phrases is used to filter out or flag comments or posts containing these keywords. This method is often combined with more sophisticated techniques for better accuracy.

2. **Machine Learning Models**: Many platforms employ machine learning models, such as natural language processing (NLP) models, to automatically detect offensive language. These models are trained on large datasets of offensive and non-offensive text to learn patterns and identify new forms of offensive language.

3. **Sentiment Analysis**: Sentiment analysis techniques are used to determine the emotional tone of a comment or post. Offensive language is often associated with negative sentiment, so sentiment analysis can be used as a component of a system for identifying offensive content.

4. **Contextual Analysis**: Understanding the context in which words or phrases are used is crucial for accurate detection. Words that might be offensive in one context could be harmless in another. Systems often consider the surrounding words and the overall context to make more accurate judgments.

5. **User Reporting**: Most platforms rely on users to report offensive content. When a user flags a comment or post as offensive, it can trigger a review process. User reporting is a valuable tool in combination with automated methods.

6. **Human Moderation**: Some platforms employ human moderators to review reported content. Human moderators can provide nuanced judgment, especially when dealing with complex or context-dependent cases.

7. **Whitelists and Blacklists**: Platforms maintain lists of known offensive and acceptable terms to aid in content filtering. Content containing blacklisted terms can be automatically flagged, while whitelisted terms are allowed.

8. **Customization**: Many platforms allow users to customize their content filters to a certain extent. Users can define which types of content they want to filter or allow, giving them control over their online experience.

9. **Legal and Ethical Considerations**: Systems need to take into account legal and ethical considerations, such as freedom of speech and avoiding censorship of legitimate content. Striking a balance between removing offensive content and respecting user rights is an ongoing challenge.

3.2 Limitations of Existing Approaches:

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

1. **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.

2. **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.

3. **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.

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

5. **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.

6. **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.

7. **Privacy Concerns:** Offensive content detection raises concerns about user privacy. To detect such content, models need access to users' posts and comments, which can be a privacy risk. Striking a balance between user privacy and content monitoring is an ongoing challenge.

8. **Scalability:** The sheer volume of social media content makes it challenging to scale up the process of identifying and moderating offensive content. Manual review is often required, which can be time-consuming and costly.

3.2 PROPOSED SYSTEM:

The proposed model is a **BERT-CNN fusion classifier** that combines the embeddings from different BERT and CNN models to detect offensive language in code-mixed Dravidian languages. The BERT models are fine-tuned on multilingual or language-specific transformer models, and the CNN models are trained on FastText word vectors. The fusion classifier is a feed-forward neural network with batch normalization and dropout. The model is trained and evaluated on a dataset of Youtube comments in Tamil. The model achieves the best results in Tamil.

PROPOSED SOLUTION ARCHITECTURE:

