SCHOOL OF GRADUATE STUDIES

SENTIMENT ANALYSIS FOR AMHARIC-ENGLISH CODE-MIXED SOCIO-POLITICAL POSTS USING DEEP LEARNING

MSC THESIS

YITAYEW EBABU

May 28, 2024

WOLKITE, ETHIOPIA

Wolkite University

School of Graduate Studies

Sentiment Analysis for Amharic-English Code-Mixed Socio-Political Posts Using Deep Learning

A Thesis Submitted to the School of Graduate Studies, in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Yitayew Ebabu

Advisor: Mesfin Abebe (Ph.D.)

May 28, 2024

Wolkite, Ethiopia

APPROVAL SHEET

WOLKITE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby certify that we have read and evaluation this Thesis titled "**Sentiment Analysis for Amharic-English Code-Mixed Socio-Political Posts Using Deep Learning**" Prepared under our guidance by ***Yitayew Ebabu Alebachew.*** We recommended that the thesis shall be submitted as fulfilling the requirements for the award of a MSc. Degree in Computer Science and Engineering.

Approved by:

Dr. MESFIN ABEBE		5/28/2024
Name of Major Advisor	Signature	Date
Name of CO-Advisor	Signature	Date

As member of the Board of Examiners of the master of science thesis open defense examination, we have read and evaluated this thesis prepared by ***Yitayew Ebabu Alebachew*** and examined the candidate. We here by certify that the thesis is accepted for fulfilling the requirements for the award of the degree of Master of Science (M.Sc.) in computer science and engineering.

Bante M. (Ph.D.)		05/28/2024
Name of External Examiner	Signature	Date
Name of Internal Examiner	Signature	Date
Name of Chairman	Signature	Date

Final approval and acceptance of the thesis is contingent upon the submission of its final copy to the council of post graduate program (CPGS) through the candidate's department or school graduate committee (DGC or SGC).

Dedication

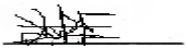
I dedicate this to my beloved family, whom I greatly admire, for their unwavering love, support, and resilience since the very start of my educational journey!

Declaration

By my signature, I declare and affirm that this Thesis is my own work. I have followed all ethical principles of scholarship in the preparation, data collection, data analysis and completion of this thesis. All scholarly matter is included in the thesis has been given recognition through citation. I affirm that I have cited and referenced all sources used in this document. Every serious effort has been made to avoid any plagiarism in the preparation of this document.

This thesis is submitted in partial fulfillment of the requirement for a degree from the school of graduate Studies at Wolkite University. This thesis is deposited in the wolkite university Library and is made available to borrow under the rules of the library. I solemnly declare that this thesis has not been submitted to any institution anywhere for the award of any academic degree, diploma or certificate.

Brief quotations from this Thesis may be used without special permission provided that accurate and complete acknowledgment of the source is made. Requests for permission for extended quotations from, or reproduction of, this thesis in whole or part may be granted by the Head of the School or Department or the Dean of the School of Graduate Studies when in his or her judgment the proposed use of the material is in the interest of scholarship. In all other instances, however, permission must be obtained from the author of the thesis.

Name: Yitayew Ebabu Signature: 

Date:

School/Department: Software Engineering Department

Acknowledgments

I am deeply grateful to my advisor, Dr. Mesfin Abebe, for his invaluable guidance and unwavering support throughout my thesis titled "Sentiment Analysis for Amharic-English Code-Mixed Socio-Political Posts Using Deep Learning". Dr. Mesfin's expertise in sentiment analysis has significantly influenced the direction of my research. His insightful feedback and constructive criticism have played a crucial role in refining the objectives and methodology of my study. I am extremely thankful for the time and effort he dedicated to reviewing and examining each chapter of my thesis, which has greatly improved its quality. I have learned not only technical aspects but also clear communication and effective writing skills from his guidance. Dr. Mesfin's encouragement and motivation have kept me focused and determined, even during challenging times. His unwavering belief in my abilities has boosted my confidence and pushed me to deliver my best. The countless discussions, brainstorming sessions, and valuable advice he has provided have broadened my knowledge and understanding of sentiment analysis. Moreover, I am thankful to Dr. Mesfin for encouraging in me a passion for research and critical thinking. In conclusion, I am massively grateful to Dr. Mesfin for his exceptional guidance and support. I consider myself privileged to have been advised by such a remarkable mentor. I sincerely appreciate his time, patience, and commitment in shaping my research and helping me navigate this challenging journey.

Table of Contents

Abstract

Sentiment analysis is crucial in natural language processing for identifying emotional nuances in text. Analyzing sentiment in natural language text is essential for discerning emotional subtleties. However, this task becomes especially intricate when dealing with code-mixed texts, like Amharic-English, which exhibit language diversity and frequent code-switching, particularly in social media exchanges. In this investigation, we propose employing CNN, LSTM, BiLSTM, and CNN-BiLSTM models to tackle sentiment classification in such code-mixed texts. Our approach involves leveraging deep learning techniques and various preprocessing methods, including language detection and code-switching integration. We conducted four experiments utilizing Count Vectorizer and TF-IDF. Our assessment reveals that incorporating language detection and code-switching significantly boosts model accuracy. Specifically, the average accuracy of the CNN model increased from 82.004% to 84.458%, the LSTM model from 79.716% to 81.234%, the BiLSTM model from 81.586% to 83.402%, and the CNN-BiLSTM model from 82.128% to 84.765%. These results underscore the efficacy of tailored preprocessing strategies and language detection in enhancing sentiment classification accuracy for code-mixed texts. Our study emphasizes the imperative of addressing language diversity and code-switching to achieve dependable sentiment analysis in multilingual environments. Furthermore, it provides valuable insights for future research, highlighting the importance of language-specific preprocessing techniques to optimize model performance across diverse linguistic contexts.

Keywords: Amharic-English, deep learning, Count Vectorizer, natural language processing, Sentiment analysis

List of Figures

Lists of Tables

Lists of Abbreviations

BPTT	backpropagation through time
BILSTM	Bidirectional Long Short-Term Memory
CBOW	Continuous Bag of Words
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Cross Validation
IDF	Inverse Document Frequency
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
OBN	Oromia Broadcasting Network
RegEx	Regular Expression
RF	Random Forest
RNN	Recurrent Neural Network
SA	Sentiment Analysis
SVM	Support Vector Machine
TF	Term Frequency
TF-IDF	Term Frequency- Inverse Document Frequency

CHAPTER ONE

1. Introduction

1.1. Backgrounds of the Study

Social media platforms, encompassing a wide array of formats such as blogs, forums, wikis, review websites, social networks, and tweets, have become vital channels for individuals around the globe to share information, express emotions, and exchange ideas. This dynamic interaction results in the generation of vast volumes of textual data[1], [2]. Blogs allow for in-depth discussion and personal reflection, forums facilitate community-based question and answer, wikis enable collaborative information building, review websites provide consumer feedback, social networks connect individuals and groups for real-time interaction, and tweets offer rapid, concise updates and opinions.

Consequently, there is an escalating need to analyze user-generated opinions across various domains to facilitate informed decision-making[3],[4]. Stakeholders, including companies, legislators, service providers, social psychologists, and academics, increasingly recognize the importance of examining these opinions[5],[6]. Their primary goal is to gain insights into public sentiment for a range of purposes, such as understanding consumer reactions to products, predicting political elections, and addressing socioeconomic concerns like stock market trends[7], [8].

For instance, businesses analyze customer feedback on social media to refine their products and services, legislators gauge public opinion to inform policy decisions, service providers monitor customer satisfaction to improve service delivery, social psychologists study social media interactions to understand human behavior, and academics use social media data to conduct research in various fields.

The field of sentiment analysis has gained prominence, centering on the assessment and comprehension of sentiments conveyed in textual data[9]. This undertaking employs natural language processing (NLP) and computational linguistics to methodically detect and extract subjective information from source materials. Examining sentiment in social media comments offers businesses invaluable insights and holds significance in various domains,

including social media analytics, customer feedback assessment, market research, and political analysis[10].

For example, companies can gauge public sentiment toward their latest product launch by analyzing tweets and Facebook posts, enabling them to tailor marketing strategies accordingly. Similarly, in political contexts, sentiment analysis can forecast election outcomes by scrutinizing the tone of online discussions regarding candidates and policies.

While sentiment analysis is a well-established field within social media analysis and natural language processing (NLP), its exploration has predominantly occurred in monolingual settings to address sentiment-related inquiries. However, this monolingual focus restricts the applicability of sentiment analysis in increasingly multilingual online environments.

Conversely, code-mixing research has primarily concentrated on combining English with other languages due to English's widespread usage as the world's second most spoken language. This emphasis stems from the prevalence of code-mixing involving English, presenting distinct challenges for sentiment analysis. For instance, an English-Spanish code-mixed sentence may contain words and expressions from both languages, necessitating sophisticated tools for accurate sentiment interpretation.

In this study, our aim is to bridge this gap by developing a deep learning model tailored for multiclass sentiment analysis of English-Amharic code-mixed social media comments related to socio-political posts. By focusing on English-Amharic code-mixed content, we address a significant yet underexplored area within sentiment analysis, acknowledging the increasing prevalence of such language combinations on social media platforms. Amharic, as the official language of Ethiopia, combined with English, mirrors the linguistic diversity observed in many social media interactions within the Ethiopian diaspora and other multilingual communities.

This study aims to assess the effectiveness of the proposed model through a series of experiments and analyses. To evaluate and compare the outcomes, we will employ well-established sentiment analysis metrics, including precision, recall, accuracy, and the F1-measure. Precision gauges the accuracy of positive predictions, recall measures the capacity to capture all relevant instances, accuracy assesses the overall correctness of the model, and

the F1-measure provides a balanced view between precision and recall. These metrics will provide valuable insights into sentiment analysis models' performance and effectiveness, thereby contributing to advancing sentiment analysis in multilingual and code-mixed contexts.

The findings of this research will not only deepen the comprehension of sentiment in English-Amharic code-mixed social media comments but also establish a framework for future studies in other code-mixed languages. By leveraging advanced deep learning techniques like convolutional neural networks (CNNs) and recurrent neural networks (RNNs), this study aims to expand current sentiment analysis methodologies' boundaries and set the stage for more sophisticated and accurate analyses in increasingly multilingual online environments. This will ultimately facilitate more precise sentiment analysis, enabling better decision-making based on a more precise understanding of diverse online communications.

1.2. Statement of the Problem

In recent years, the rapid growth of various social media platforms and the rise of multilingual communication have brought significant attention to code-mixed texts in sentiment analysis research[11]. Code mixing, the practice of blending two or more languages within a single utterance or text [11],[12],[13]., is prevalent in bilingual or multilingual communities. People engage in code mixing for various reasons, such as expressing cultural identity, linguistic convenience, or social integration[12], [14],[15],[16].

This trend is particularly widespread in multilingual communities, including those that employ a combination of Amharic and English. However, conventional sentiment analysis models, primarily designed for monolingual text, frequently encounter difficulties in accurately comprehending and categorizing the sentiments conveyed in code-mixed text due to its distinctive linguistic complexities.

Our proposed solution involves incorporating English words into the sentiment analysis process to leverage their contextual significance and emphasis, thereby enabling a more accurate determination of the expressed sentiment. By integrating both languages, we aim to improve the effectiveness of sentiment analysis for Amharic-English code-mixed texts.

Understanding sentiments in code-mixed texts is crucial for various applications, including social media monitoring, political analysis, and market research. Accurate sentiment analysis

of code-mixed texts is crucial for several reasons. Firstly, it enables governments and policymakers to better understand public opinion and emerging trends, thereby allowing them to make more informed decisions. Secondly, it helps political campaigns and organizations tailor their communication strategies to effectively engage with diverse audiences. Thirdly, it aids in the early detection of harmful discourse, such as polarization and extremism, promoting a more inclusive and constructive public dialogue. Moreover, insights from sentiment analysis can inform the creation of policies that are responsive to the needs and concerns of multilingual populations, fostering social cohesion and intercultural understanding. Therefore, addressing the complexities of sentiment analysis in code-mixed socio-political texts is essential for enhancing the effectiveness of communication and governance in increasingly diverse societies.

This study aims to address the challenges of sentiment analysis in Amharic-English code-mixed texts by exploring the following research questions:

RQ1: How does sentiment analysis perform on code-mixed texts compared to monolingual texts, particularly in the context of Amharic-English communication?

RQ2: How are vectorization techniques best suited for the proposed model to effectively classify sentiment in code-mixed text?

RQ3: To what extent do the proposed deep learning algorithms demonstrate effectiveness in multiclass sentiment classification of code-mixed text?

By addressing these research questions, this study aims to contribute to the understanding of sentiment analysis in code-mixed texts, with a focus on Amharic-English text. The findings will provide insights into the performance of sentiment analysis methods, the suitable vectorization techniques, and the effectiveness of deep learning algorithms for multiclass sentiment classification in code-mixed text. Ultimately, this research will facilitate the development of improved sentiment analysis models for code-mixed language data, enabling better understanding and interpretation of sentiment in diverse linguistic contexts.

1.3. Objectives of the Study

1.3.1. General Objective

The primary objective of this research is to develop and evaluate a deep learning-based multiclass sentiment analysis model for Amharic-English code-mixed social media texts.

1.3.2. Specific Objectives

The study aims to achieve the following specific objectives:

- To explore novel preprocessing techniques and language-specific adaptations to enhance the accuracy and reliability of sentiment analysis models for code-mixed texts.
- To evaluate the performance of sentiment analysis on code-mixed texts compared to monolingual texts, specifically focusing on Amharic-English communication.
- To investigate the impact of linguistic diversity and code-mixing patterns on sentiment analysis outcomes in Amharic-English communication.
- To analyze the impact of using various vectorization techniques on the sentiment classification performance of the proposed model for code-mixed data.
- To compare the sentiment classification results obtained using the various vectorization techniques to identify the most effective approach for the code-mixed text.

1.4. Scope and Limitations of the Study

The scope of this study focuses on multiclass sentiment classification of English-Amharic code-mixed text using deep learning approaches, specifically targeting on social media texts including code-mixed texts. The study aims to address the challenges and opportunities associated with sentiment analysis in this context and propose effective strategies for handling language mixing. It also aims to compare and evaluate different models and assess the performance of the developed model. This analysis aims to understand the sentiment of the code-mixed text and categorize it as positive, negative, or neutral.

When analyzing sentiment in code-mixed texts, especially those involving multiple languages like Amharic-English, it's crucial to acknowledge various limitations and potential biases:

- Linguistic Complexity: Code-mixing introduces linguistic complexities as different languages are merged within a single expression. Traditional sentiment analysis algorithms may struggle to accurately interpret sentiments in such mixed-language contexts.

- Lexical and Semantic Differences: Models trained on monolingual data may not capture the nuanced meanings of code-mixed words or phrases. Sentiment lexicons or embeddings trained on one language may not adequately represent sentiments in code-mixed texts.
- Data Sparsity: Limited availability of labeled code-mixed datasets, particularly in less-studied language pairs, can hinder the training of accurate sentiment analysis models, leading to performance and generalization issues.
- Cultural and Sociolinguistic Factors: Sentiment expressions in code-mixed texts are influenced by cultural and sociolinguistic factors unique to the language communities involved. Failure to account for these factors can introduce biases in sentiment analysis outcomes.
- Annotation Ambiguity: Manual sentiment annotation in code-mixed texts can be subjective and prone to ambiguity, particularly when annotators have different linguistic backgrounds or cultural perspectives, leading to inconsistent annotations and reduced reliability.
- Transliteration challenge: The transliteration challenge is particularly prominent in scenarios involving code-mixing or code-switching, where multiple languages with different writing systems are used within the same text. Effectively addressing this challenge is crucial for tasks such as sentiment analysis, machine translation, and information retrieval in multilingual and code-mixed language contexts. Addressing the transliteration challenge often involves the development of language-specific transliteration rules, the use of machine learning models trained on parallel corpora, or the application of hybrid approaches that leverage both rule-based and data-driven techniques.

Addressing these challenges requires specialized sentiment analysis techniques tailored to code-mixed texts, including language-specific sentiment lexicons, cross-lingual sentiment transfer learning, and considerations of cultural and sociolinguistic factors. Rigorous evaluation on diverse code-mixed datasets is essential to ensure the robustness and reliability of sentiment analysis models in such contexts.

1.5. Significance of the Study

This study significantly advances the field of sentiment analysis in code-mixed texts, particularly in contexts involving English and Amharic, by tackling prevalent challenges in multilingual societies and digital platforms. It fills a notable research gap, underscores the importance of addressing challenges in multilingual contexts, and provides invaluable insights and recommendations for future researchers, developers, and organizations engaged in sentiment analysis and social media analytics. The study's evaluation of sentiment analysis performance on code-mixed texts compared to monolingual texts, with a focus on Amharic-English communication, sheds light on the effectiveness of existing models and techniques in handling code-mixing phenomena, contributing to our understanding of language variation's impact on sentiment classification accuracy.

1.6. Organization of the Study

The subsequent chapters of this thesis are structured into distinct sections. Chapter two provides an in-depth overview of sentiment analysis, encompassing a thorough review of past research and critical analysis to identify areas for further investigation, along with a concise summary of its contents. Moving to Chapter three, a detailed explanation of the research approach and design is presented, covering dataset preparation, preprocessing techniques, data analysis methods, and sentiment analysis model selection. Chapter four delves into the design of the sentiment detection and classification model used for this thesis. Chapter five delves into the development and implementation of the sentiment detection and classification model used in the thesis. Chapter six focuses on presenting experimental results and datasets created for sentiment detection and classification, offering a comprehensive exploration of the experiment outcomes. Finally, Chapter seven serves as the concluding section, summarizing key concepts discussed and outlining potential avenues for future research exploration. The following figure illustrates the general organization of this thesis structure.