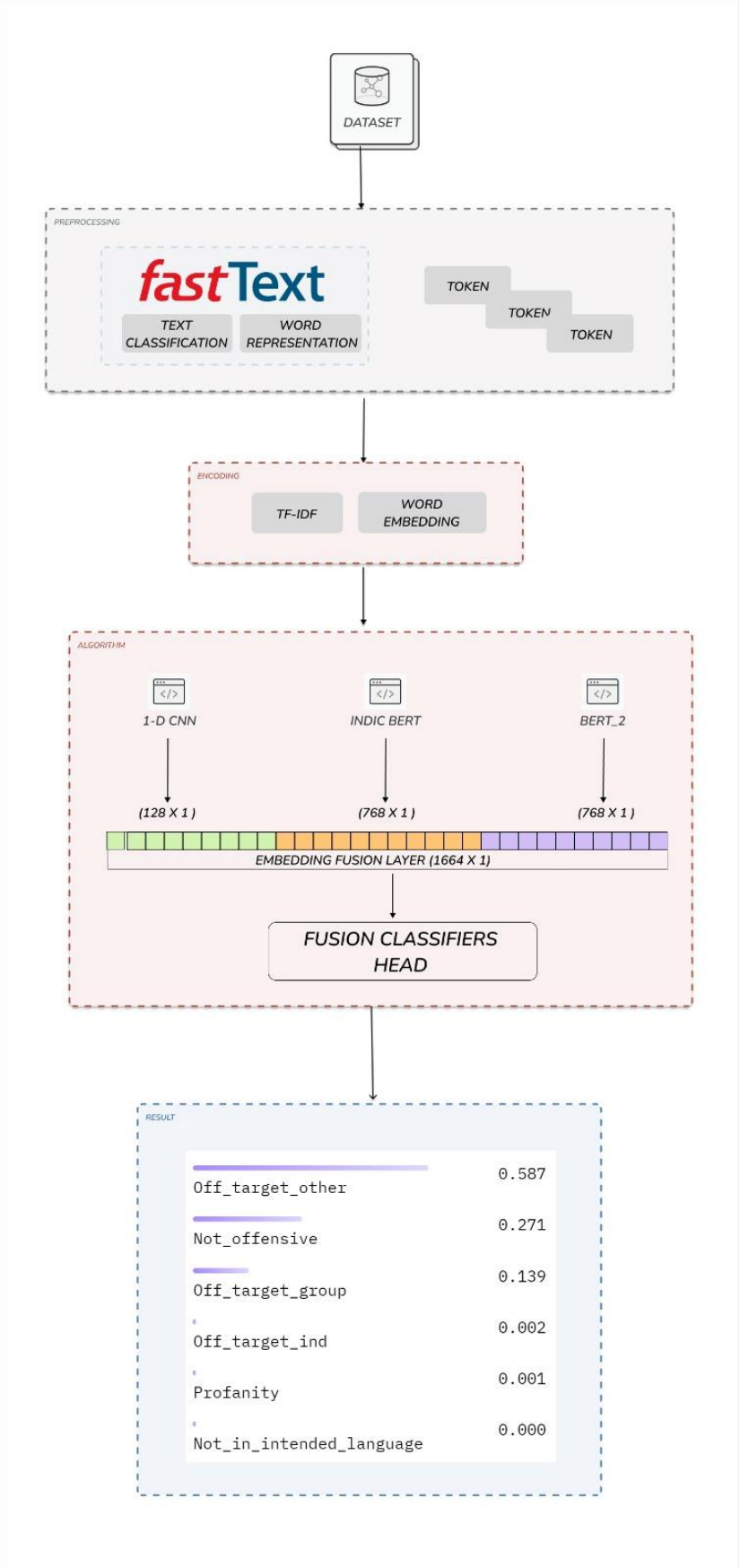


Fig 3.2: System Architecture

Advantages of Our Approach:

1. **Multilingual and Code-Mixed Competence**: Our approach excels in handling the multilingual and code-mixed nature of social media content, ensuring effectiveness in real-world scenarios where languages blend seamlessly.

2. **Customization through Fine-Tuning**: We fine-tune state-of-the-art transformer models on the specific dataset, enhancing the model's adaptability to the linguistic nuances of Dravidian languages, resulting in improved performance.

3. **Ensemble Fusion Model**: By combining BERT and CNN models, we leverage the strengths of each, capturing both semantic and neighborhood information, leading to more robust and accurate offensive language detection.

4. **Class Imbalance Handling**: Our approach acknowledges and addresses class imbalance problems, reflecting real-world scenarios, thus improving the model's effectiveness across different classes.

5. **Domain-Specific Pretraining**: We pretrain our model on the target dataset using Masked Language Modeling, enabling it to understand the semantics and context of the code-mixed corpus, enhancing its context-awareness.

6. **Transparent Architecture**: Our methodology is transparently defined, making it easily comprehensible and reproducible for research and further development.

7. **Scalability**: The flexibility in the number of BERT models in the fusion classifier ensures scalability, accommodating different resource constraints and needs.

8. **Utilization of State-of-the-Art Models**: We leverage cutting-edge models and techniques, benefiting from the latest advancements in natural language processing and machine learning, potentially outperforming older models.

These advantages collectively position our approach as a robust and adaptable solution for offensive language detection in Dravidian languages, ensuring that it is well-equipped to handle the complexities of real-world social media content.

CHAPTER 4

REQUIREMENTS

4.1 HARDWARE SPECIFICATIONS

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

RAM - 8 GB DDR4@3200MHz

Storage - 128 GB M.2 NvMe SSD

4.2 SOFTWARE SPECIFICATIONS

OS - Windows 8, 10 or 11

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
- cffi-1.16.0
- charset-normalizer-3.3.0
- colorama-0.4.6
- comm-0.1.4
- debugpy-1.8.0
- decorator-5.1.1
- 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-1.59.0

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

CHAPTER 5

MODULE DESCRIPTION

5.1. DATASET PREPARATION AND PREPROCESSING

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

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
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 5.1.2: Tamil Comments Sample Data

DATA PREPROCESSING:

The process begins with the definition of a custom Python function named read_csv_from_link(url). This function is designed to read CSV files from Google Drive links. It takes a URL as its input, extracts the file ID from the URL, and constructs a new URL that triggers the download of the CSV data.

The pandas library is then utilized to read this downloaded data into a DataFrame, making it readily available for manipulation and analysis. After this, the code proceeds to load specific datasets from their respective Google Drive URLs. In this case, three datasets, namely tamil_train, tamil_dev, and tamil_test, are fetched using the read_csv_from_link() function.

Subsequently, the preprocessing phase involves refining the structure of the data. Specifically, the tamil_train and tamil_dev datasets are subjected to a series of transformations. These transformations entail selecting only the first two columns from each dataset, which are conventionally used for text and labels. Furthermore, the column names are adjusted to "text" and "label" for consistency and clarity in data analysis.

One notable step in this preprocessing stage is the conversion of label data within the tamil_train and tamil_dev datasets to categorical data types. This categorical encoding is crucial, especially in classification tasks, as it simplifies the handling of label-based data. Lastly, the code provides insight into the distribution of labels in the tamil_train dataset by printing the value counts. This serves as a useful summary, offering a quick overview of how the different categories are distributed within the training data.