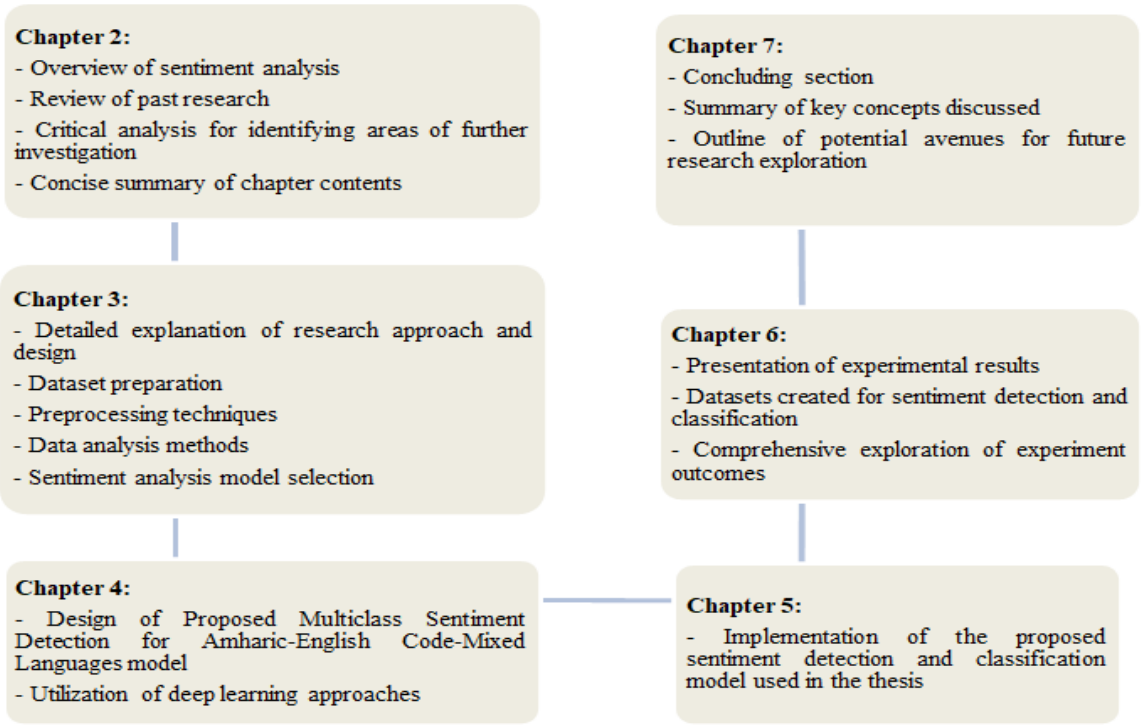


Figure 1. Organization of Thesis Structure

CHAPTER TWO

2. Literature Review and Related Works

2.1. Introduction

The literature review and related works section of this study examines previous research and scholarly contributions in the field of sentiment analysis. Its primary goal is to establish a strong basis by exploring prior studies, methodologies, and findings that are relevant to sentiment analysis. The section critically evaluates different methodologies, highlighting their strengths and limitations, and emphasizes significant findings and insights from existing research studies. Through a comprehensive literature review, the study aims to identify research gaps, and synthesize the literature to guide the research approach and methodology. Ultimately, the literature review and related works section are essential for establishing a theoretical foundation, demonstrating the researcher's familiarity with the existing literature, and providing a framework for subsequent chapters.

2.2. Definition of Code-Mixing

Code-mixing refers to the practice of alternating between two or more languages within a single conversation or discourse[17],[18]. It encompasses various types, including intra-sentential code-mixing, where different languages are mixed within the same sentence, and inter-sentential code-mixing, where language switches occur between sentences or clauses. Moreover, code-mixing can manifest in different forms, such as insertion (embedding words or phrases from one language into another), alternation (switching between languages at specific points), and congruent lexicalization (using language-specific structures within a mixed-language utterance).

In Amharic-English text, code-mixing is pervasive, reflecting the linguistic diversity and bilingual proficiency of speakers. This phenomenon is observed across various domains, including informal conversations, social media interactions, and online discourse[17],[18]. Speakers seamlessly integrate English borrowings, expressions, and idioms into their Amharic speech, creating hybrid linguistic forms that reflect cultural identity and social belonging. The fluidity of language alternation in Amharic-English communication poses challenges for sentiment analysis algorithms, as sentiment expressions may be embedded within code-mixed utterances, requiring sophisticated techniques for accurate interpretation.

2.3. Sentiment Analysis

Sentiment analysis is a fundamental natural NLP technique that categorizes text into positive, negative, or neutral tones to distinguish emotional sentiments within documents [17],[18],[19],[20]. It holds significant value for individuals and organizations seeking to understand public opinions and sentiments towards various topics, products, or services [21]. The rise of social media platforms has heightened interest in sentiment analysis due to the abundance of user-generated content available [18],[22]. Customer reviews, social media interactions, and news articles serve as vital resources for businesses to gauge public sentiment and make informed decisions.

However, analyzing sentiment in code-mixed social media texts, particularly in languages like Amharic-English, presents numerous challenges due to the complex language mixture and linguistic diversity observed in countries like Ethiopia. Sentiment analysis of code-mixed socio-political texts can provide valuable insights into the nuanced attitudes, opinions, and underlying emotions expressed by individuals and communities on various socio-political issues. Some key applications of sentiment analysis in this domain include:

- **Monitoring and Understanding Public Sentiment**: Code-mixed posts and comments on social media platforms, online forums, and news articles can offer a rich tapestry of public sentiment on socio-political topics. Analyzing the sentiment of these code-mixed texts can help governments, policymakers, and researcher's better gauge public opinion, identify emerging trends, and understand the underlying drivers of societal sentiments.
- **Informing Political Campaigns and Communication Strategies**: In multilingual and diverse societies, political actors often employ code-mixing to connect with their constituents. Sentiment analysis of code-mixed socio-political texts can help campaign teams and political organizations better understand the sentiments of their target audiences, tailor their messaging and communication strategies accordingly, and respond more effectively to evolving public concerns.
- **Informing Policymaking and Governance**: Insights derived from sentiment analysis of code-mixed socio-political texts can inform the decision-making processes of

policymakers and government officials. By understanding the sentiments and concerns of diverse communities, they can develop more responsive and inclusive policies that address the needs and aspirations of the population.

2.1. Sentiment Analysis Approaches

There are various methods available for analyzing emotions, each presenting distinct advantages and considerations. The primary approaches encompass lexicon-based techniques, machine learning, and hybrid methods.

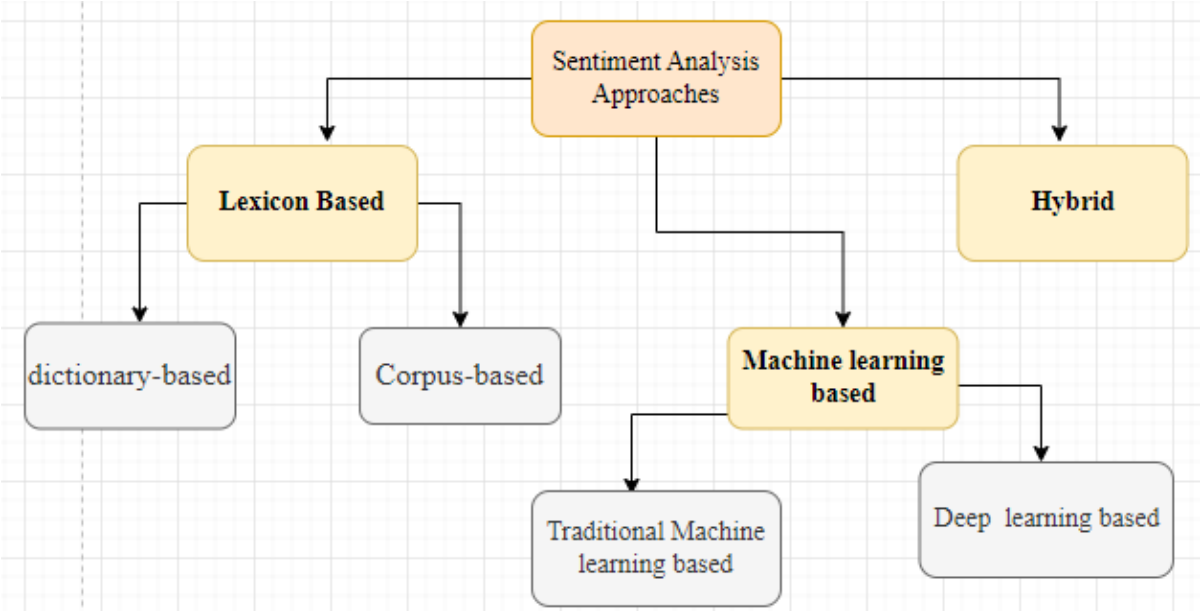


Figure 2. Sentiment Analysis Approaches

2.1.1. Lexicon-Based Techniques

Lexicon-based techniques rely on sentiment lexicons, which are dictionaries assigning sentiment scores to terms [15],[26],[7]. These techniques are commonly classified into two categories: dictionary-based and corpus-based. In the dictionary-based approach, sentiment classification makes use of a predefined dictionary of terms [27].

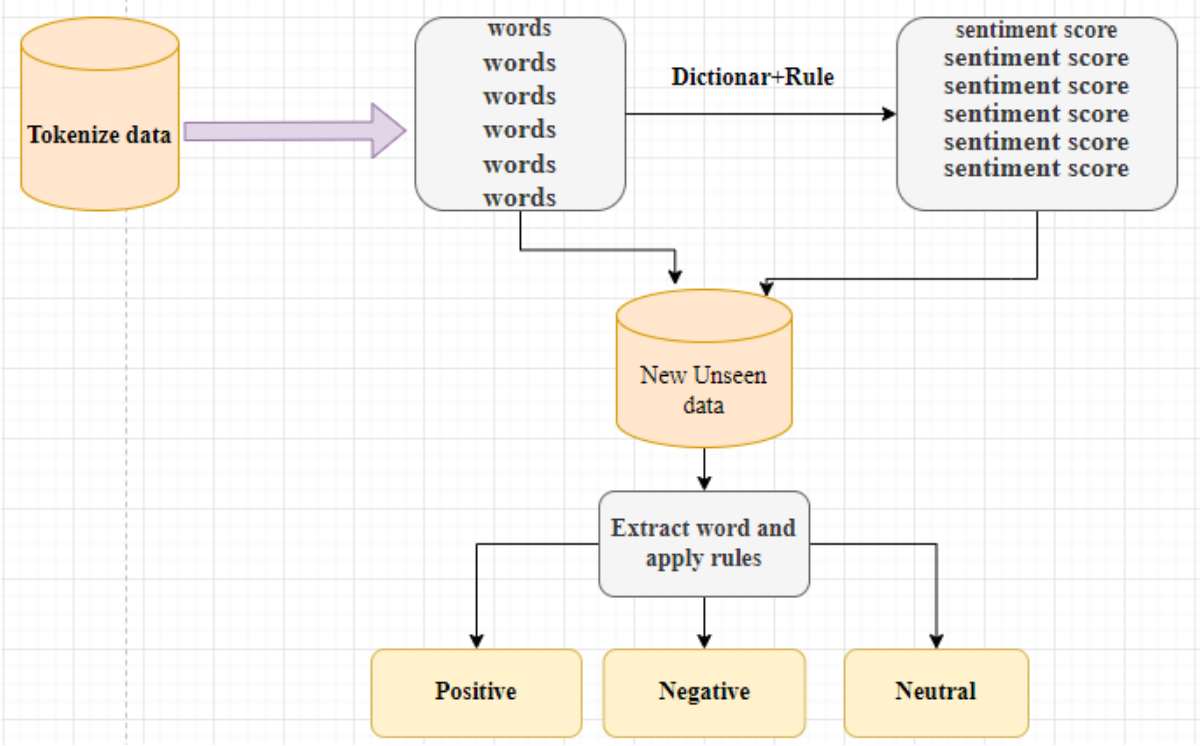


Figure 3. Lexicon -Based Approach

Each term in the dictionary is assigned a sentiment score corresponding to positive, negative, or neutral sentiments [15]. To conduct sentiment analysis using this approach, the text is first tokenized into individual words or phrases. Subsequently, each word or phrase is referenced in the sentiment dictionary to retrieve its sentiment score. The sentiment scores of all words or phrases are then combined to determine the overall sentiment of the text. Conversely, the corpus-based approach achieves sentiment classification through statistical analysis of content within a document collection [15],[20],[26]. This method involves utilizing a training corpus to understand statistical patterns and relationships between words and sentiments. The training corpus comprises labeled documents, where each document is annotated with its corresponding sentiment. By training statistical models on this corpus, the approach facilitates the classification of new, unseen documents based on their sentiment.

2.1.2. Machine Learning-based Techniques

Sentiment analysis in machine learning relies on training models with labeled data to distinguish patterns and relationships between words and sentiments [28]. These techniques fall into two categories: traditional models and deep learning models [29]. Traditional

methods, including support vector machines (SVM), maximum entropy classifiers, and Naive Bayes classifiers, utilize various features such as sentiment-based attributes, lexical features, and parts of speech [4],[29].

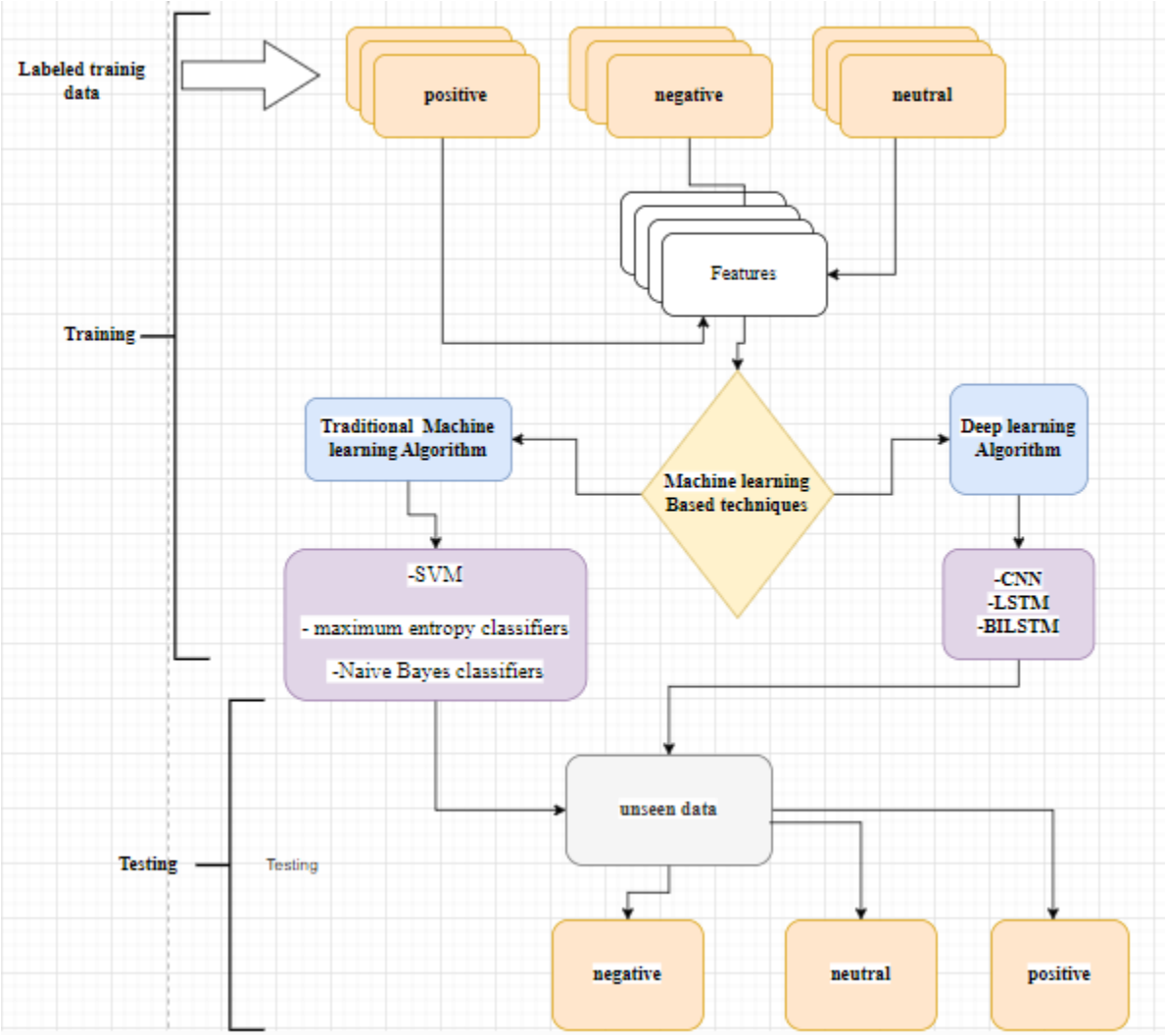


Figure 4. Machine Learning-based Techniques

To perform sentiment analysis with traditional models, a labeled training dataset serves as the basis for model training. Features are extracted from the text, and the model is trained to categorize the text into positive, negative, or neutral sentiment groups. Trained models can effectively classify new and unseen texts based on their prevailing sentiment.

Deep learning, which falls within the realm of machine learning, utilizes sophisticated algorithms to build models characterized by complex architectures featuring numerous nonlinear transformations [30]. The primary objective of these models is to grasp abstract

concepts within the data, thus enabling a more profound comprehension and analysis of the information they process. In the context of sentiment analysis, a task within natural language processing, deep learning techniques are employed to convert textual data into a numerical format, such as word embedding or character embedding. This numerical representation serves as input for the deep learning model, which then analyzes the complex patterns and correlations between the input data and the associated sentiment labels. By undergoing training using labeled datasets, deep learning models refine their capabilities, gradually enhancing their proficiency in accurately categorizing new textual inputs based on their sentiment, thereby contributing to the advancement of sentiment analysis methodologies.

2.1.3. Hybrid Approach

The hybrid approach integrates machine learning and lexicon-based techniques to enhance sentiment analysis[29]. Sentiment lexicons play a crucial role, enriching features used by machine learning models with added context and information. The sentiment analysis process involves lexicon-based techniques extracting sentiment-related features from the text. These features combined with others, like lexical features or parts of speech, forming the input for machine learning models trained on labeled data. The incorporation of sentiment lexicons improves accuracy and performance during sentiment analysis.

2.2. Feature Extraction Techniques

Text feature extraction is converting raw text data into numerical representations for analysis [31], [32]. This study explores common techniques, including Count Vectorizer and Term Frequency- Inverse Document Frequency (TF-IDF) vectorization, which serve as input for sentiment analysis models. By utilizing these techniques, such as TF-IDF vectorization, and count vectorization, the study aims to transform the raw text data into meaningful numerical representations that can be utilized by sentiment analysis models. These techniques enhance the machine's ability to understand and process human language, thereby improving the accuracy and effectiveness of sentiment analysis tasks [21].

2.2.1.N-Gram

An n-gram is a text that consists of n words, where n is a predetermined number, can have any value for n; it can be bigrams, trigrams, unigrams, etc. NLP uses n-grams for linguistic modeling and sentiment detection, among other applications. One can learn a great deal

about word relationships and patterns through looking at the frequency and arrangement of n-grams in a text. The n-gram range parameter in natural language processing refers to the range of n-grams considered when tokenizing text. In this case, n-gram_range=(1,3) means that we consider unigrams (individual words), bigrams (sequences of two words), and trigrams (sequences of three words). Let's explain n-gram_range=(1,3) using the following example sentences containing Amharic-English code-mixed texts:

1. Sample sentence 1: "ኢትዮጵያ corruption ባይኖር የት በደረሰች ነበር።"
2. Sample sentence 2: "ኢትዮጵያ the richest country የምትባለው መቼ ይሆን"

For the given sentences, considering n-gram_range=(1,3), then generate the following n-grams: Unigrams(individual words), Bigrams(sequences of two words), trigrams: sequences of three words).

Table 1.n-grams representation

Sentence	**Unigrams**	**Bigrams**	**Trigrams**
1	ኢትዮጵያ, corruption, ባይኖር, የት, በደረሰች, ነበር።	ኢትዮጵያ corruption, corruption ባይኖር, ባይኖር የት, የት በደረሰች, በደረሰች ነበር።	ኢትዮጵያ corruption ባይኖር, corruption ባይኖር የት, ባይኖር የት በደረሰች, የት በደረሰች ነበር።
2	ኢትዮጵያ, the, richest, country, የምትባለው, መቼ, ይሆን	ኢትዮጵያ the, the richest, richest country, country የምትባለው, የምትባለው መቼ, መቼ ይሆን	ኢትዮጵያ the richest, the richest country, richest country የምትባለው, የምትባለው መቼ, መቼ ይሆን

In the table, each row represents a sentence, and the columns represent the generated n-grams for each sentence. This table illustrates the application of n-gram_range=(1,3) in tokenizing the provided Amharic-English code-mixed sentences into unigrams, bigrams, and trigrams, capturing different levels of linguistic context.

2.2.2.TF-IDF

TF-IDF vectorization calculates weights for words based on frequency and inverse document frequency(IDF), numerically representing word importance in a document[19]. This technique is valuable for identifying significant words contributing to expressed sentiment. Term Frequency (TF), which measures how often a term appears in a document by dividing

its frequency by the total number of terms, and Inverse Document Frequency, which assesses the importance of a term across the entire document collection by taking the logarithm of the ratio between the total number of documents and the number of documents containing the term. By multiplying the TF and IDF values of a term, the TF-IDF score is calculated, with higher scores indicating greater importance or relevance of a term within a specific document compared to the entire document corpus.

By using TF-IDF in combination with unigrams and trigrams, the feature extraction process not only considers the frequency of individual words but also the importance of those words in the entire corpus. TF-IDF with n-gram_range=(1,3) involves considering not only individual words (unigrams) but also sequences of 2 words (bigrams) and sequences of 3 words (trigrams) to capture more comprehensive contextual information from the text. This can help in identifying key terms and phrases that are significant in the text data. Let's explain TF-IDF vectorization with n-gram_range=(1,3) using the following example sentences containing Amharic-English code-mixed texts:

- sentence 1: "ኢትዮጵያ corruption ባይኖር የት በደረሰች ነበር።"
- sentence 2: "ኢትዮጵያ the richest country የምትባለው መቼ ይሆን"

Firstly, generate n-grams (unigrams, bigrams, and trigrams) for each sentence as represented in Table 2. Next, compute the TF-IDF scores for each n-gram in the sentences. TF-IDF considers both the frequency of each n-gram in the document and its rarity across the entire corpus.

Sentence	ኢትዮጵያ	corruption	ባይኖር	የት	በደረሰች	ነበር	the	richest	country	የምትባለው	መቼ	ይሆን
Sentence 1	0.29	0.29	0.29	0.29	0.29	0.29	0	0	0	0	0	0
Sentence 2	0.29	0	0	0	0	0	0.29	0.29	0.29	0.29	0	

Table 2.TF-IDF representation

In the above, the TF-IDF scores for each n-gram in the sentences are presented. The values indicate the importance of each n-gram within the respective sentence and are computed based on its frequency within the sentence and its rarity across the entire corpus.

For instance, In Sentence 1, each n-gram has a TF-IDF score of 0.29, indicating their importance in this particular sentence. In Sentence 2, the n-grams "ኢትዮጵያ", "the", "richest",

"country", "የምትባለው", "መቼ", and "ይሆን" have non-zero TF-IDF scores, indicating their significance within this sentence. This TF-IDF representation captures the importance of n-grams in each sentence while considering their rarity across the entire corpus, making it a valuable tool for text analysis and classification tasks.

2.2.3. Count Vectorizer

The Count Vector technique is a straightforward approach for feature extraction in NLP, wherein text documents are transformed into vectors, with each element representing the count of a specific word in the document. By integrating both unigrams and trigrams into the Count Vectorizer, this process captures both individual words and sequences of three words in the text. Utilizing Count Vectorizer with n-gram_range=(1,3) entails converting a collection of text documents into a matrix that records token occurrences, accounting for not only single words (unigrams) but also sequences of two words (bigrams) and three words (trigrams). The resulting matrix takes the form of a table, where each row corresponds to a document and each column represents a unique n-gram. The cell value indicates the frequency of that n-gram within the respective document. To illustrate the application of Count Vectorizer with n-gram_range=(1,3), let's explore provided example sentences containing Amharic-English code-mixed texts.

- sentence 1: "ኢትዮጵያ corruption ባይኖር የት በደረሰች ነበር"
- sentence 2: "ኢትዮጵያ the richest country የምትባለው መቼ ይሆን"

Table 3. Count Vector representation for each n-gram

Document	ኢትዮጵያ	corruption	ባይኖር	የት	በደረሰች	ነበር።	the	richest	country	የምትባለው	መቼ	ይሆን
Sentence 1	1	1	1	1	1	1	0	0	0	0	0	0
Sentence 2	1	0	0	0	0	0	1	1	1	1	1	1

In the above table, each row corresponds to a sentence, and each column corresponds to an n-gram. The cell values represent the frequency of each n-gram in the respective sentence. This Count Vector representation provides a numerical input format suitable for machine learning algorithms to analyze and classify text data. In Sentence 1, the n-grams "ኢትዮጵያ", "corruption", "ባይኖር", "የት", "በደረሰች", and "ነበር።" each occur once, while the other n-grams have a frequency of 0. In Sentence 2, the n-grams "ኢትዮጵያ", "the", "richest", "country",

"የምትባለው", "መቼ", "ይሆን" each appear once, and the remaining n-grams do not appear in this sentence.

2.3. Types of Sentiment Analysis

There are several types of sentiment analysis, each categorized based on the nature of the sentiment evaluated. Some prevalent types include binary sentiment analysis, multi-class sentiment analysis, and emotion detection.

2.3.1. Binary Sentiment Analysis

Its primary goal is to determine whether a given piece of text expresses positive or negative sentiment[23]. For instance, consider the sentence, "I love the movie." In binary sentiment analysis, analysts categorize this sentence as expressing a positive sentiment. Conversely, if the sentence were "I hated the movie," analysts classify it as conveying a negative sentiment. Binary sentiment analysis has broad applications, including assessing consumer opinions, conducting social media sentiment analysis, and evaluating product feedback. It provides a foundational understanding of sentiment, serving as a valuable starting point for various sentiment analysis tasks.

2.3.2. Multi-Class Sentiment Analysis

This approach categorizes text into multiple sentiments, as positive, negative, and neutral sentiment classes[25]. Unlike binary sentiment analysis, it provides a deeper understanding of emotions. For instance, consider an item evaluation with the sentence: "Camera quality is good, but battery life is disappointing." Under multiclass sentiment analysis, this sentence designated as conveying both positive and negative sentiment. Commonly applied in market research, brand monitoring, and social media analysis, it enables a nuanced understanding of sentiment.

2.3.3. Multiclass Emotion Detection

Emotion detection surpasses mere emotion classification, aiming to pinpoint specific emotions conveyed in text, such as joy, sadness, anger, or fear[23],[24],[25]. Exploring emotional cues in-depth, emotion detection provides valuable insights into the emotional state of individuals or groups. For instance, consider the sentence: "I'm extremely excited about the upcoming concert!" In this particular context, the phrase conveys positive

emotions, particularly happiness. Conversely, if the sentence were to express "The news has left me feeling sad," it would indicate a leaning towards negative emotions, specifically sadness. Emotion detection is vital in domains where understanding the emotional state of individuals or groups holds significance. It finds utility in realms such as consumer feedback analysis, sentiment assessment on social media, and monitoring mental well-being.

2.5. Sentiment Analysis Levels

Many researchers in the field of Sentiment analysis operates at distinct levels mainly in a set of three levels, Document level sentiment classification ,Sentence level sentiment classification ,Aspect-based sentiment classification, each offering unique perspectives on the emotional content within text [33], [34], [35], [36]. By conducting sentiment analysis at these different levels in Amharic-English code-mixed texts, researchers and analysts can gain a deeper understanding of the emotions, opinions, and attitudes expressed in the text data, considering the nuances and complexities introduced by mixing languages.

2.5.1. Document Level

Document-level sentiment analysis provides a broad overview of emotional expressions within the text[38]. At this level, the analysis aims to determine the overarching sentiment conveyed in an entire document, whether it is an article, review, or social media post[19],[20], [4]. The objective is to capture the general tone of the complete document [35], [37]. For instance, a positive document might reflect satisfaction or appreciation, while a negative one could express dissatisfaction or criticism. In the case of Amharic-English code-mixed texts, document-level sentiment analysis would focus on determining the general sentiment conveyed throughout the entire document, considering both languages to capture the overall mood or opinion expressed.

2.5.2. Sentence Level

Sentence-level sentiment analysis operates on individual sentences within a document. This level evaluates the sentences independently to determine its sentiment[19][4]. This approach offers a nuanced understanding of sentiment distribution throughout the text[35]. For Amharic-English code-mixed texts, sentence-level sentiment analysis would involve assessing the sentiment of individual sentences in both languages to understand the emotions or opinions expressed within each sentence.

2.5.3. Aspect Level

The aspect level focuses on specific aspects or entities mentioned in the text[19],[20], [35], [39],[38]. For example, in product reviews, aspect-level analysis can reveal sentiments related to various product attributes like display design and customer service. This level of analysis provides insights into how different aspects of the text evoke emotional responses. In the context of Amharic-English code-mixed texts, aspect-based sentiment analysis would focus on identifying sentiments related to different aspects discussed in the text, considering both languages for a comprehensive analysis of opinions on various topics.

2.6. Related works for Sentiment Analysis

Researchers are advancing the development of tools to evaluate sentiment in Amharic, a language renowned for its intricate morphology and nuanced emotional expressions. Leveraging machine learning, deep learning, and lexicon-based techniques, they aim to enhance sentiment analysis capabilities. Recent progress in this field, driven by expanding datasets and accumulated knowledge, has significantly improved understanding sentiments conveyed in Amharic texts.

Numerous studies have contributed to sentiment analysis in Amharic and other code-mixed languages, employing various methodologies with differing degrees of success. This literature review will delve into existing research on sentiment analysis of code-mixed social media texts, exploring diverse approaches, methodologies, and tools used to tackle this complex challenge. Furthermore, we will analyze the challenges and limitations encountered by researchers in this domain, while also considering potential future research directions.

Sentiment analysis has received significant attention across various languages and domains. However, research focused on under-researched languages, such as Amharic, remains sparse. This review delves into existing studies on sentiment analysis using deep learning, specifically targeting Amharic and other similar languages, to highlight the gaps and underscore the need for further research.

In a comprehensive approach, Researcher [40] presented a sentiment classification method for Amharic using deep learning and multilingual datasets. This study addressed the notable lack of resources for sentiment analysis in under-researched languages like Amharic. By

experimenting with several deep learning techniques, including CNN, LSTM, FFNN, BiLSTM, and transformers, the researchers classified sentiment in Amharic social media messages. Their findings revealed that machine translation and cross-lingual models can effectively grasp the nuances of the target language even when trained on another language. Notably, the FFNN classifier combined with sentence transformers showed superior performance, achieving high accuracy in both three-class and two-class sentiment classification tasks.

In contrast, [21] focused on sentiment analysis in Amharic political texts using CNN, BiLSTM, and a hybrid CNN-BiLSTM model. Achieving an accuracy of 91.60%, this study underscored the challenges of handling more nuanced sentiments, such as sarcasm, and the need for larger, more diverse datasets. The authors advocated for transitioning from binary to multi-class sentiment analysis to enhance understanding. They also stressed the importance of a standardized corpus for Amharic sentiment analysis that spans various domains and includes sarcastic content as a future research avenue.

Another pivotal study by [32] investigated the impact of preprocessing techniques on LSTM-based sentiment analysis for Amharic. This research examined the effects of stemming and the integration of labeled emojis with lexicons for automatic labeling. It found that stemming reduces accuracy, whereas integrating emojis slightly improves it. The study highlighted the critical role of preprocessing in optimizing sentiment analysis performance for Amharic text.

Researcher [41] explored a deep sentiment analysis model utilizing a CNN-LSTM architecture for English and Roman Urdu social media texts. By evaluating different word embeddings, the study demonstrated high accuracy rates for sentiment classification in both languages. It underscored the importance of considering morphological complexities and dialect variations, particularly in Roman Urdu.

Similarly, [4] analyzed sentiments in code-mixed Bambara-French Facebook comments using LSTM and CNN-based models. To address the scarcity of Bambara text on social media, the researchers employed character and word dictionaries along with embedding techniques. The study achieved an accuracy rate of 83.23%, showcasing the efficacy of deep learning models in handling code-mixed texts.

The reviewed studies collectively contribute to the advancement of sentiment analysis across various languages, including Amharic, Roman Urdu, and Bambara-French. They demonstrate the potential of deep learning techniques in overcoming language-specific challenges. However, gaps remain, particularly in the comprehensive coverage of socio-political posts and the handling of nuanced sentiments like sarcasm. There is a need for more extensive datasets, standardized corpora, and improved preprocessing methods tailored to each language's unique characteristics.

While significant strides have been made in sentiment analysis for under-researched languages using deep learning, the field still requires a more extensive and comprehensive exploration, particularly for socio-political content. Future research should focus on creating larger, more diverse datasets, developing standardized corpora, and enhancing preprocessing techniques to better capture nuanced sentiments. This will not only advance sentiment analysis in under-researched languages but also contribute to a more nuanced understanding of multilingual social media content.

The strengths of the existing studies lie in their innovative use of deep learning techniques and their pioneering efforts in sentiment analysis for under-researched languages. However, weaknesses include limited dataset sizes, insufficient handling of nuanced sentiments, and the lack of standardized corpora. Comparing methodologies, techniques, and findings reveals the need for more robust preprocessing methods and comprehensive datasets. Addressing these limitations could significantly enhance the performance and applicability of sentiment analysis models in various linguistic contexts.

2.6.1. Summary of Related Works

In this section, we provide a summary of the related works in a table format for easy understanding.

Table4. Table of Related works

Author (Year)	Title	Methods	Findings (with Performance)	Limitation
Tesfagergish et al. (2023)[40]	Sentiment Analysis on Amharic Social Media Texts	CNN, LSTM, FFNN, BiLSTM, transformers, cosine similarity, word/sentence embedding techniques	Achieved 80% accuracy in sentiment analysis with proposed models.	translation of the English portions, rather than maintaining the code-mixed context, can actually change the overall meaning and sentiment of the whole text
Yeshiwas Getachew (2023)	Sentiment Analysis on Scraped Amharic Social Media Data	ANN with TF-IDF and Count Vectorizer	Attained 85% accuracy in classifying socio-political sentiments using deep learning approaches.	The foreign linguistic elements embedded within Amharic texts are frequently undervalued
Yohannes Mekonnen (2020) [34]	Hybrid Approach of Amharic Sentiment Analysis for Kana TV	Rule-Based, Dictionary-Based	Achieved an average accuracy of 85%, precision of 95%, recall of 89%, and F1-score of 91%.	not taking into consideration or analyzing any English language

				content that may be interspersed within the Amharic text
Alemayehu et al. (2023) [21]	Amharic Political Sentiment Analysis using Deep Learning Approaches	CNN, Bi-LSTM, Hybrid CNN-Bi-LSTM	Achieved 91.60% accuracy in sentiment classification. Emphasized the need for a standardized corpus and exploring sarcastic comments in Amharic. Recommended transitioning to multi-class classification.	The English language content interspersed within Amharic texts is often overlooked or neglected.
Wondwossen Mulugeta (2023) [3]	A Machine Learning Approach to Multi-Scale Sentiment Analysis of Amharic Online Posts	Naïve Bayes machine learning algorithm with unigram, bigram, and hybrid variants as features	Attained 44.3% accuracy using Bigram features for sentiment classification of social media, product marketing, and news content.	not taking into consideration or analyzing any English language content that may be interspersed within the Amharic text

Our research aims to fill the significant gap in sentiment analysis for Amharic-English code-mixed texts, a unique linguistic context largely overlooked in existing studies. While substantial progress has been made in sentiment analysis for monolingual Amharic texts and other code-mixed languages, there is a notable lack of research specifically focused on Amharic-English code-mixed data. This gap is crucial, given the increasing prevalence of Amharic-English code-switching in social media interactions, particularly in socio-political contexts.