Overall, the data import and preprocessing steps presented in this code snippet are a fundamental starting point in the workflow of data analysis and machine learning, ensuring that the data is structured and prepared for further exploration or model training. This is a common preprocessing step in Natural Language Processing tasks, where preparing your data for further analysis or model training.

Pseudo Code:

function read_csv_from_link(url):

  # Extract the file id from the url

file_id = extract the part after the last '/' from url

  # Construct the download path

  path = concatenate 'https://drive.google.com/uc?export=download&id=' and file_id

  # Read the CSV file from the path

  df = read CSV file from path with delimiter "\t" and no header

  # Return the dataframe

  return df

Load all data

tamil_train = call read_csv_from_link with url 'https://drive.google.com/file/d/15auwrFAlq52JJ61u7eSfnhT9rZtI5sjk/view?usp=sharing'

tamil_dev = call read_csv_from_link with url 'https://drive.google.com/file/d/1Jme-Oftjm7OgfMNLKQs1mO_cnsQmznRI/view?usp=sharing'

tamil_test = call read_csv_from_link with url 'https://drive.google.com/file/d/10RHrqXvIKMdnvN_tVJa_FAm41zaeC8WN/view?usp=sharing'

Preprocess Tamil data

Keep only the first two columns of tamil_train and tamil_dev

tamil_train = keep only the first two columns of tamil_train

tamil_dev = keep only the first two columns of tamil_dev

Rename the columns of tamil_train and tamil_dev

tamil_train = rename the columns of tamil_train to "text" and "label"

tamil_dev = rename the columns of tamil_dev to "text" and "label"

Convert the 'label' column of tamil_train and tamil_dev to categorical

tamil_train['label'] = convert tamil_train['label'] to categorical

tamil_dev['label'] = convert tamil_dev['label'] to categorical

Print the count of each label in tamil_train

print the count of each label in tamil_train['label']

5.2. MODEL BUILDING

The process entails feeding the algorithm with training data. An algorithm will process data and output a model that is able to find a target value in new data. The purpose of model training is to develop a model. Supervised learning allows for processing data with target attributes or labeled data. The goal of model training is to predict the most accurate abusive content.

In our project, we employed three types of models:

ML Model: We used random forests and logistic regression, trained on TF-IDF vectors. The best results were achieved with the ExtraTrees Classifier, yielding weighted F1-scores of 0.70, 0.63, and 0.95.

Transformer Model: We fine-tuned various multilingual BERT models, including XLMRo BERTa, multilingual-BERT, Indic BERT, and MuRIL. Additionally, we

pre-trained XLM-Roberta-Base on the target dataset using Masked Language Modeling for 20 epochs. All models were fine-tuned with both unweighted and weighted cross-entropy loss functions.

Fusion Model: To capture neighborhood information effectively, we introduced a BERT-CNN fusion classifier. This model utilizes embeddings from different BERT and CNN models, with the BERT models initialized with fine-tuned weights. We used FastText word vectors for the CNN part and implemented a feed-forward neural network with four layers for the fusion classifier.

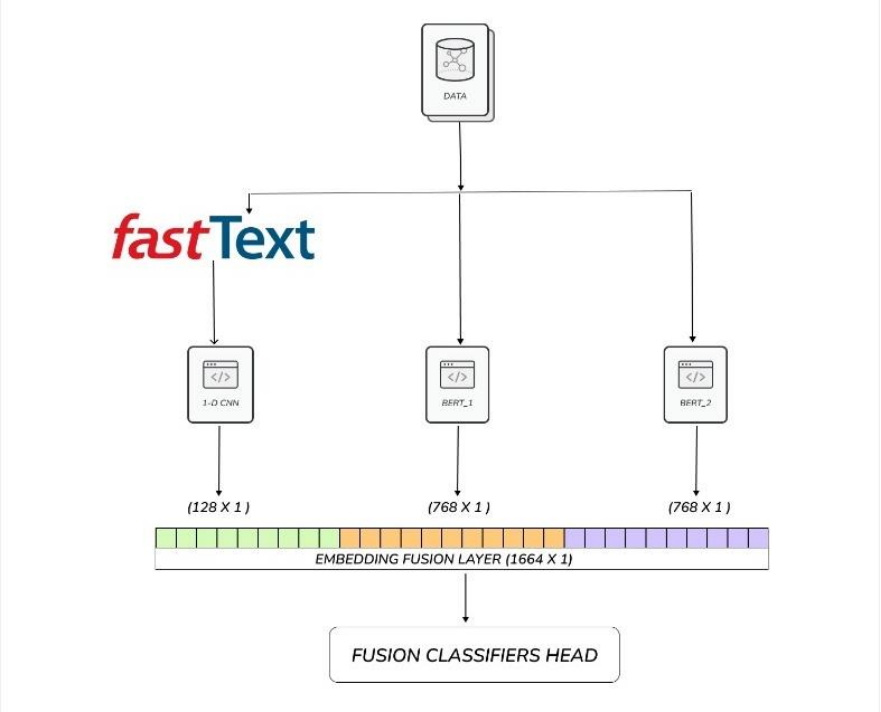


Figure 5.3.1: Fusion Model Architecture

Pseudo Code:

# Import necessary libraries
import MaxPooling2D from keras.layers.pooling
import other necessary libraries

# Define the input shape
inputs = define input shape (15, 300, 1)

Define the CNN layers

conv_0 = define a 2D convolutional layer with 64 filters, kernel size of 5, 'relu' activation, and 'valid' padding
conv_1 = define a 2D convolutional layer with 32 filters, kernel size of 3, 'relu' activation, and 'valid' padding
conv_2 = define a 2D convolutional layer with 32 filters, kernel size of 3, 'relu'

activation, and 'valid' padding
drop = apply dropout with rate 0.6
conv_3 = define a 2D convolutional layer with 16 filters, kernel size of 3, 'relu' activation
pool0 = apply 2D max pooling with pool size (2, 2) and 'valid' padding
conv_4 = define a 2D convolutional layer with 16 filters, kernel size of 3, 'relu' activation
pool1 = apply 2D max pooling with pool size (2, 2) and 'valid' padding

# Flatten the tensor and define the dense layers
flatten = flatten the tensor
hidden1 = define a dense layer with 128 units and 'relu' activation
output = define a dense layer with 6 units and 'softmax' activation

# Define the models
model1 = define a model with inputs and output
model2 = define a model with inputs and hidden1

Compile the models
compile model1 with 'adam' optimizer, 'categorical_crossentropy' loss, and 'accuracy' metrics
compile model2 with 'adam' optimizer, 'categorical_crossentropy' loss, and 'accuracy' metrics
# Train the model
for each epoch in range(50):
  fit model1 on x_train and y_train, validate on x_dev and y_dev
  predict on x_train and x_dev using model1
  calculate and print the f1 score for train and validation predictions
  if validation f1 score is greater than best f1 score:
    update best f1 score

predict on x_train, x_dev, and x_test using model2
save the predictions to numpy files
else:
increment counter
if counter is greater than or equal to 7:
break the loop

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

Pseudo Code :

```
# Define a function to build a vocabulary from a list of sentences
function build_vocab(sentences):
  # Create an empty dictionary to store word counts
  word_counts = {}
  # Loop through each sentence in the list
  for each sentence in sentences:
    # Split the sentence into words
    words = split sentence by spaces
    # Loop through each word in the sentence
    for each word in words:
      # If the word is already in the dictionary, increment its count by one
      if word is in word_counts:
        word_counts[word] = word_counts[word] + 1
      # Otherwise, add the word to the dictionary with a count of one
      else:
        word_counts[word] = 1
  # Sort the dictionary by word counts in descending order
```

```
  word_counts = sort word_counts by values in reverse order
  # Create an empty list to store the vocabulary
  vocabulary = []
  # Loop through each word and count pair in the dictionary
  for each word, count in word_counts:
    # Append the word to the vocabulary list
    append word to vocabulary
  # Return the vocabulary list
  return vocabulary

# Define a function to map sentences and labels to vectors based on a vocabulary
function build_input_data(sentences, labels):
  # Create an empty list to store the input vectors
  x = []
  # Create an empty list to store the label vectors
  y = []
  # Loop through each sentence and label pair in the list
  for each sentence, label in zip(sentences, labels):
    # Create an empty list to store the word vectors
    word_vectors = []
    # Split the sentence into words
    words = split sentence by spaces
    # Loop through each word in the sentence
    for each word in words:
      # Get the fasttext vector for the word
      word_vector = get_word_vector(word)
      # Append the word vector to the word vectors list
      append word_vector to word_vectors
    # Convert the word vectors list to a numpy array
word_vectors = numpy array of word_vectors
```

```
    # Append the word vectors array to the input vectors list
    append word_vectors to x
    # Append the label to the label vectors list
    append label to y
  # Convert the input vectors list to a numpy array
  x = numpy array of x
  # Convert the label vectors list to a numpy array
  y = numpy array of y
  # Return the input vectors and label vectors
  return x, y

# Define a function to pad sentences to the same length
function pad_sentences(sentences, padding_word):
  # Set the sequence length to 15
  sequence_length = 15
  # Create an empty list to store the padded sentences
  padded_sentences = []
  # Loop through each sentence in the list
  for each sentence in sentences:
    # Strip the sentence of any leading or trailing whitespace
    sentence = strip sentence of whitespace
    # Split the sentence into words
    words = split sentence by spaces
    # If the number of words is greater than the sequence length
    if length of words is greater than sequence_length:
      # Truncate the words to the first 15 elements
      words = first 15 elements of words
      # Append the words to the padded sentences list
      append words to padded_sentences
    # Otherwise
```

```
    else:
      # Calculate the number of padding words needed
      num_padding = sequence_length - length of words
      # Create a new list of words with the padding word appended to the end
      new_words = words + [padding_word] * num_padding
      # Append the new words to the padded sentences list
      append new_words to padded_sentences
  # Return the padded sentences list
  return padded_sentences

# Define a function to load and preprocess data for the dataset
function load_data(train_text, label):
  # Pad the train text sentences to the same length
  sentences_padded = pad_sentences(train_text, "<PAD/>")
  # Build the input vectors and label vectors for the train text and label
  x, y = build_input_data(sentences_padded, label)
  # Print the type of x
  print type of x
  # Return the input vectors and label vectors
  return x, y

# Load the train set
x_train, y_train = load_data(tamil_train["text"], tamil_train["label"])
# Encode the labels using a dictionary
coded = {"Not_offensive": 0, "Offensive_Targeted_Insult_Group": 1,
"Offensive_Targeted_Insult_Individual": 2, "Offensive_Untargetede": 3, "not-
Tamil": 4, "Offensive_Targeted_Insult_Other": 5}
# Loop through each label in the train set
for each i, j in enumerate(y_train):
  # Replace the label with the corresponding code
```

```
  y_train[i] = coded[j]

# Reshape the train input vectors to have four dimensions
x_train = reshape x_train to (length of x_train, 15, 300, 1)
# Convert the train label vectors to categorical format
y_train = to_categorical(y_train)

# Load the dev set
x_dev, y_dev = load_data(tamil_dev["text"], tamil_dev["label"])
# Loop through each label in the dev set
for each i, j in enumerate(y_dev):
  # Replace the label with the corresponding code
  y_dev[i] = coded[j]

# Reshape the dev input vectors to have four dimensions
x_dev = reshape x_dev to (length of x_dev, 15, 300, 1)
# Convert the dev label vectors to categorical format
y_dev = to_categorical(y_dev)

# Load the test set
x_test, y_test = load_data(tamil_test[0], tamil_test[1])
# Reshape the test input vectors to have four dimensions
x_test = reshape x_test to (length of x_test, 15, 300, 1)
# Loop through each label in the test set
for each i, j in enumerate(y_test):
  # Replace the label with the corresponding code
  y_test[i] = coded[j]
# Convert the test label vectors to categorical format
y_test = to_categorical(y_test)
```