This research is essential for several reasons. Firstly, it addresses the underexplored area of Amharic-English code-mixed sentiment analysis, filling a significant void in the literature. By developing specialized sentiment analysis models tailored to this linguistic combination, our research provides tools optimized for accurately capturing sentiments expressed in Amharic-English code-mixed texts. This is particularly important for socio-political contexts, where understanding public sentiment can influence policy-making, social dynamics, and community engagement.

Our study extends existing knowledge by overcoming key challenges in sentiment analysis of code-mixed texts. One such challenge is the typically small dataset sizes available for these analyses. We address this by prioritizing the acquisition of larger datasets, thereby enhancing the robustness and generalizability of our models. Additionally, we tackle the issue of code-switching detection through innovative methodologies designed to handle the complexities of language mixing effectively. These advancements push the boundaries of current sentiment analysis techniques, making them more applicable to multilingual environments.

The potential impact of our research is multifaceted. By expanding the scope of sentiment analysis to include multilingual expressions involving Amharic and English, our work enhances the applicability of sentiment analysis in regions like Ethiopia, where these languages are frequently mixed. This contribution is useful, as it facilitates better decision-making and provides informed insights into public sentiment. Stakeholders such as businesses, policymakers, and social researchers can leverage these insights to understand consumer behavior, gauge public opinion, and analyze social trends more accurately.

Furthermore, our research has the potential to set a precedent for future studies in other code-mixed language pairs, promoting a deeper understanding of sentiment dynamics in multilingual communities globally. By leveraging advanced deep learning techniques, we aim to improve the accuracy and sophistication of sentiment analysis models, paving the way for more effective analysis of diverse linguistic contexts.

In summary, our research addresses critical gaps in the literature by focusing on sentiment analysis of Amharic-English code-mixed texts. It extends existing knowledge by developing robust, specialized models and innovative methodologies to handle the complexities of code-switching and small dataset sizes. The potential impact of this research is significant, providing valuable insights into public sentiment in multilingual contexts and enhancing the practical applications of sentiment analysis in socio-political domains. Through these contributions, we aim to advance the field of sentiment analysis and support better decision-making in diverse linguistic environments.

CHAPTER THREE

3. Research Methodologies

3.1. Introduction

This section highlights the significance of methodologies in conducting rigorous research, explores the key components of research methodologies, and emphasizes the importance of their appropriate selection and implementation. By following sound research methodologies, researchers can contribute to the advancement of sentiment analysis using deep learning and foster more accurate and insightful sentiment analysis applications.

3.2. Research Design

In this study we employed an experimental approach, systematically designing and conducting experiments to investigate a specific phenomenon or problem. We made decisions regarding the number of experiments, algorithms, parameters, weights, and datasets to be used. These decisions had a significant impact on the research direction and outcomes. Experimental research involves repetitive testing and evaluation to test hypotheses and draw reliable conclusions. We conducted a comprehensive literature review, reviewing various sources to inform their research objectives and identify gaps in existing knowledge. We undertook tasks such as data preparation, feature selection, model selection, and selecting evaluation metrics to conduct a thorough experiment. These tasks laid the foundation for analyzing data and drawing meaningful conclusions.

3.3. Literature Review

Performing a thorough examination of journals and articles sourced from various platforms including ACL, Science Direct, Research Gate, Sci-Hub, Scopus Index, and several others was instrumental in uncovering both gaps and trends present in the current literature. Through this comprehensive examination, insights were gained into the current state of research in sentiment analysis, particularly in the context of code-mixed text, prevalent in multilingual societies and digital platforms. By analyzing methodologies, findings, and limitations of prior studies, notable gaps emerged, revealing a need for more robust approaches to sentiment classification, especially in multiclass scenarios. It became evident that existing research primarily focused on binary sentiment classification or monolingual

text, overlooking the complexities of sentiment analysis in code-mixed environments. This identified gap paved the way for the formulation of the research title "multiclass sentiment classification for Amharic-English code-mixed texts," aiming to develop innovative methods to address the nuanced challenges inherent in sentiment analysis of code-mixed texts with multiple sentiment classes.

3.4. Building Dataset

To construct a rich and comprehensive dataset for sentiment analysis, a meticulous multi-channel approach was implemented. Initially, we obtained a dataset comprising 9,500 labeled instances from Seid Muhie Yimam's GitHub repository. These data were meticulously curated from Twitter by Hizkiel Alemayehu and Seid Muhie Yimam, where different individuals assigned sentiment labels. The sentiments were categorized into positive, negative, mixed, or neutral categories. However, for the specific objectives of our study, we focused solely on instances labeled as positive, negative, and neutral, excluding mixed sentiments to enhance clarity and maintain focus on distinct sentiment categories. This selective approach led to a reduction in the dataset size, prompting the need for additional data. To address this, we extracted 2,500 comments from various Facebook pages using a specialized comment exporter tool. In order to collect a diverse set of comments, we targeted the Facebook pages of political parties, activist groups, and public figures with large followings. Subsequently, these comments underwent manual annotation by three independent annotators, who meticulously assigned sentiment labels to each comment, categorizing them as positive, negative, or neutral based on their individual assessments. The manually annotated data was then seamlessly integrated with the existing dataset, resulting in a robust dataset ideal for training sentiment analysis models. In total, the amalgamated dataset comprised 8,819 instances, showcasing a diverse spectrum of sentiments accurately reflected in the comments. This diverse dataset served as a solid foundation for the development and evaluation of precise sentiment analysis algorithms. Following the manual annotation process, rigorous preprocessing steps were employed to ensure the data's quality and suitability for analysis. These preprocessing steps involved meticulously removing irrelevant information, and standardizing comment formats to ensure consistency. Subsequently, the dataset underwent thorough preparation for further analysis and was utilized for training sentiment analysis models, paving the way for insightful exploration and

accurate sentiment analysis outcomes. The objective of this phase is to annotate each comments sentence level into positive, negative, and neutral sentiment classes.

Table 5. Dataset and Sentiment Classes

Sentiment Class	**Count**	**Total**
Positive	4194	8819
Negative	2442	
neutral	2183	

3.5. Preprocessing Techniques

Preprocessing plays a vital role in preparing social media texts suitable for sentiment analysis. It involves several steps to handle the mixture of languages found in such texts. The following methods are applied during the preprocessing stages: cleaning(includes removing irrelevant characters, punctuation, symbols, emails, usernames, blank values, whitespace, URLs, and emoji's; removing elongation from the comments; language identification, handling code-switching), conducting text normalization; and performing tokenization to extract individual words or tokens from the text and removal of stopwords.

3.5.1. Data Cleaning

Data cleaning for sequential text data involves a thorough set of procedures aimed at refining and preparing the data for subsequent analytical or modeling tasks. This iterative process adopts a systematic approach to correct and standardize the text, ensuring its quality and suitability for downstream operations. Essential aspects of data cleaning include removing special characters such as symbols and non-alphanumeric characters, as well as filtering out user names, punctuation marks, URLs, user mentions, and HTML tags. These elements add no value to the text's meaning and can introduce noise to the dataset [21]. By systematically addressing these components, data cleaning enhances the integrity and coherence of the sequential text data, enabling more precise and insightful analyses. Integrating language detection and code-switching handling within the data cleaning process ensures that the text is appropriately segmented and processed based on the languages present, further improving the quality of the data for analysis. Language detection and code-switching handling are essential components of data cleaning for sequential text data. Language detection identifies

the languages present in the text, enabling the application of language-specific preprocessing techniques and models tailored to each language. Similarly, code-switching handling identifies points in the text where there is a switch between languages and segments the text accordingly. Integrating language detection and code-switching handling within the data cleaning process ensures that each language segment is appropriately processed.

3.5.2. Normalization

Text mining techniques encompass a variety of steps aimed at achieving text normalization; these steps include converting characters to their base form, removing diacritics, and managing special symbols or characters. In Amharic and English text normalization, character normalization addresses variations caused by ligatures and orthographic conventions by converting characters to their base forms, ensuring consistency across different representations. Additionally, case normalization ensures uniformity by converting text to a consistent case, such as lowercase or uppercase, mitigating issues arising from case sensitivity during processing. In the Amharic writing system, the inconsistent usage of homophones, characters with similar sounds presents challenges in identifying unique words crucial for discerning between positive, negative, or neutral sentiments. For instance, the word 'ሰላም' can also be written as 'ሠላም' without altering its meaning, leading to an increase in feature size. Thus, normalization becomes imperative to prevent redundant representations of words in different formats.

3.5.3. Stopwords Removal

Stopwords are commonly occurring words in natural language text that have little semantic significance. They typically consist of function words such as articles, conjunctions, prepositions, and pronouns. Examples of stopwords include words like "the," "a," "an," "and," "but," "or," "in," "on," "at," "I," "you," "he," "she," as well as their Amharic equivalents like "እኔ," "አንተ," "እሷ." These words are found extensively in texts but contribute minimally to the overall meaning or context of the text. In tasks like sentiment analysis in natural language processing stopwords are often removed during the preprocessing phase. This removal simplifies the analysis process by focusing on words that carry more substantial meaning and are essential for determining sentiment. By eliminating stopwords, computational resources required for analysis are reduced, and the accuracy of sentiment

analysis models can be enhanced by ensuring that the emphasis is on meaningful content rather than common linguistic structures.

3.5.4. Tokenization

In the process of natural language processing, after obtaining the corpus, which is essentially a collection of text data, the subsequent fundamental step involves segmenting this text into meaningful units that the computer can understand and process. This segmentation process is known as tokenization. Tokenization breaks down the text into smaller components, such as individual words or sub-words, depending on the chosen tokenization strategy. Each of these components, or tokens, becomes a unit of analysis, forming the basis for various NLP tasks like sentiment analysis, named entity recognition, and machine translation. Additionally, tokenization is crucial for constructing the vocabulary necessary for building the embedding matrix, which represents the textual data in a numerical format, that machine learning algorithms can work with effectively. Therefore, tokenization serves as a foundational step in NLP pipelines, facilitating the transformation of raw text into a format suitable for computational analysis and modeling.

3.6. Data Balancing Technique

Imbalanced datasets in sentiment analysis can detrimentally affect model performance, leading to biased predictions that favor the majority class. To counter this issue, techniques such as undersampling, oversampling, and SMOTE can be employed. SMOTE stands out due to its effectiveness in handling class imbalances by generating synthetic samples of the minority class rather than duplicating existing ones, thereby enhancing its representation in the dataset. Unlike under-sampling methods that discard samples from the majority class, SMOTE allows the model to learn more generalized patterns and reduces the risk of overfitting while preserving the complete dataset information. By augmenting the representation of the minority class, SMOTE empowers the sentiment analysis model to better grasp and predict these critical insights. Its compatibility with various machine learning algorithms, including deep learning models, renders SMOTE a versatile and potent technique for achieving balanced and precise sentiment classification. By tackling class imbalance, SMOTE helps alleviate bias during model training, ultimately yielding fairer and more accurate predictions.

3.7. Feature Extraction Techniques

To enable deep learning models to process textual data, vectorization techniques play a vital role in converting it into numerical vectors. In the domain of multiclass sentiment analysis, this study utilizes a combination of TF-IDF and Count Vector techniques, incorporating both unigram and trigram features. These chosen vectorization methods effectively transform textual data into a format that deep learning models can readily comprehend, facilitating efficient analysis. By integrating TF-IDF and Count Vector with an n-gram approach, specifically combining unigram and trigram features, the process of sentiment classification becomes streamlined, enabling the accurate interpretation and categorization of sentiment across multiple classes.

In the case of TF-IDF, an n-gram range of 1 to 3 features is employed, assigning weights to words based on their occurrence within a document and inversely to their frequency across the entire corpus[46]. This technique takes into account sequences of n words and gives priority to unique and less common terms, capturing their significance in sentiment analysis. On the other hand, Count Vector with n-gram representation represents each document as a vector by counting the frequency of individual words or sequences of n words. This method provides a clear representation through a word frequency matrix, facilitating the efficient processing of data by deep learning models.

3.8. Deep Learning Models

After undergoing preprocessing and conversion to numerical format using Count and TF-IDF Vectorizer, the dataset is inputted into multiple deep learning classification models to assess their performance. To classify code-mixed texts into positive, negative, and neutral sentiment categories, various deep learning models such as CNN, LSTM, BILSTM, and CNN-BILSTM are employed. To achieve precise sentiment classification, these models are trained with labeled data and optimization methods. Researchers can select the most suitable model for their sentiment analysis studies and enhance classification results by understanding the workings and success rates of these models.

3.8.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have garnered widespread acclaim for their proficiency in capturing intricate patterns and nuances within textual data, rendering them

exceptionally adept for tasks such as sentiment analysis. The rationale behind choosing a CNN model for multiclass text classification of code-mixed texts lies in its innate ability to discern fine-grained features and contextual cues embedded within linguistic expressions. In the realm of code-mixed texts, where language alternation and hybrid linguistic forms prevail, CNNs offer a robust framework for information extraction and representation learning.

In the context of code-mixed texts, CNNs operate by leveraging convolutional layers to extract salient features from the input data. These convolutional layers serve to detect local patterns and linguistic structures, thereby facilitating the identification of language-specific cues and sentiment indicators. Subsequently, pooling layers are employed to condense the extracted features, effectively capturing the essence of the text while mitigating the impact of noise and irrelevant information. Finally, fully connected layers enable the CNN model to perform classification tasks by synthesizing the learned representations and making predictions across multiple sentiment classes.

One of the key strengths of CNNs lies in their capacity to develop layered representations of textual data, wherein each layer extracts increasingly abstract and informative features. This hierarchical representation allows CNNs to capture both low-level linguistic attributes, such as word sequences and syntactic structures, as well as high-level semantic associations and sentiment expressions. By iteratively refining these representations through the network's architecture, CNNs can discern subtle sentiment cues and discern sentiment orientations within code-mixed texts.

3.8.2. Long Short-Term Memory

Long Short-Term Memory (LSTM) networks, a specialized form of recurrent neural networks (RNNs), are particularly well-suited for sentiment analysis in code-mixed texts. LSTMs excel at capturing long-range dependencies and temporal patterns in sequential data[47], which is crucial for understanding the context and preserving information across code-mixed texts where the alternation between languages and the continuity of sentiments span multiple sentences or clauses.

The unique architecture of LSTMs, which includes mechanisms to handle the vanishing gradient problem common in traditional RNNs, enables them to process sequential data effectively. LSTM networks utilize memory cells with gates that control the flow of information, allowing them to retain relevant information and forget irrelevant data over long sequences. This capability is essential for maintaining a coherent understanding of the sentiment flow in code-mixed texts, where the complex linguistic structures and context often span across multiple tokens.

By leveraging these mechanisms, LSTMs are well-equipped to handle the intricacies of code-mixed texts, where maintaining context and understanding the temporal relationship between words are vital for accurate sentiment classification. The ability of LSTM networks to remember important information and discard irrelevant details over long sequences provides a robust framework for extracting meaningful sentiment cues from code-mixed data, leading to more precise and reliable sentiment analysis.

3.8.3. Bidirectional Long Short-Term Memory

Bidirectional Long Short-Term Memory models enhance traditional LSTM networks by processing input sequences in both forward and backward directions. This bidirectional approach captures context from both past and future tokens[48], which is crucial for sentiment analysis, especially in code-mixed texts.

In code-mixed texts, sentiments and contextual cues can span across different parts of a sentence or multiple sentences, and the alternation between languages adds complexity. BiLSTM models address this challenge by simultaneously processing sequences from start to end and end to start, capturing the full context and providing a richer representation by concatenating hidden states from both directions.

This bidirectional processing is particularly effective for capturing the complex dependencies in code-mixed texts, where sentiment expressions can be influenced by the interplay between languages, cultural nuances, and overall context. By incorporating context from both directions, BiLSTMs can better understand and classify sentiments, even in the presence of linguistic alternations and mixed-language constructs.

Additionally, BiLSTM models can be trained with labeled data using optimization techniques like backpropagation through time (BPTT) and gradient descent, ensuring accurate representation and classification of sentiments in code-mixed texts.

3.8.4. CNN-BILSTM

CNN-BiLSTM models combine the strengths of Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory (BiLSTM) networks, making them well-suited for the complex task of sentiment analysis in code-mixed texts. This hybrid approach leverages the feature extraction capabilities of CNNs and the context-capturing abilities of BiLSTMs, effectively addressing both local and global patterns within the text. Code-mixed texts, which involve the alternation of languages within a single discourse, pose unique challenges for sentiment analysis due to their intricate linguistic structures and nuanced contextual dependencies. CNN-BiLSTM models are well-suited for this task, as they can integrate the complementary strengths of CNNs and BiLSTMs.

CNNs are highly effective at capturing local patterns and features within the text, such as specific word sequences, n-grams, and syntactic structures. By applying convolutional layers to the input text data, CNNs can detect and extract salient features crucial for sentiment classification, providing a detailed representation of the text at a granular level. The features extracted by the CNN layers are then passed on to the BiLSTM layers, which excel at capturing long-range dependencies and contextual information by processing the text in both forward and backward directions. This bidirectional processing allows the model to understand the context from both past and future tokens, which is essential for accurately interpreting sentiment in code-mixed texts, where the sentiment can be influenced by elements spread across the entire text.

To further enhance the effectiveness of sentiment analysis in code-mixed texts, various preprocessing techniques are employed, including language detection and handling code-switching. Language detection helps the model to distinguish between different languages within the text, ensuring that features are accurately extracted for each language. Code-switching techniques enable the model to handle the seamless alternation between languages, maintaining contextual integrity and improving sentiment classification accuracy.

3.9. Model Evaluation

Assessing the model's ability to generalize to new, out-of-sample data is particularly vital. Two common approaches to model evaluation exist. The first is the holdout method, which involves testing the model on data it hasn't seen during training. While simple and fast, this method may lead to variability in accuracy estimates due to differences among training, validation, and test sets. The second approach, widely used and preferred, is k-Fold Cross-Validation. This method provides less biased and more stable estimates of performance, making it suitable for assessing model proficiency. Therefore, k-Fold Cross-Validation is chosen as the model evaluation method in this study.

CHAPTER FOUR

4. Proposed Multiclass Sentiment Detection for Amharic-English Code-Mixed Languages

4.1. Introduction

This chapter focuses on the design of the proposed model for multiclass sentiment analysis of code-mixed social media texts. It encompasses various stages such as data preprocessing, feature extraction, model training, and model evaluation. The details of each step will be discussed and elaborated upon in this chapter.

4.2. Architecture of the Proposed Multiclass Sentiment Detection Model

This section presents the architectural framework of our proposed model designed for sentiment analysis of code-mixed texts. It encompasses a process consisting of data preprocessing, dataset splitting, feature extraction, model training, optimization, and model evaluation.

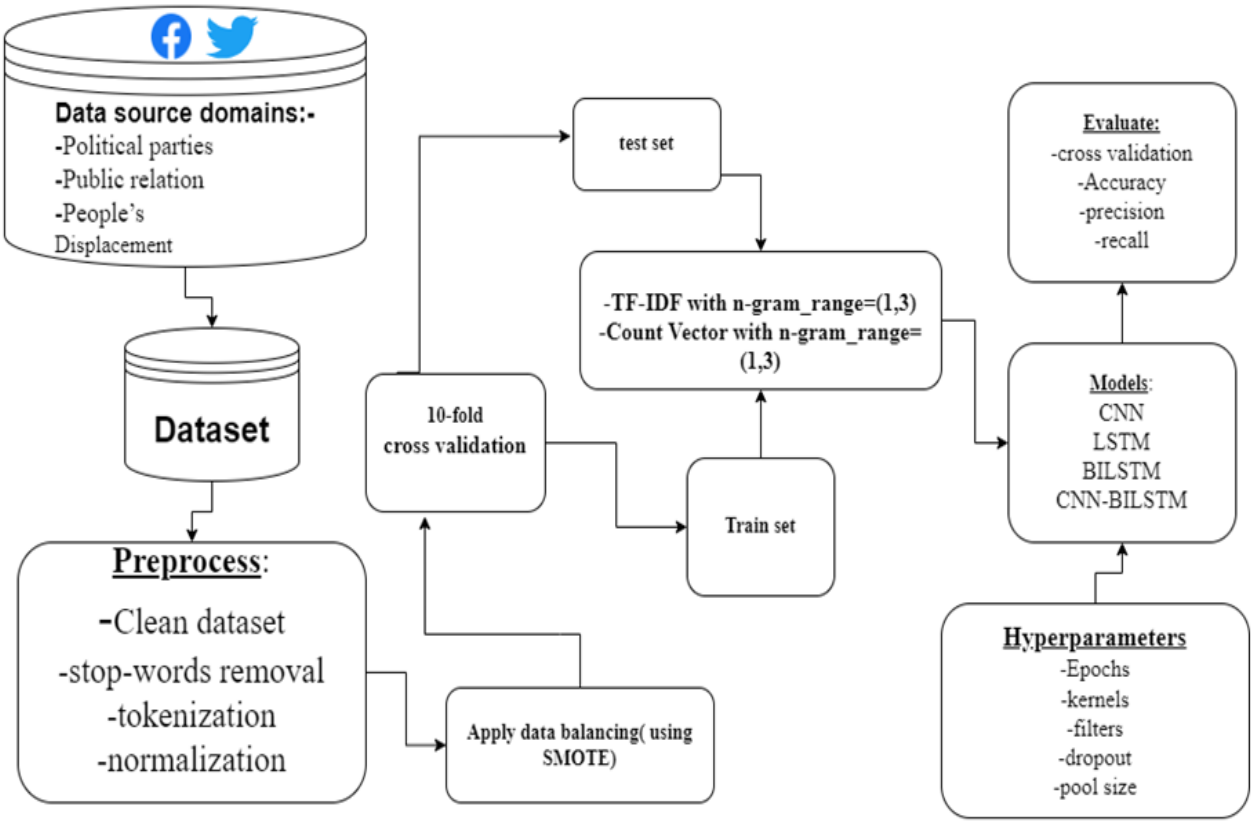


Figure 5. Architecture of Proposed multiclass code-mixed sentiment detection model

The overview of the provided model offers a detailed understanding of its structure and functionality. Specifically, the input to our model comprises of code-mixed text sourced from social media platforms. The architectural framework begins with data preprocessing, an essential step where the raw input data undergoes various transformations to ensure its suitability for analysis. This involves tasks such as language detection, code-switching, tokenization, normalization, and removal of noise or irrelevant information. Following data preprocessing, feature extraction is performed to derive meaningful representations from the preprocessed text. This step involves techniques such as TF-IDF, Count Vectorizer, n-grams to capture the linguistic characteristics and contextual information present in the code-mixed text.

Subsequently, the model undergoes training using the extracted features to learn the underlying patterns and relationships between the input text and corresponding sentiment classes. Various deep learning models like CNN, LSTM, BILSTM, and CNN-BILSTM can be employed for this purpose. Once the model is trained, it undertakes rigorous evaluation to assess its performance in predicting sentiment labels. This evaluation phase includes cross validation(CV) and evaluation metrics such as accuracy, precision, recall, and F1-score, among others, to estimate the model's effectiveness across different sentiment classes.

4.3. Dataset Preprocessing

Preprocessing plays a vital role in preparing social media texts suitable for sentiment analysis. It involves several steps to handle the mixture of languages found in such texts. The following methods are applied during the preprocessing stages: cleaning(includes removing irrelevant characters, punctuation, symbols, emails, usernames, blank values, whitespace, URLs, and emoji’s; removing elongation from the comments; conducting language detection and code-switching); conducting text normalization; and performing tokenization to extract individual words or tokens from the text and removal of stopwords.

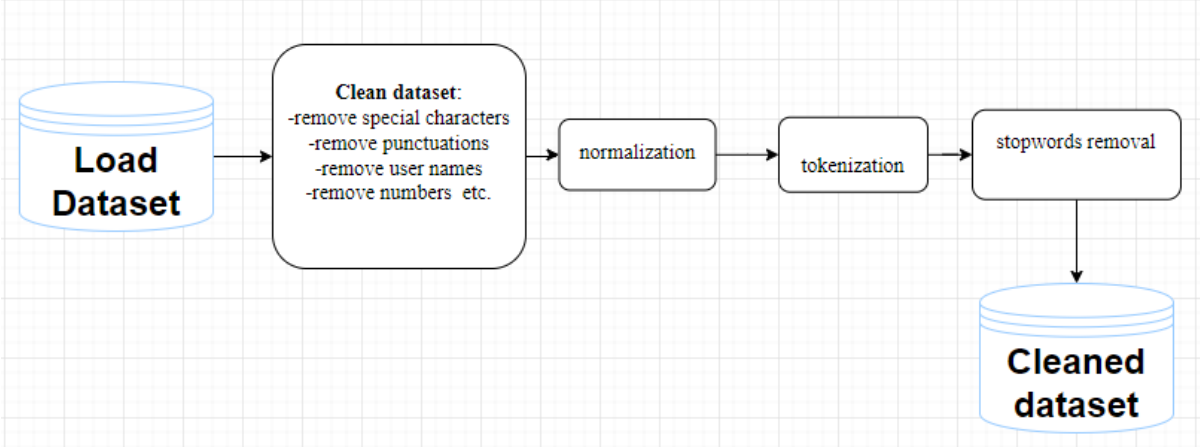


Figure 6. Preprocessing

4.3.1. Data Cleaning

Integrating language detection, code-switching, and text cleaning is crucial for comprehensive text preprocessing. By leveraging sophisticated libraries such as langdetect or polyglot, the predominant language within the text can be determined through advanced algorithms that analyze character, word, or n-gram distribution. This process reduces the potential languages used in the text. For effective language detection in Amharic-English code-mixed texts, the following algorithm can be applied:

Algorithm for Language Detection and Code-Switching in Amharic-English Code-Mixed Texts:

Input: Amharic-English code-mixed text.

Output: Segmented text based on language switches.

1. Initialize an empty list to store segmented text.
2. Include language detection libraries for Amharic and English, such as langdetect.
3. Initialize variables for the previous language and segmented phrase.
4. Start iterating through the code-mixed text.
5. For each word or phrase:
    a. Use the language detection library to determine the language of the current word or phrase.
    b. If the detected language is different from the previous language:
        i. Add the previous segmented phrase to the segmented text list.
        ii. Set the segmented phrase as the current word or phrase.

iii. Update the previous language to the detected language.

c. If the detected language is the same as the previous language: Concatenate the current word or phrase with the segmented phrase.

6. Add the final segmented phrase to the segmented text list.
7. Return the segmented text list.

By incorporating language detection libraries like langdetect, the algorithm remains functionally the same, but with the use of libraries instead of models for language detection. This approach ensures efficient language detection and segmentation of code-mixed text.

Once the text is segmented based on language switches, the next step is text cleaning. This involves eliminating irrelevant characters, punctuation, symbols, blank values, whitespace, and URLs. Python libraries like re and BeautifulSoup provide effective tools for text cleaning, with regular expressions enabling pattern matching and text manipulation. The following algorithm outlines the text cleaning process:

Algorithm for Text Cleaning:

Input: Text data.

Output: Cleaned text data.

1. Initialize an empty list to store cleaned segments.
2. Iterate through each segment of the text.
3. For each segment:
    a. Remove irrelevant characters, punctuation, symbols, etc.: Use regular expressions with the pattern r'[^\w\s]|_' to replace non-alphanumeric characters or underscores with an empty string.
    b. Remove whitespace and blank values: Use regular expressions with the pattern r'\s+' to replace consecutive whitespace characters with a single space.
    c. Remove URLs: Use regular expressions with the pattern r'http\S+' to replace URLs with an empty string.
    d. Remove HTML tags if any: Use regular expressions with the pattern r'<.*?>' to replace HTML tags with an empty string.
    e. Remove numbers: Use regular expressions with the pattern r'\d+' to replace numeric digits with an empty string.

f. Append the cleaned segment to the list of cleaned segments.

4. Return the list of cleaned segments as the cleaned text data.

By integrating language detection, code-switching, and text cleaning, the text data can be effectively preprocessed, ensuring its cleanliness and integrity for subsequent analysis.

4.3.2. Normalization

Text normalization is the process of standardizing and structuring Amharic text data to make it uniform and consistent for analysis and processing. Due to the complex nature of Amharic script and language, normalization is essential to address variations in spelling, character representation, and formatting, ensuring that text data is in a suitable format for natural language processing tasks. Below is an algorithm for normalizing Amharic characters:

```
def normalizeAmharicCharacters(inputString):
    normalizedString = ""
    for character in inputString:
        if character in ['ኀ', 'ሐ']:
            replace with character = 'ሀ'
        elif character == 'ዐ':
            replace with  character = 'አ'
        elif character == 'ሠ':
            replace with character = 'ሰ'
        elif character == 'ጸ':
            replace with character = 'ፀ'
        normalizedString += character
    return normalizedString
```

This algorithm iterates through each character in the input string, replacing specific characters with their normalized counterparts based on the defined rules. After each iteration, whether a character is replaced or not, it is appended to the normalized string. Finally, the normalized string is returned as the output.

4.3.3. Stopwords removal

Removing stopwords is a standard preprocessing step in tasks like sentiment analysis in natural language processing. By removing stopwords, the focus can shift to more informative words in the text, resulting in more precise and effective analyses.

Algorithm to remove stopwords:

Function remove_stopwords (text):

Define English stopwords **as** a **set** of stopwords **from** the nltk corpus **for** the English language

Read Amharic stopwords **from** a file located at specified loacation using pandas

Tokenize the **input** 'text' into words

filtered_words = **[]**

FOR EACH word IN words DO

IF word **is not in** english_stopwords AND word **is not in** amharic_stopwords THEN

ADD word TO filtered_words

END IF

END FOR

cleaned_text = join filtered_words **with** ' '

RETURN cleaned_text

END FUNCTION

4.3.4. Tokenization

Once we have acquired the corpus, the next crucial step is to break down the text into distinct units, such as individual words or sub-words. Tokenization plays a pivotal role in this process by subdividing the text into smaller components, which forms the basis for creating the vocabulary needed for the embedding matrix.

Algorithm for Tokenizing Text:

1. Input: text (string) - the text to be tokenized
2. Initialize an empty list called tokens
3. Tokenize the text using word_tokenize function and store the result in tokens
4. Initialize an empty list called filtered_tokens
5. Iterate through each token in tokens
6. Check if the length of the token is greater than or equal to 2

a. If the condition is met, add the token to filtered_tokens

7. Return the filtered_tokens as the final result of the function.

4.4. Feature Extraction Techniques

Essential methods for converting textual data into numerical vectors suitable for processing by deep learning models include vectorization techniques. In the context of multiclass sentiment analysis, TF-IDF and Count Vector with combination of unigram and tri-gram features are employed for this study. We choose these vectorization techniques for multiclass sentiment classification because they effectively represent textual data in a format that deep learning models can understand and analyze easily. By incorporating TF-IDF and Count Vector with n-gram (in this case combination of unigram and trigram) into deep learning models, sentiment classification becomes efficient, enabling effective understanding and classification of sentiment across multiple classes.

Term Frequency-Inverse Document Frequency (TF-IDF) with n-gram (in this case an n-gram range of 1 to 3 features) assigns weights to words based on their frequency in a document and inversely to their frequency across all documents in the corpus[46], taking into account sequences of n words. This prioritizes unique and less common words, capturing their significance in sentiment analysis. The following pseudocode outlines the process of feature extraction using TF-IDF with combination of unigram and tri-gram for code-mixed texts.

Define "code_mixed_texts" as a list of Amharic-English code-mixed texts.

1. Initialize TF-IDF Vectorizer with ngram_range= (1, 3).
2. Extract features using TF-IDF with combination of unigram and trigram:
   tfidf_features = TF-IDF.fit_transform (code_mixed_texts).
3. Get feature names for TF-IDF with combination of unigram and trigram:
   tfidf_feature_names = TF-IDF.get_feature_names_out ().

On the other hand, Count Vector with n-gram represents each document as a vector with the count of each word or sequence of n words. This technique provides a straightforward word frequency matrix representation, allowing deep learning models to process the data efficiently. The following pseudocode outlines the process of feature extraction using Count

Vector with combination of unigram and tri-gram for code-mixed texts. Define "code_mixed_texts" as a list of Amharic-English code-mixed texts.

1. Initialize Count Vectorizer with ngram_range= (1, 3).
2. Extract features using Count Vectorizer with combination of unigram and trigram: count_features = CountVectorizer.fit_transform (code_mixed_texts).
3. Get feature names for Count Vectorizer with combination of unigram and trigram: count_feature_names = CountVectorizer.get_feature_names_out ().

Feature names obtained for each text in the "code_mixed_texts" list, demonstrating the extraction process effectively.

4.5. Deep Learning Models

After undergoing preprocessing and conversion to numerical format using Count and TF-IDF Vectorizer, the dataset is inputted into multiple deep learning classification models to assess their performance. To classify code-mixed texts into positive, negative, and neutral sentiment categories, various deep learning models such as CNN, LSTM, BILSTM, and CNN-BILSTM are employed. To achieve precise sentiment classification, these models are trained with labeled data and optimization methods. Researchers can select the most suitable model for their sentiment analysis projects and enhance classification results by understanding the workings and success rates of these models.

4.5.1. Convolutional Neural Networks

The Convolutional Neural Network (CNN) model is designed with different layers, including convolutional layers, pooling layers, and dense layers, specifically designed for sentiment classification on code-mixed texts for multiclass classification.

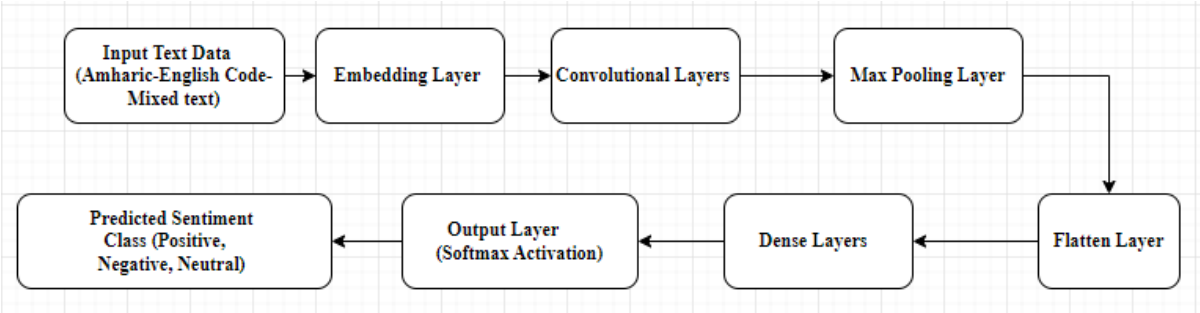


Figure 7. CNN Model workflow

Each layer serves a unique purpose in enhancing the model's ability to understand and classify the sentiment of code-mixed texts. Convolutional layers are responsible for feature extraction and employ the Conv1D layer in this model to perform 1-dimensional convolutions on the input data. Within these layers, the number of filters and kernel size is randomly chosen from predefined ranges, dictating the characteristics of the filters applied to the data. This random selection enables the model to identify diverse patterns and relationships within the text, generating higher-level representations conducive to sentiment classification.

Pooling layers are tasked with information consolidation and utilize the MaxPooling1D layer to down sample the feature maps by selecting maximum values within specified pool sizes. By reducing feature dimensionality while retaining essential information, the max-pooling layer enhances the model's focus on significant features and computational efficiency, addressing the need for efficient feature extraction and retention. A dropout layer is integrated after the pooling layers to combat overfitting by randomly deactivating neurons during training. This randomness is essential in promoting diverse feature utilization and regulating the extent of neuron deactivation, thereby fostering model robustness and generalization to unseen data.