5.4. USER INTERACTION

This is the module where the user can able to interact with our model. The values for the prediction are collected from the user and the output will be displayed.

USER PAGE:

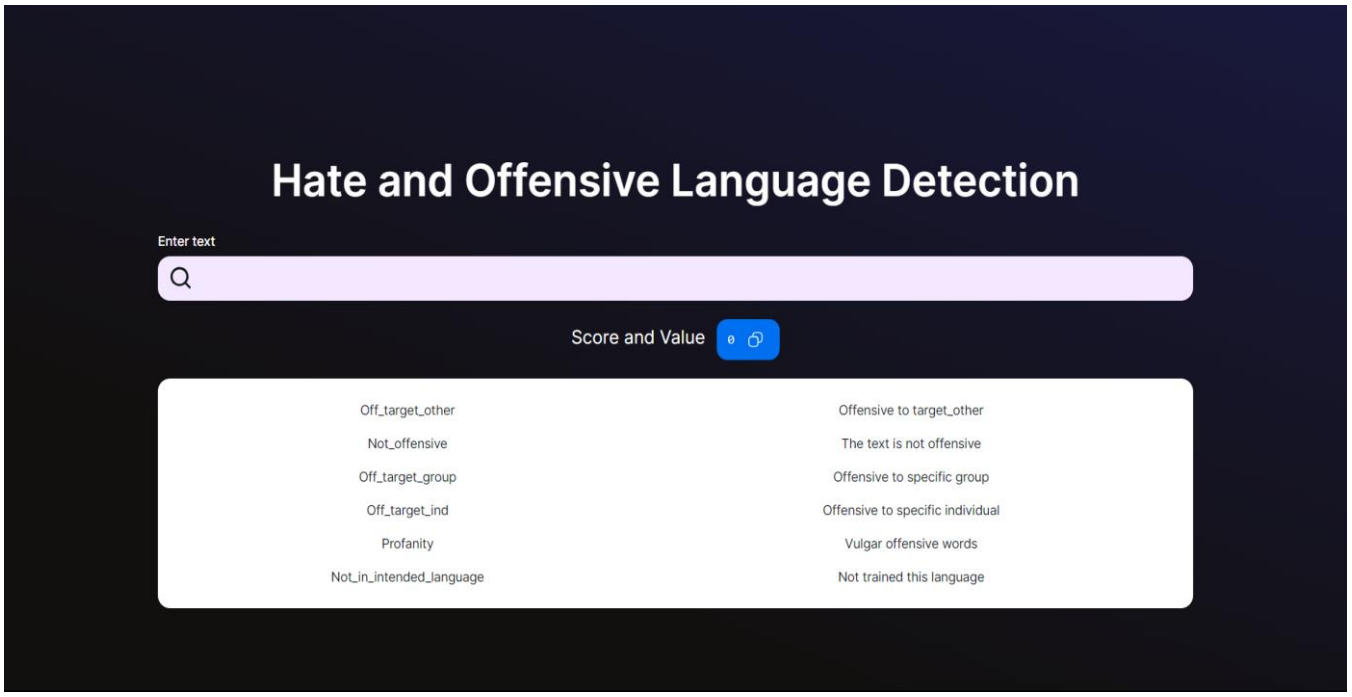


Fig No 5.4.1: Home Page

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 CODES:

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_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())
# 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)
```

CNN Model

```
from keras.layers.pooling import MaxPooling2D

inputs = Input(shape=(15, 300, 1))
# embedding = Embedding(input_dim=vocabulary_size, output_dim=embedding_dim,
input_length=sequence_length)(inputs)

conv_0 = Conv2D(64, kernel_size=5, activation='relu', padding='valid')(inputs)
conv_1 = Conv2D(32, kernel_size=3, activation='relu', padding='valid')(conv_0)
conv_2 = Conv2D(32, kernel_size=3, activation='relu', padding='valid')(conv_1)
drop = Dropout(0.6)(conv_2)
conv_3 = Conv2D(16, kernel_size=3, activation='relu')(drop)
pool0 = MaxPooling2D(pool_size=(2, 2), padding='valid')(conv_1)
conv_4 = Conv2D(16, kernel_size=3, activation='relu')(pool0)
pool1 = MaxPooling2D(pool_size=(2, 2), padding='valid')(conv_2)
# pool2 = MaxPooling2D(pool_size=(2, 2), padding='valid')(conv_3)
# maxpool_0 = MaxPool2D(pool_size=(sequence_length - filter_sizes[0] + 1, 1),
strides=(1,1), padding='valid')(conv_0)
# maxpool_1 = MaxPool2D(pool_size=(sequence_length - filter_sizes[1] + 1, 1),
strides=(1,1), padding='valid')(conv_1)
# maxpool_2 = MaxPool2D(pool_size=(sequence_length - filter_sizes[2] + 1, 1),
strides=(1,1), padding='valid')(conv_2)

# concatenated_tensor = Concatenate(axis=1)([pool0, pool1, pool2])
```

```
flatten = Flatten()(pool1)
hidden1 = Dense(128, activation='relu')(flatten)
output = Dense(6, activation='softmax')(hidden1)

# this creates a model that includes
model1 = Sequential()  # To train the model on dataset
model2 = Sequential()  # To extract embeddings from cnn layer
model1 = Model(inputs=inputs, outputs=output)
model2 = Model(inputs=inputs, outputs=hidden1)
adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Training the model
epoch = 50
cnt = 0
best_f1 = 0
for i in range(epoch):
    model1.fit(x_train, y_train, epochs=1, validation_data=(x_dev, y_dev))
    pred = model1.predict(x_train)
    prediction = []
    for i, j in enumerate(pred):
        a = np.argmax(j)
        prediction.append(a)
    y_true = []
    for i, j in enumerate(y_train):
        a = np.argmax(j)
        y_true.append(a)
    train_f1 = f1_score(y_true, prediction, average='weighted')
    print("train f1 - ", train_f1)
```

```
    pred = model1.predict(x_dev)
    prediction = []
    for i, j in enumerate(pred):
        a = np.argmax(j)
        prediction.append(a)

    y_true = []
    for i, j in enumerate(y_dev):
        a = np.argmax(j)
        y_true.append(a)

    val_f1 = f1_score(y_true, prediction, average='weighted')
    # Updating best F1 socre and saving corresponding embeddings
    if (val_f1 > best_f1):
        cnt = 0
        best_f1 = val_f1
        x_train_dense_cnn = model2.predict(x_train)
        x_dev_dense_cnn = model2.predict(x_dev)
        x_test_dense_cnn = model2.predict(x_test)

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

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

np.save('/home/punyajoy/Dravidian_Offensive_Classification/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