Following the dropout layer, the Flatten layer reshapes the multidimensional feature maps into a single vector, facilitating subsequent processing by dense layers. These dense layers, responsible for sentiment classification, adapt the flattened features to higher-level representations and produce class probabilities through Softmax activation. This sequential processing ensures efficient classification of sentiment in code-mixed texts.

4.5.2. Long Short-Term Memory

The Long Short-Term Memory (LSTM) model is specifically designed for multiclass sentiment classification of code-mixed texts, focusing on its sequential architecture known for effectively capturing temporal dependencies present in sequential data. This model is particularly suitable for analyzing Amharic-English code-mixed social media texts, which undergo preprocessing steps like tokenization and embedding matrix creation to align with the requirements of the LSTM model.

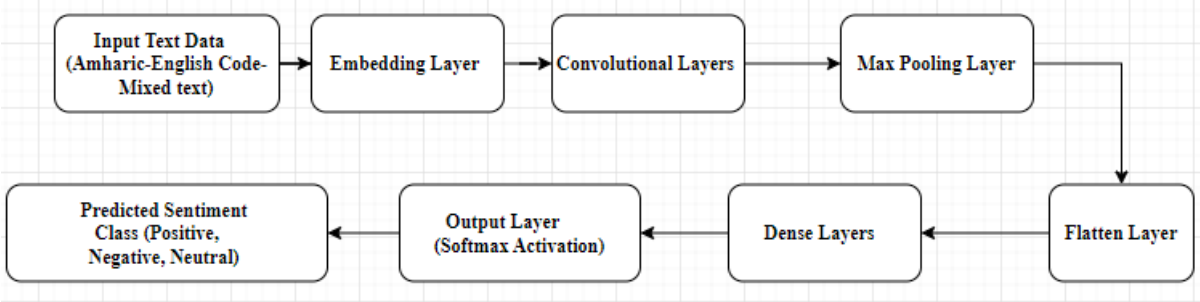


Figure 8. LSTM model workflow

Consist of multiple layers, each with a distinct role in sentiment classification, the LSTM architecture begins with the embedding layer. An Embedding layer is added as the first layer, which converts the input data into dense vectors of fixed size. This layer helps in learning the representation of words in a continuous space. Then following the embedding layer LSTM layer is added. This layer is responsible for capturing sequential information and modeling the long-term dependencies within the code-mixed texts. The number of units in the LSTM layer is randomly selected from a range of 64 to 256, which influences the layer's ability to retain and process temporal information. A higher unit count enables the layer to capture more intricate patterns and dependencies, potentially improving sentiment classification performance. To address overfitting and enhance generalization, a dropout layer is added after the LSTM layer. During training, this layer randomly deactivates a portion of neurons, with the dropout rate chosen randomly from a uniform distribution between 0.4 and 0.6. This regularization technique encourages the model to rely less on specific information and promotes the learning of more robust representations adaptable to unseen data. By randomly dropping neurons, the model diversifies its feature reliance across training iterations, thereby improving generalization performance.

The architecture is finalized by a dense layer with units corresponding to the output units representing the sentiment classes. Using the Softmax activation function, this layer generates a probability distribution across the sentiment classes. The Softmax function ensures that the probabilities sum to one, enabling sentiment assignment based on the highest probability. The dense layer maps the representations learned by the LSTM layer and the features extracted by the dropout layer to sentiment predictions.

4.5.3. Bidirectional Long Short-Term Memory

The Bidirectional Long Short-Term Memory (BiLSTM) model, designed for multiclass sentiment classification of code-mixed texts, operates within a sequential architecture proficient in capturing temporal dependencies inherent in sequential data. This architecture is particularly well-suited for code-mixed texts, known for their complex linguistic structures and cross-lingual dependencies.

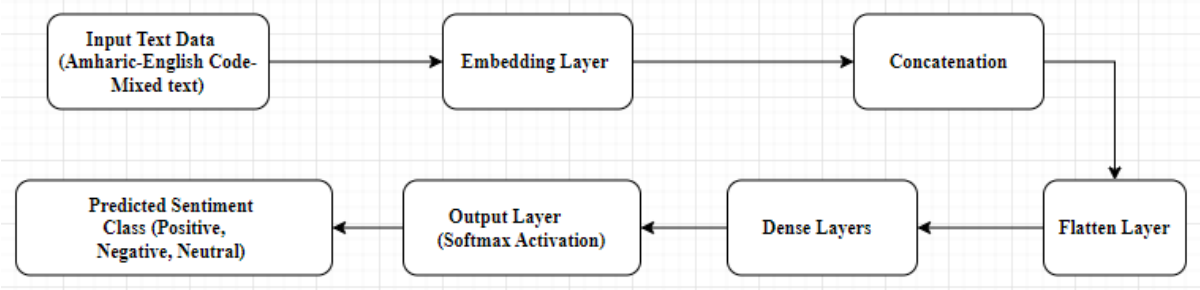


Figure 9. BILSTM model workflow

Initially, starting with input data consisting of Amharic-English code-mixed social media texts, preprocessing steps such as tokenization and embedding creation are applied to ensure compatibility with the BiLSTM model's requirements. Following preprocessing, the data undergoes processing through the BiLSTM model, which comprises bidirectional LSTM cells. Utilizing bidirectional LSTM cells enables the model to understand mixed-language contexts by considering the entire input sequence in both forward and backward directions. This bidirectional capability equips the model to grasp intricate language nuances and dependencies present in code-mixed texts. By processing the input sequence bidirectionally, the model effectively leverages information from both past and future contexts, enriching its understanding of the input data. After constructing the BiLSTM architecture, the model is trained using labeled data of Amharic-English code-mixed social media texts. Throughout training, the model adjusts its parameters to optimize sentiment information capture within the code-mixed texts. Parameters of the BiLSTM layer, particularly the number of units, are randomly selected within the range of 64 to 256, affecting the layer's ability to capture and process temporal information. A higher unit count empowers the layer to distinguish more complex patterns and dependencies, potentially enhancing sentiment classification performance.

To prevent overfitting and enhance generalization, a dropout layer is introduced after the BiLSTM layer. This layer randomly deactivates a portion of neurons during training, with the dropout rate randomly selected from a uniform distribution between 0.4 and 0.6. This regularization strategy encourages the model to rely less on specific information, fostering the learning of more robust representations adaptable to unseen data. Through random neuron dropout, the model diversifies its feature reliance across training iterations, thereby refining generalization performance.

Finally, the model integrates a dense layer with units corresponding to the output units, representing the sentiment classes. Utilizing the Softmax activation function, this layer generates a probability distribution across sentiment classes, ensuring predicted probabilities sum to one and enabling sentiment label assignment based on the highest probability. The Dense layer learns to map representations learned by the BiLSTM layer and features extracted by the dropout layer to sentiment classes, facilitating predictions.

4.5.4. CNN-BILSTM

The CNN-BiLSTM model integrates Convolutional Neural Networks (CNNs) with Bidirectional Long Short-Term Memory (BiLSTM) networks, rendering it highly effective for parsing code-mixed texts with diverse linguistic structures.

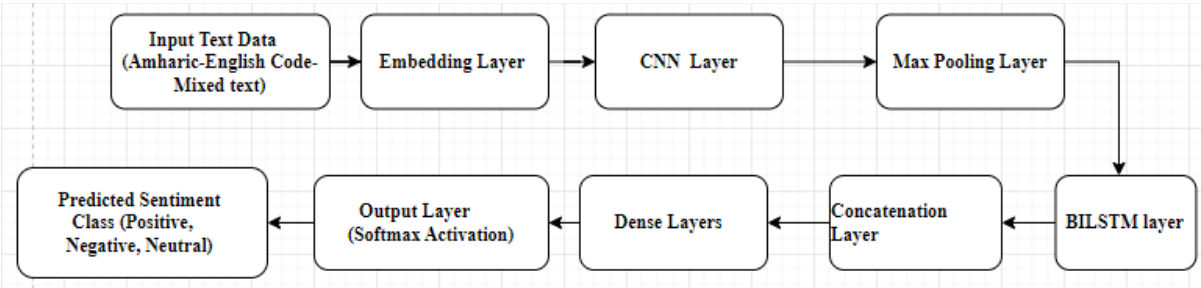


Figure 10. CNN-BILSTM model workflow

Initially, the model begins with an embedding layer, which converts input text words into dense vectors, effectively capturing their semantic meaning and contextual relationships. This embedding process lays the foundation for subsequent processing stages.

Subsequently, the model employs a 1D convolutional layer to process the embedded text, extracting local features and patterns using filters of various sizes. By incorporating filters of different sizes, the model can capture both fine-grained and coarse-grained features, allowing

for pattern recognition at multiple text granularity levels. This adaptability enables the identification of crucial local insights contributing to sentiment analysis. Following the convolutional layer, the model introduces a max-pooling layer, which down samples features maps by selecting maximum values within each pooling window. This step preserves essential features while reducing feature map dimensionality, thereby decreasing computational complexity and preventing overfitting by focusing on relevant information.

The output from the pooling layer then feeds into a Bidirectional LSTM (BiLSTM) layer. This layer processes input sequences simultaneously in both forward and backward directions, capturing long-range dependencies and comprehending the text sequentially. By considering the entire input sequence bidirectionally, the BiLSTM layer adeptly captures complex linguistic structures and dependencies within code-mixed texts. To mitigate overfitting and enhance generalization, dropout layers are incorporated after the convolutional and BiLSTM layers. These layers randomly deactivate neurons during training, forcing the model to rely on diverse feature combinations and reducing dependence on specific information. This regularization technique fosters robust learning and improves the model's ability to generalize to unseen data.

Finally, the output from the BiLSTM layer undergoes processing through a dense layer with Softmax activation. This layer facilitates multiclass classification by computing sentiment probabilities using the Softmax function, ensuring that the summed probabilities equal one and allowing for sentiment label assignment based on the highest probability. The dense layer maps the learned representations from the BiLSTM layer to sentiment classes, enabling predictions.

4.6. Model Evaluation

Evaluating machine learning models through cross-validation is a widely adopted technique. Cross-validation helps to obtain a more reliable assessment of a model's performance by partitioning the dataset into multiple subsets and then evaluating the model's performance on each of these subsets. This approach ensures a more comprehensive evaluation, as it mitigates the potential biases and uncertainties that may arise from using a single train-test split.

4.6.1. Cross Validation

During k-fold cross-validation, the dataset undergoes division into k equal-sized groups or folds. Each fold serves once as the validation set, while the remaining k-1 folds train the model. The parameter k determines the number of folds in the partitioning process. The process of k-fold cross-validation unfolds as follows:

1) Data Partitioning: The dataset is randomly split into k folds to ensure fairness and representation across all subsets.
2) Training and Validation: In each cross-validation iteration:
   - One fold acts as the validation set.
   - The remaining k-1 folds constitute the training set.
   - The model trains on the training set to unveil underlying patterns and relationships within the data.
3) Model Evaluation: Post-training, the model undergoes assessment using the designated validation set to gauge its performance on unseen data.
4) Iteration: Steps 2 and 3 repeat k times, with each fold serving as the validation set exactly once, ensuring every data point participates in the validation process across all iterations.
5) Performance Summary: After all iterations, performance metrics (e.g., accuracy, precision, recall) from each fold's validation process are typically averaged to provide an overall evaluation of the model's performance.

K-fold cross-validation systematically assesses the performance of a model by rotating the validation set across each fold and aggregating the outcomes. This approach offers a more reliable estimation of the model's performance compared to a simple percentage split. By minimizing variability resulting from specific choices of training and test sets, it provides more trustworthy evaluations of the model's performance and its ability to generalize to unseen data. The evaluation metrics employed in this process provide insights into various aspects of the model's performance for each sentiment class. These metrics, including accuracy, precision, recall, and F1-score, collectively offer a comprehensive understanding of the model's effectiveness in predicting sentiment for each category.

CHAPTER FIVE

5. Experiment

5.1. Experimental Setup

The experimental setup incorporated hardware components that included an Intel(R) Core(TM) i5-5200U CPU operating at a clock speed of 2.20GHz and 8.00GB of RAM. This hardware configuration provided sufficient computational capabilities for executing the model training and evaluation processes efficiently. The software infrastructure was built upon Windows 10 Pro as the operating system, while Python version 3.9 served as the primary programming language. Python's extensive collection of libraries, including NLTK, scikit-learn, TensorFlow, and Keras, played a crucial role in various stages of the experimentation pipeline, facilitating tasks such as data preprocessing, model development, and evaluation.

Text preprocessing tools, such as tokenizers and language identification tools were employed to effectively prepare the dataset for analysis. The chosen model architectures encompassed CNN, LSTM, BILSTM and CNN-BILSTM, each offering unique strengths in capturing intricate patterns within code-mixed text data. The Adam optimization algorithm was utilized to fine-tune the model's parameters and expedite convergence during the training process. Evaluation metrics, such as accuracy, precision, recall, and F1 score and validation curve were meticulously tracked to comprehensively assess the model's performance across multiclass classification tasks. These metrics provided valuable insights into both the overall predictive accuracy and the performance of the model on specific classes, ensuring a comprehensive evaluation of its capabilities.

5.2. Development Tools and Packages

We employed various development tools and packages to develop a proposed model for sentiment classification of code-mixed texts for multiple classes. We conducted implementation for each tasks from data preprocessing to model construction, utilizing the Python programming language, selected for this research due to its widespread adoption among developers, researchers, and data scientists engaged in machine-learning models. Since it has vast array of libraries and frameworks, including Keras, TensorFlow, and Scikit-

learn, established it as the preferred choice for implementing and evaluating deep learning algorithms.

5.2.1. Package Manager

Package managers are essential tools for managing external libraries and dependencies in Python. They simplify the process of installing, upgrading, and uninstalling packages, making it easier for developers to work with different libraries and frameworks. In Python, two popular package managers are pip and conda.

5.2.2. Implementation Environment

In this study, Jupyter Notebook and Google Colab are two popular environments used for sentiment analysis used while implementing using Python. It provides an interactive and user-friendly interface for writing and deploying code in both environments. Jupyter Notebook mostly used for data analysis and machine learning projects, while Google allows free access to Colab GPUs and TPUs, making it suitable for training deep learning models.

5.2.3. Data Processing and Modeling Tools

Deep learning models have become increasingly popular in recent years, revolutionizing various businesses with their ability to extract complex patterns from large datasets. To effectively training these models, it is essential to use robust data processing and modeling tools that rationalize the training process and make precise predictions on unseen user inputs.

5.2.3.1. Data Processing Tools

Data preprocessing plays a vital role in deep learning by converting raw data into a suitable format for model training. We employ the following for processing our dataset.

Table 6. Data preprocessing tools

Tool	Description
Pandas	A versatile Python library offering high-performance data manipulation and analysis capabilities, including functions for cleaning and transforming data.
NumPy	Essential for numerical computing and data manipulation in Python, providing support for multidimensional arrays, matrices, and mathematical functions.

Scikit-learn	A popular machine learning library in Python that includes tools for data preprocessing, feature extraction, and implementing various machine learning algorithms.

5.2.3.2. Modeling Tools

After data preprocessing, the next step in developing deep learning models involves using modeling tools to build and train these models. Tensor Flow stands out as a comprehensive platform offering a range of tools for model development, training, and deployment. Notably, TensorFlow's high-level APIs such as Keras provide a user-friendly interface for constructing and training deep learning models. Moreover, TensorFlow supports distributed computing, enabling seamless scalability of models across multiple GPUs or distributed systems. Another valuable tool is Keras, an open-source neural network library that simplifies the process of building deep learning models. Built on top of TensorFlow and other deep learning frameworks, Keras offers high-level abstractions and pre-defined building blocks, facilitating the experimentation with different architectures and hyperparameters. Its ease of use makes it a popular choice among both beginners and experienced practitioners in the field.

5.3. Preprocessing

In text mining activities, preprocessing plays an essential role in cleaning up the comments and making the dataset ready for feature extraction. Several methods used in the preprocessing process to achieve this goal. By performing these preprocessing tasks, we can ensure that the text data is cleaned, organized, and transformed into a suitable format for sentiment analysis. Initial steps involve text cleaning, which entails removing noise, irrelevant symbols, and handling common challenges such as contractions and emoticons. Tokenization breaks down the text into individual words, while removing stop words eliminates commonly used but non-informative terms. Padding or truncation ensures uniform sequence length, a prerequisite for input to neural networks. Encoding labels, splitting the data into training and testing sets, and addressing class imbalances add the finishing touches to the dataset preprocessing trip, setting the stage for the development of robust sentiment analysis models.

5.3.1. Loading Dataset

The initial step in the preprocessing pipeline involves loading the dataset, which serves as the foundation for subsequent tasks. To accomplish this, we make use of the Pandas library, which provides efficient handling of data through the use of DataFrames. DataFrames offer several advantages, including fast processing capabilities and convenient access to data instances using column names. The dataset is loaded using the read_csv method provided by Pandas, specifically designed for reading comma-separated value files.

```
# Load dataset
import pandas as pd
data = pd.read_csv('.content/drive/final/dataset.csv')

# Split data using cross-validation
X = data['tweet1'].values
y = data['sentiment'].tolist()
```

Figure 11. Implementation of data loading using pandas

After loading the dataset, we can proceed with the preprocessing stage, which involves several key tasks to prepare the data for sentiment analysis. These tasks can be performed in the following subsections.

5.3.2. Cleaning the Data

To preprocess Amharic-English code-mixed texts for analysis, you can employ language detection, code-switching handling, and text cleaning techniques. These steps help prepare the data by segmenting it based on language switches, applying language-specific preprocessing, and removing irrelevant characters, punctuation, symbols, blank values, whitespace, and URLs that may introduce noise into the dataset. Python provides various libraries for data cleaning, including the notable "re" library, short for regular expression, which enables powerful pattern matching and text manipulation. By utilizing regular expressions to define patterns, you can efficiently eliminate special characters, HTML tags, and numbers from the text data.

```
def clean_text(text):
    try:
        # Language detection
        lang = detect(text)
        # Code-switching handling
        if lang == 'am':
            # Remove Ge'ez numbers
            cleaned_text = re.sub(r'[\u1369-\u137C]', '', text)
            # Remove characters that come with numbers
            cleaned_text = re.sub(r'[\u1200-\u137F]+\d+', '', cleaned_text)
        else:
            cleaned_text = text
        # Remove URLs
        text = re.sub(r'http\S+|www.\S+', '', text)
        # Remove usernames (mentions)
        text = re.sub(r'@[^\s]+', '', text)
        # Remove hashtags
        text = re.sub(r'#([^\s]+)', '', text)
        # Remove special characters, punctuation, and emojis
        text = re.sub(r'[^\w\s]|_+', '', text)
        # Remove numbers
        text = re.sub(r'\d+', '', text)
        return cleaned_text
    except LangDetectException:
        return text
```

Figure 12. Implementation for text cleaning

Each particular step in this processing code significantly contributes to purifying the text data, ensuring its readiness for subsequent analysis or modeling tasks. These techniques help to clean and standardize the text data, making it easier for deep learning models to extract meaningful patterns and sentiments.

5.3.3. Normalization

To normalize Amharic text within the Python working environment, the replace function offers unique role for handling homophone characters replacing

```
import re
def replace_multiple(main_string, to_be_replaced, new_string):
    for elem in to_be_replaced:
        if elem in main_string:
            main_string = main_string.replace(elem, new_string)
    return main_string

def normalize_char_level_missmatch(input_token):
    replacements = {
        'ሃ|ኅ|ኃ|ሐ|ኻ': 'ሀ',
        'ሑ|ኁ|ዅ': 'ሁ',
        'ሒ|ኂ|ኺ': 'ሂ',
        'ኌ|ሔ|ኼ': 'ሄ',
        'ሕ|ኅ': 'ህ',
        'ኆ|ሖ|ኾ': 'ሆ',
        'ዓ|ኣ|ዐ': 'አ',
        'ሠ|ሡ|ሢ|ሣ|ሤ|ሥ|ሦ': 'ሰ|ሱ|ሲ|ሳ|ሴ|ስ|ሶ',
        'ጸ|ጹ|ጺ|ጻ|ጼ|ጽ|ጾ': 'ፀ|ፁ|ፂ|ፃ|ፄ|ፅ|ፆ',
        'ዑ|ዒ|ዔ|ዕ|ዖ': 'ኡ|ኢ|ኤ|እ|ኦ',
    }

    for pattern, replacement in replacements.items():
        input_token = re.sub(pattern, replacement, input_token)

    return input_token
```

Figure 13. Implementation for Normalization of Amharic characters

After each iteration, whether a character is replaced or not, it is appended to the normalized string. Finally, the normalized string is returned as the output. In the context of the provided code, normalization is used to handle character-level mismatches in the input token. Character-level mismatches can occur due to different input methods, or variations in spelling conventions. For example, in the Amharic language, different characters may represent the same sound or phoneme. By normalizing these character-level mismatches, the code ensures consistency and allows for accurate processing and analysis of the text. Normalization helps in various NLP tasks, such as, sentiment analysis. It ensures that different forms of the same word or character are treated as equivalent, reducing ambiguity; reduce token size and improving the accuracy and efficiency of subsequent processing steps.

5.3.4. Tokenization

Once we have acquired the preprocessed corpus, the next essential step is to tokenize the text into distinct units, such as individual words or sub-words. Tokenization plays a pivotal role in this process by subdividing the text into smaller components, which forms the basis for creating the vocabulary needed for the embedding matrix. We employed the Python module nltk for obtaining tokens or words from the text present in the dataset. To achieve this, we utilized the split function in Python and the words tokenize method provided by nltk. The resulting tokens serve as the input for feature extraction methods.

```
def tokenize_text(text):
    tokens = word_tokenize(text)
    filtered_tokens = [token for token in tokens if len(token) >= 2]
    return filtered_tokens
```

Figure 14. Implementation for Tokenization

5.3.5. Stopwords Removal

Stopwords are frequently occurring common words that are typically disregarded during the analysis of natural language text. These words hold minimal significance in understanding the text's meaning, as they are used frequently and do not add much to the overall context. Removing stopwords is a standard preprocessing step in tasks like sentiment analysis in natural language processing. By removing stopwords, the focus can shift to more informative words in the text, resulting in more precise and effective analyses. We provide sample code for removing stopwords as below.

```
import nltk
import pandas as pd

def remove_stopwords(text):
    # Define English stopwords
    english_stopwords = set(nltk.corpus.stopwords.words('english'))

    # Read Amharic stopwords from a file
    amharic_stopwords = pd.read_csv('/content/gdrive/My Drive/final/stopwords.txt')

    # Tokenize the text into words
    words = text.split()

    # Remove stopwords from both languages
    filtered_words = [word for word in words if word not in english_stopwords
                      and word not in amharic_stopwords]

    # Join the filtered words back into a text
    cleaned_text = ' '.join(filtered_words)

    return cleaned_text
```

Figure 15. Implementation for Stopwords removal

By removing stopwords, the focus can shift to more informative words in the text, resulting in more precise and effective analyses.

5.4. Data Balancing

Following the preprocessing of the unstructured dataset, achieving balance among the sentiment classes becomes imperative. In this context, we opt to utilize the Synthetic Minority Over-sampling Technique (SMOTE) as our data balancing method. In our dataset, SMOTE is applied to augment the minority classes (2 and 1) to match the abundance of the majority class (0). The resulting balanced dataset exhibits equal representation across all classes, with 4194 instances for each class. SMOTE achieves this balance by creating synthetic samples through interpolation within the feature space, effectively amplifying the representation of the minority classes. Here's the Python script used to balance the unbalanced sentiment classes using SMOTE:

```
# Separate features and target variable
X = df.drop('tweet', axis=1)
y = df['sentiment']

# Apply SMOTE to balance the classes
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)
```

Table 7. Balanced Sentiment Classes

Sentiment Class	Count
Positive	4194
Negative	4194
neutral	4194

After applying the SMOTE balancing technique, the data is now in a balanced state.

5.5. Feature Extraction

Before training our model, feature extraction is an essential step to represent the different features of the data into numerical form, in which our model will learn from, many techniques can be used, but in this case we have chosen use TF-IDF and Count Vectorizer with combination of unigram and tri-gram features. We conducted experiments with a variety of feature extraction techniques, and we will provide an overview of some of them below.

Feature extraction techniques like TF-IDF with n-gram and Count Vectorizer with n-gram play a crucial role in multiclass sentiment classification of code-mixed social media texts, particularly when employing deep learning models such as CNN, LSTM, BILSTM, and CNN-BILSTM. TF-IDF computes weights for words based on their frequency and inverse frequency, enabling it to efficiently capture individual words and n-grams.

```
# TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 3))
X_train_tfidf = tfidf_vectorizer.fit_transform(data['tweet1'])
X_test_tfidf = tfidf_vectorizer.transform(data['tweet1'])
```

Figure 16. Implementation for TF-IDF vectorization

Conversely, Count Vectorization tabulates word frequencies, thus boosting sentiment analysis across various sentiment classes.

```
# Count Vectorizer with (unigram + trigram)
count_vectorizer = CountVectorizer(ngram_range=(1, 3))
X_train_count = count_vectorizer.fit_transform(data['tweet1'])
X_test_count = count_vectorizer.transform(data['tweet1'])
```

Figure 17. Implementation for Count vector

These methods apply frequency-based weights and distributed word representations to extract meaningful features from code-mixed texts, leading to enhanced performance in sentiment analysis and other NLP activities. We provide below a sample code for extraction using TF-IDF with n-gram and Count Vector with n-gram for Amharic-English code-mixed texts.

5.6. Model Implementation

After obtaining pre-processed and vectored data required for training the deep learning model, we are now able to implement all the considered models in this study. The implementation of the model for multiclass sentiment analysis of social media texts involves training various deep learning architectures, including CNN, LSTM, BILSTM, and CNN-BILSTM models. We design each model to capture different linguistic features and temporal dependencies essential in code-mixed texts. During the implementation stage, we configure the deep learning models with specific hyperparameters, such as the number of layers,

neurons, activation functions, loss functions, optimization algorithms, the number of filters, kernel size, pool size, dropout rate, and number of units. In our study, we have identified the most critical hyperparameters and their optimal values, which are detailed in the table below.

Table 8. Lists of parameters used in modeling

Hyperparameters	**Optimal Value Range**
Filters	16-64
Kernel Size	[3, 5, 7]
Pool Size	[2, 3]
Dropout Rate	0.4 - 0.6
Dense Units	32-128
LSTM Units	64-256
Output Units	32-128
Loss Function	['sparse_categorical_crossentropy', 'categorical_crossentropy']
Optimizer	adam
Patience (Early Stopping)	2-6

The model architecture used in this study is designed to effectively classify sentiment and takes into consideration various parameters and their corresponding values. These parameters are carefully configured within the model to enhance its performance and address issues such as overfitting, which can negatively impact model accuracy. By fine-tuning the parameters, the model aims to strike a balance between capturing important features and avoiding unnecessary complexity. This helps prevent overfitting, where the model becomes too specialized to the training data and performs poorly on unseen data.

The analysis of the model summaries highlights various architectural approaches and parameter configurations, offering insights into how the models process and learn from textual data for accurate classification tasks. Based on the specified parameters, we have designed several model architectures. Each architecture is crafted to capture different aspects of the input text data, employing a combination of convolutional, recurrent, and pooling layers, along with dropout regularization, to effectively extract features and classify sentiments.

5.6.1. Convolutional Neural Network

The Python implementation provided below represents a high-level CNN architecture for sentiment classification. Keras, a popular deep learning library, is utilized with a TensorFlow backend to build and train the model. TensorFlow provides efficient computation and optimization capabilities, making it a powerful framework for deep learning tasks.

```
"CNN": [
        Embedding(input_dim=vocab_size,output_dim=embedding_dim,
                  input_length=max_sequence_length),
        Conv1D(filters=random.randint(16, 64),
                  kernel_size=random.choice([3, 5, 7])),
        MaxPooling1D(pool_size=random.choice([2, 3])),
        Dropout(random.uniform(0.4, 0.6)),
        Flatten(),
        Dense(units=random.randint(32, 128)),
        Dense(units=output_units, activation='softmax')
        ]
```

Figure 18. CNN Model implementation

The implementation takes into account the parameters and values mentioned earlier, which are set to appropriate values based on the specific requirements of the study. These parameter configurations are crucial for achieving optimal model performance and mitigating potential issues that could hinder accuracy.

The CNN model for text classification consists of an Embedding layer to convert input text data into dense vectors, followed by Conv1D layers to extract features using convolutional filters. The GlobalMaxPooling1D layer aggregates the features, and dense layers learn patterns for classification. The model's parameters, including filter numbers, kernel sizes, pool sizes, dropout rate, and dense layer units, are randomly selected within predefined

ranges to allow for exploration of various configurations. This exploration aids in optimizing model performance by adapting to different text patterns and variations, ultimately enhancing sentiment classification accuracy in code-mixed texts. . Based on the above-mentioned parameter values, we have the following model architectures.

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_9 (Embedding)     (None, 147, 100)          1065900

 conv1d_12 (Conv1D)          (None, 143, 35)           17535

 max_pooling1d_12 (MaxPooli  (None, 71, 35)            0
 ng1D)

 dropout_20 (Dropout)        (None, 71, 35)            0

 flatten_4 (Flatten)         (None, 2485)              0

 dense_33 (Dense)            (None, 41)                101926

 dense_34 (Dense)            (None, 3)                 126

=================================================================
Total params: 1185487 (4.52 MB)
Trainable params: 1185487 (4.52 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 19. Model summary for CNN model

The CNN model architecture starts with a Conv1D layer that applies random filters within the range of 16 to 64 and a kernel size randomly selected from [3, 5, and 7]. This layer performs convolutions on the input text, followed by a MaxPooling1D layer that randomly selects a pool size of 2 or 3 to down sample the outputs. A Dropout layer with a random dropout rate between 0.4 and 0.6 is then applied to prevent overfitting. Next, there is a dense layer with a random number of units ranging from 32 to 128. Finally, the model is connected to a dense layer with output units representing the predicted sentiment classes, using the Softmax activation function for multiclass sentiment classification.

5.6.2. Long Short-Term Memory

This section presents a high-level Python implementation of the LSTM architecture using Keras with a TensorFlow backend.

```
"LSTM": [
        Embedding(input_dim=vocab_size, output_dim=embedding_dim,
                  input_length=max_sequence_length),
        LSTM(units=random.randint(64, 256)),
        Dropout(random.uniform(0.4, 0.6)),
        Dense(units=output_units, activation='softmax')
    ]
```

Figure 20. LSTM Model implementation

The LSTM model utilizes recurrent connections to capture sequential information in forward directions. Based on the above-mentioned parameter values, we have the following model architectures.

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 147, 100)          1065900

 lstm_16 (LSTM)              (None, 229)               302280

 dropout_21 (Dropout)        (None, 229)               0

 dense_35 (Dense)            (None, 3)                 690

=================================================================
Total params: 1368870 (5.22 MB)
Trainable params: 1368870 (5.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 21. Model summary for LSTM model

The LSTM model consists of an LSTM layer with a random number of units between 64 and 256. This layer captures sequential dependencies in the input text. A Dropout layer with a random dropout rate between 0.4 and 0.6 is applied after the LSTM layer to prevent

overfitting. The output is then connected to a dense layer with output units representing the predicted sentiment classes, using the Softmax activation function.

5.6.3. Bidirectional Long Short-Term Memory

In this section, we present a comprehensive Python implementation of the BiLSTM architecture for sentiment classification, using Keras with a TensorFlow backend.

```
"BiLSTM": [
    Embedding(input_dim=vocab_size, output_dim=embedding_dim,
              input_length=max_sequence_length),
    Bidirectional(LSTM(units=random.randint(64, 256))),
    Dropout(random.uniform(0.4, 0.6)),
    Dense(units=output_units, activation='softmax')
]
```

Figure 22. BILSTM Model implementation

The LSTM model utilizes recurrent connections to capture sequential information, with the BiLSTM model enhancing this by processing sequences in both forward and backward directions simultaneously. With consideration to the aforementioned parameter values, we present the following model architectures.

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_11 (Embedding)    (None, 147, 100)          1065900

 bidirectional_8 (Bidirecti  (None, 168)               124320
 onal)

 dropout_22 (Dropout)        (None, 168)               0

 dense_36 (Dense)            (None, 3)                 507

=================================================================
Total params: 1190727 (4.54 MB)
Trainable params: 1190727 (4.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 23. Model summary for BILSTM model

The BiLSTM model incorporates a Bidirectional LSTM layer, which combines both forward and backward sequential dependencies in the text. The number of units in the Bidirectional LSTM layer is randomly selected between 64 and 256. A Dropout layer with a random dropout rate between 0.4 and 0.6 follows the Bidirectional LSTM layer to prevent overfitting. Finally, the output is connected to a dense layer with output units representing the predicted sentiment classes, using the Softmax activation function.

5.6.4. Convolutional Neural Network-Bidirectional Long Short Term Memory

Within this section, we provide a high-level Python implementation of the CNN-BiLSTM architecture for sentiment classification using Keras with a TensorFlow backend.

```
"CNN-BiLSTM": [
    Conv1D(filters=random.randint(16, 64),
           kernel_size=random.choice([3, 5, 7])),
    MaxPooling1D(pool_size=random.choice([2, 3])),
    Embedding(input_dim=vocab_size, output_dim=embedding_dim,
              input_length=max_sequence_length),
    Bidirectional(LSTM(units=random.randint(64, 256))),
    Dropout(random.uniform(0.4, 0.6)),
    Dense(units=output_units, activation='softmax')
]
```

Figure 24. CNN-BILSTM Model implementation

The analysis of the model summaries reveals the different architectural approaches and parameter distributions, providing insights into how the model processes and learns from textual data for accurate classification tasks. Based on the above-mentioned parameter values, we have the following model architectures.

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_13 (Embedding)    (None, 147, 100)          1065900

 conv1d_14 (Conv1D)          (None, 143, 59)           29559

 max_pooling1d_14 (MaxPooli  (None, 47, 59)            0
 ng1D)

 bidirectional_9 (Bidirecti  (None, 484)               584672
 onal)

 dropout_24 (Dropout)        (None, 484)               0

 dense_38 (Dense)            (None, 3)                 1455

=================================================================
Total params: 1681586 (6.41 MB)
Trainable params: 1681586 (6.41 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 25. Model summary for CNN-BILSTM model

The CNN-BiLSTM model combines features from both convolutional and bidirectional LSTM layers. It starts with a Conv1D layer, similar to the previous models, with random filters in the range of 16 to 64 and a kernel size randomly selected from [3, 5, and 7]. A MaxPooling1D layer with a random pool size of 2 or 3 is applied. The architecture then includes a Bidirectional LSTM layer with a random number of units between 64 and 256 to capture both forward and backward sequential dependencies. A Dropout layer with a random dropout rate between 0.4 and 0.6 is used for regularization. Finally, the output is connected to a dense layer with output units representing the predicted sentiment classes, using the Softmax activation function.