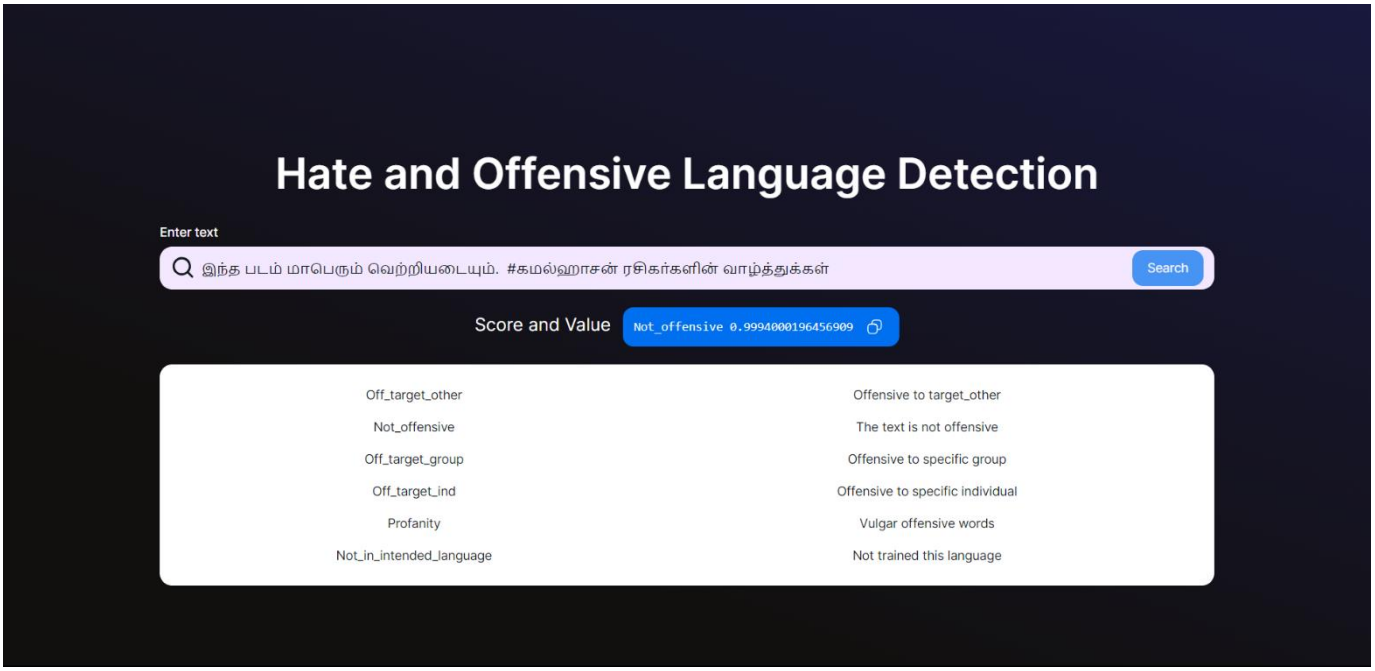


Fig No 6.2.1: Prediction for Not – Offensive Language

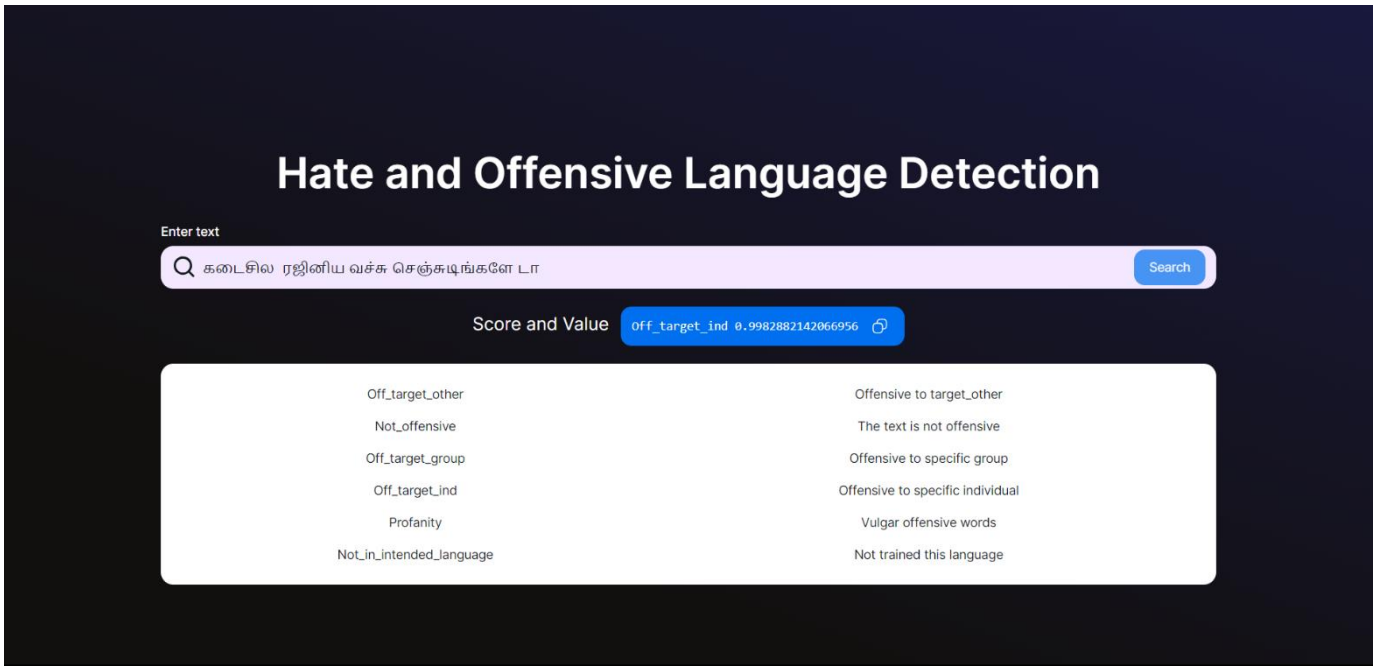


Fig No 6.2.2: Prediction for Offensive Language

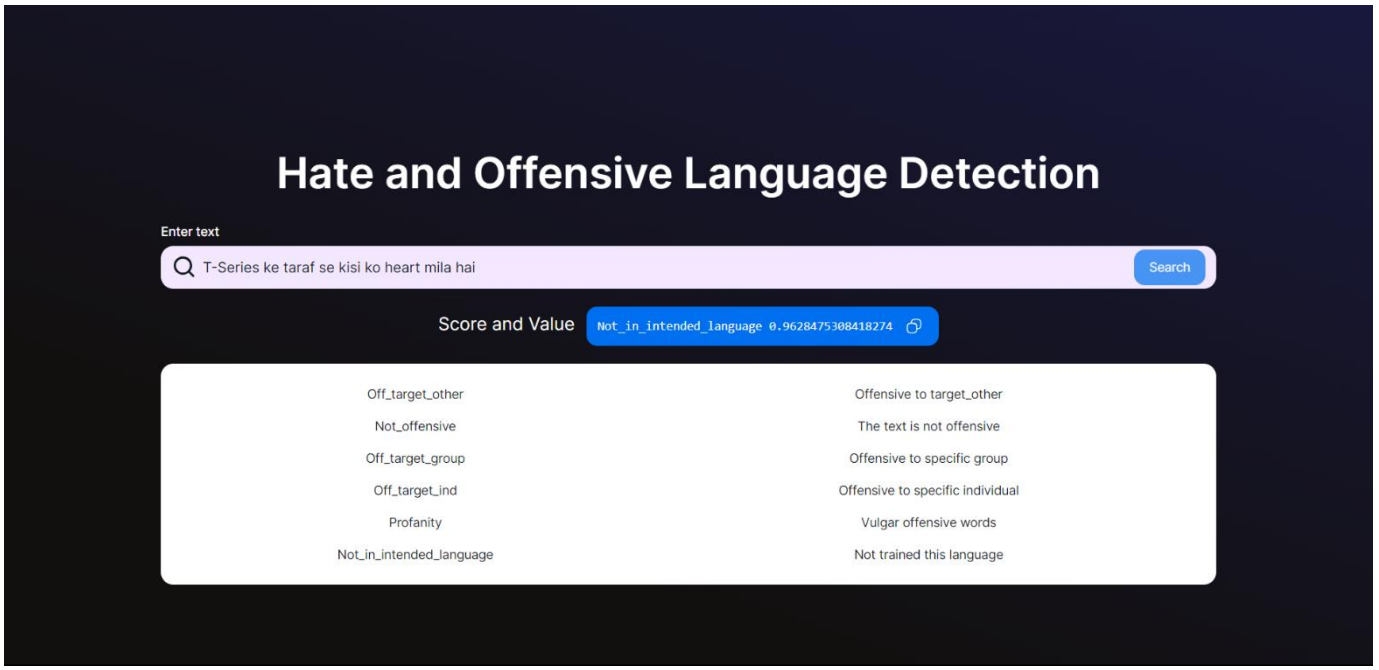


Fig No 6.2.3: Prediction for Not in Intended Language

CHAPTER 7

CONCLUSION

While several studies have been published on offensive language identification, there is considerable room for experimentation. New insights into low-resource Dravidian languages may result in more efficient and accurate models. Furthermore, to the best of our knowledge, this is the first study to propose an exhaustive examination of models, and newly proposed fusion of BERT and CNN model used for offensive language identification in three Dravidian languages. As demonstrated in this paper, such models perform admirably on a particular language but do not generalise effectively. It can be deduced that the application of deep neural networks to identify offensive language in Dravidian languages is promising. In the future, we will examine better neural network architectures apart from CNN and pre-trained embedding.

CHAPTER 8

FUTURE WORKS

In our ongoing project, we have successfully implemented an offensive language detection system for the Tamil language using a combination of Convolutional Neural Networks (CNN) and BERT Multilingual models, based on a research paper dataset. As we move forward, we intend to channel our efforts towards further research and development within the Dravidian language family.

Our upcoming goals primarily revolve around extending our offensive language detection capabilities to encompass additional Dravidian languages, specifically Malayalam, Telugu, and Kannada. This strategic expansion is aimed at enhancing the effectiveness and inclusivity of our system, ensuring that it can adequately address the unique linguistic characteristics and nuances present in these languages.

By concentrating on the Dravidian language family, we aim to contribute to a broader and more culturally diverse range of natural language processing solutions. Our commitment to this project remains resolute, as we seek to create a robust and reliable offensive language detection system that serves a significant portion of the linguistic diversity within the Dravidian language family.

CHAPTER 9

REFERENCE

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.neucom.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.2212.00851 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/10.1007/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. 10.1007/978-981-15-0694-9_41. 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/10.1016/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.