5.7. Model Training and Evaluation

To evaluate its performance, we integrate cross-validation into the training process. We establish specific parameters, such as patience for early stopping and a learning rate.

```
skf = StratifiedKFold(n_splits=k_fold, shuffle=True, random_state=42)
for model_name, model_layers in models.items():

    for train_index, test_index in skf.split(X_pad, balanced_df["sentiment"].values):
        X_train_fold, X_test_fold = X_pad[train_index], X_pad[test_index]
        y_train_fold, y_test_fold = balanced_df["sentiment"].values[train_index],
                                    balanced_df["sentiment"].values[test_index]
        vocab_size = len(tokenizer.word_index) + 1
        embedding_dim = 100
        model = Sequential()
        model.add(Embedding(input_dim=vocab_size,
                            output_dim=embedding_dim,
                            input_length=max_sequence_length))
        for layer in model_layers:
        model.add(layer)
        model.compile(loss='sparse_categorical_crossentropy',
                      optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                      metrics=['accuracy'])

        history = model.fit(X_train_fold, y_train_fold, epochs=epochs,
                  batch_size=batch_size,
                  validation_data=(X_test_fold, y_test_fold),
                  callbacks=[early_stopping])
```

Figure 26. Implementation for model training

To implement cross-validation, we use the StratifiedKFold method. This helps us iterate over the training and test indices. In each split, we divide the padded input sequences, X_pad, into training (X_train_fold) and testing (X_test_fold) subsets. Similarly, we split the sentiment labels from the balanced_df dataframe into y_train_fold and y_test_fold.

Next, we calculate the vocabulary size by adding 1 to the tokenizer's word index. We then create a Sequential model. The model starts with an embedding layer that specifies the input dimensions, output dimensions, and input sequence length. We add layers from the model_layers list, which represent the chosen architecture. For model compilation, we use sparse categorical cross-entropy as the loss function, the Adam optimizer with the specified learning rate, and the accuracy metric. Once compiled, we train the model using the training data (X_train_fold, y_train_fold), setting the number of epochs and batch size accordingly. The validation data (X_test_fold, y_test_fold) is also provided. Additionally, we include the early stopping callback to monitor validation loss and stop training if no improvement is seen for consecutive epochs. By running the model.fit () function, we train the model on the training data while evaluating its performance on the testing data.

CHAPTER SIX

6. Result and Discussions

6.1. Results

We conducted four distinct experiments to investigate the impact of TF-IDF and Count Vectorizer, ranging from unigrams to trigrams, on sentiment classification in code-mixed texts. Additionally, we integrated various preprocessing techniques, including language detection and code-switching. Our aim was to identify the most effective model for this task by employing both TF-IDF vectorization and Count Vectorizer with n-grams. Through a thorough examination of cross-validation results and performance metrics across all folds, along with validation curves and visualization using bar graphs, we scrutinized the experimental outcomes of this study. Our research underscores the significance of preprocessing methods that incorporate code-switching and language detection for analyzing code-mixed text, specifically focusing on code-mixed text classification in conjunction with the mentioned vectorization techniques and various deep learning algorithms configured with different parameters.

6.1.1. Cross Validation Result

Experiment 1: ***Using Count Vector with various preprocessing techniques involving Language detection and code-switching***

In the first experiment, we utilized Count Vectorizer (with an n-gram range of 1 to 3) on a code-mixed text dataset. We applied several preprocessing steps, including cleaning, normalization, stopwords removal, tokenization, language detection, and code-switching. The models CNN, LSTM, BiLSTM, and CNN-BiLSTM were evaluated using 10-fold cross-validation, and their average accuracy scores were recorded. The following table presents the average accuracy scores across the 10-fold cross-validation:

	Accuracy										
Models	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Fold 7**	**Fold 8**	**Fold 9**	**Fold 10**	**Average**
CNN	0.8963	0.7801	0.8548	0.8506	0.8299	0.8299	0.8417	0.85	0.8625	0.85	**0.84458**
LSTM	0.8797	0.7884	0.805	0.7676	0.8257	0.7801	0.7792	0.7792	0.8	0.7667	0.79716
BiLSTM	0.8548	0.7884	0.7469	0.8672	0.8506	0.8382	0.7917	0.7833	0.8083	0.8292	0.81586
CNN-BiLSTM	0.8423	0.7759	0.8133	0.8423	0.8465	0.7925	0.8542	0.7958	0.8333	0.8167	0.82128

Figure 27. Performance of models using Count Vectorizer with code-switching

Upon conducting a comprehensive analysis of the accuracy attained by each model for multiclass sentiment classification across various folds, several notable findings arise. The CNN model showcases performance variability, with accuracy ranging from 0.7801 to 0.8963 across folds and an average accuracy of 0.84458. Despite exhibiting relatively high average performance, there is variance in its performance across different folds, suggesting potential inconsistency in effectiveness across subsets of the data. Specifically, Fold 1 and Fold 9 demonstrate the highest accuracy, while Fold 2 exhibits the lowest accuracy. Similarly, the LSTM model demonstrates variability in accuracy, ranging from 0.7667 to 0.8797 across folds, with an average accuracy of 0.79716. Like the CNN model, LSTM also displays variance in performance across folds, indicating that its effectiveness may vary depending on the evaluated data subset. Among the folds, Fold 1 performs the best, while Fold 10 displays the lowest accuracy. The BiLSTM model also exhibits performance variability, with accuracy ranging from 0.7469 to 0.8672 and an average accuracy of 0.81586. Similar to the CNN and LSTM models, BiLSTM showcases variance across folds, with Fold 4 demonstrating the highest accuracy and Fold 3 displaying the lowest. In contrast, the CNN-BiLSTM model maintains relatively consistent performance, with accuracy ranging from 0.7759 to 0.8542 and an average accuracy of 0.82128. While its performance is comparable to the other models, it also exhibits variability across folds. Notably, Fold 4 achieves the highest accuracy, while Fold 3 exhibits the lowest accuracy. Overall, all models demonstrate potential for multiclass sentiment classification. However, the CNN model stands out due to its consistently superior performance across different folds, as evidenced by its highest average accuracy. The CNN-BiLSTM model achieves the second-highest accuracy, followed by BiLSTM and LSTM.

Experiment 2: **Using Count Vector with various preprocessing techniques without language detection and code-switching**

In Experiment 2, the same vectorization techniques (Count Vectorizer) with an n-gram range of 1 to 3 were applied to the text after performing preprocessing steps like cleaning, normalization, stopwords removal, and tokenization without language detection and code-switching technique. The models (CNN, LSTM, BiLSTM, and CNN-BiLSTM) underwent evaluation using 10-fold cross-validation, and their average accuracy scores were recorded.

	Accuracy										
Models	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Fold 7**	**Fold 8**	**Fold 9**	**Fold 10**	**Average**
CNN	0.8902	0.7755	0.849	0.8041	0.7837	0.7633	0.8163	0.8408	0.8163	0.8612	**0.82004**
LSTM	0.8699	0.7959	0.8367	0.8245	0.7714	0.702	0.7878	0.7755	0.8327	0.8327	0.80291
BiLSTM	0.813	0.8327	0.8082	0.8408	0.8041	0.7878	0.8245	0.7796	0.7755	0.8204	0.80866
CNN-BiLSTM	0.874	0.8245	0.7837	0.7959	0.8204	0.7673	0.7837	0.7837	0.7755	0.8286	0.80373

Figure 28. Performance of models using Count Vectorizer without code-switching

The experiment evaluated different models using Count Vectorizer and various preprocessing techniques without language detection and code-switching. The evaluation results for each model are summarized in the table. The experiment assessed various models using Count Vectorizer and different preprocessing techniques while excluding language detection and code-switching. The evaluation outcomes for each model are summarized in the table. The CNN model exhibited relatively high accuracy values across the ten folds, ranging from 0.7633 to 0.8902. Although there is some variance across folds, it is comparatively lower than the previous experiments that did not involve language detection and code-switching.

In comparison, the LSTM model demonstrated slightly lower performance than the CNN model, achieving accuracy values ranging from 0.702 to 0.8699. The variance in performance across folds remains noticeable, indicating that the effectiveness of LSTM may vary for different subsets of data. Likewise, the BiLSTM model performed similarly to the LSTM model, with accuracy values ranging from 0.7755 to 0.8408. Similar to LSTM, it also displayed variance across folds, suggesting potential variability in its effectiveness. The CNN-BiLSTM model achieved accuracy values ranging from 0.7673 to 0.874, with an average accuracy of 82.128%. The variance across folds is present but comparatively lower than LSTM and BiLSTM. When considering overall performance, the CNN model consistently outperformed the other models in terms of accuracy, precision, recall, and F1 score. Therefore, based on the evaluation results, the CNN model with Count Vectorizer and the preprocessing techniques without language detection and code-switching appears to be the most suitable model for the given task.

Experiment 3: **Using TF-IDF Vectorizer with various Preprocessing Techniques involving Language Detection and Code-Switching**

In the third experiment using the TF-IDF Vectorizer with an n-gram range of 1 to 3 range on the code-mixed text following preprocessing steps such as cleaning, normalization,

stopwords removal, tokenization, language detection, and code-switching, the same models were evaluated, and the average accuracy scores across the 10-fold cross-validation are as follows:

	Accuracy										
Models	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Fold 7**	**Fold 8**	**Fold 9**	**Fold 10**	**Average**
CNN	0.7842	0.7925	0.7759	0.8299	0.8465	0.7884	0.8333	0.8583	0.8583	0.8958	0.82631
LSTM	0.8174	0.7593	0.8008	0.8257	0.8216	0.805	0.8208	0.875	0.8	0.875	0.82006
BiLSTM	0.8091	0.8091	0.8216	0.8423	0.7925	0.8091	0.7917	0.8083	0.8	0.8333	0.8117
CNN-BiLSTM	0.834	0.8257	0.8465	0.8133	0.8797	0.834	0.8375	0.8333	0.8333	0.8125	**0.83498**

Figure 29. Performance of models using TF-IDF with code-switching

The evaluation results offer valuable insights into the performance metrics of several models, including CNN, LSTM, BiLSTM, and CNN-BiLSTM, in the field of sentiment classification. These models utilize TF-IDF vectorization and a variety of preprocessing techniques, excluding language detection and code-switching, to showcase their ability to capture subtle nuances of sentiment. Beginning with the CNN model, it achieves an average accuracy of 82.631%, consistently performing well across the ten folds. The accuracy ranges from 77.59% to 85.89% across the folds, while precision, recall, and F1 score remain relatively stable with minor variations.

Moving on to the LSTM model, it achieves an average accuracy of 82.006%, with accuracy values ranging from 75.93% to 87.5% across the folds. Similar to the CNN model, precision, recall, and F1 score exhibit consistent performance with slight fluctuations. The BiLSTM model achieves an average accuracy of 81.117%, with accuracy ranging from 79.17% to 84.23% across the folds. Like the previous models, precision, recall, and F1 score demonstrate consistent performance with minor variances. Lastly, the CNN-BiLSTM model emerges as the leading model with the highest average accuracy of 83.498%. Its accuracy ranges from 81.25% to 87.97% across the folds, showcasing its robust performance. Consistent with the other models, precision, recall, and F1 score exhibit reliable consistency, further affirming the effectiveness of the model.

In summary, all models demonstrate commendable efficacy in sentiment classification when utilizing TF-IDF vectorization and various preprocessing techniques, excluding language detection and code-switching. The CNN-BiLSTM model stands out with its superior average

accuracy, closely followed by the CNN model. Despite slightly lower performance metrics, the LSTM and BiLSTM models also present strong contenders for sentiment analysis tasks.

Experiment 4: **Using TF-IDF with various Preprocessing Techniques without Language Detection and Code-Switching**

For the fourth experiment, we applied the same vectorization techniques (TF-IDF) with an n-gram range of 1 to 3 to the text following basic preprocessing steps like cleaning, normalization, stopwords removal, and tokenization. The models (CNN, LSTM, BiLSTM, and CNN-BiLSTM) underwent evaluation using 10-fold cross-validation, with their average accuracy scores recorded.

	Accuracy										
Models	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Fold 7**	**Fold 8**	**Fold 9**	**Fold 10**	**Average**
CNN	0.8589	0.8185	0.8145	0.8347	0.7944	0.8178	0.7935	0.8138	0.8178	0.8462	0.82101
LSTM	0.8427	0.8669	0.7984	0.8065	0.7742	0.7773	0.7409	0.7895	0.8259	0.7814	0.80037
BiLSTM	0.8226	0.8226	0.8347	0.8065	0.8266	0.8016	0.7409	0.8178	0.8016	0.7652	0.80401
CNN-BiLSTM	0.8468	0.8427	0.875	0.8669	0.8226	0.834	0.8381	0.834	0.8057	0.834	**0.83998**

Figure 30. Performance of models using TF-IDF Vectorizer without code-switching

The evaluation results offer a comprehensive understanding of the performance of multiple models (CNN, LSTM, BiLSTM, and CNN-BiLSTM) in sentiment classification using TF-IDF vectorization with diverse preprocessing techniques, excluding language detection and code-switching. Starting with the CNN model, it consistently demonstrates strong performance across all folds, achieving an accuracy range of 79.35 % to 85.89 % and an average accuracy of 82.10%. Notably, precision, recall, and F1 score exhibit stability, highlighting the CNN model's robustness without language detection and code-switching preprocessing. The LSTM model closely follows the CNN model, with performance ranging from 74.09 % to 86.69% and an average accuracy of 80.037%. While generally solid, precision, recall, and F1 score show some fluctuations across folds, indicating sensitivity to dataset variations.

Similarly, the BiLSTM model achieves performance within the range of 74.09 % to 83.47%, with an average accuracy of 80.401%. Although consistent, precision, recall, and F1 score exhibit variability across folds, suggesting susceptibility to dataset nuances. On the other hand, the CNN-BiLSTM model outperforms individual LSTM and BiLSTM models, achieving accuracy between 80.57% and 87.50%, with an average accuracy of 83.998%.

Additionally, precision, recall, and F1 score maintain stability across folds, highlighting the CNN-BiLSTM model's consistent performance without language detection and code-switching preprocessing. In summary, all models perform well without language detection and code-switching preprocessing, with the CNN model consistently leading and the CNN-BiLSTM model closely following. Despite slightly lower performance, the LSTM and BiLSTM models still demonstrate respectable accuracy.

6.1.2. Validation Curve for Models

Validation curves with average accuracy are a common tool used to evaluate the performance of machine learning models during the training process. These curves plot the average accuracy scores obtained from cross-validation or validation data against different hyperparameter values or model complexities. By analyzing these curves, researchers can gain insights into how the model's performance changes as the hyperparameters or model complexity are varied. Validation curves provide a visual representation of the model's learning dynamics, allowing for informed decision-making during the model selection and hyperparameter tuning process. To conduct a comprehensive analysis of sentiment classification results and experimental findings across each experiment, let's delve into the specifics:

In Experiment 1, The CNN model consistently achieves the highest average accuracy, followed by BILSTM and CNN-BILSTM, while LSTM consistently performs the poorest.

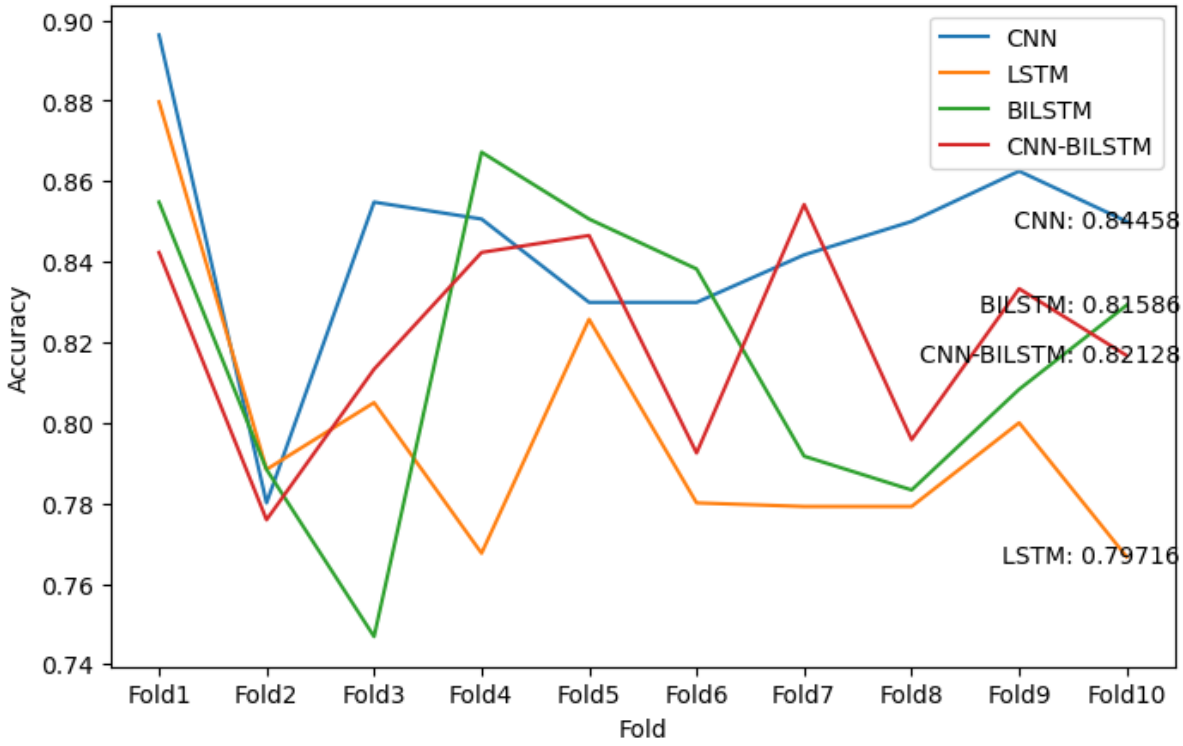


Figure 31. Accuracy Validation curves for models using Count Vectorizer with code-switching

Notably, there's notable variability in model performance across different folds, indicating sensitivity to dataset partitioning. Regarding socio-political posts, the CNN model's superior performance suggests its efficacy in capturing sentiment features within such discourse, while LSTM's struggles may indicate difficulties with nuanced sentiment expressions or long-term dependencies. The performance gap between BILSTM and CNN might be attributed to BILSTM's complexity, potentially leading to overfitting or challenges in learning hierarchical representations compared to the more straightforward CNN architecture, which excels at capturing local patterns, crucial for short text inputs like tweets.

Experiment 2 reveals consistent trends where CNN and CNN-BILSTM outperform LSTM and BILSTM across folds. BILSTM consistently exhibits lower average accuracy, particularly in later folds.

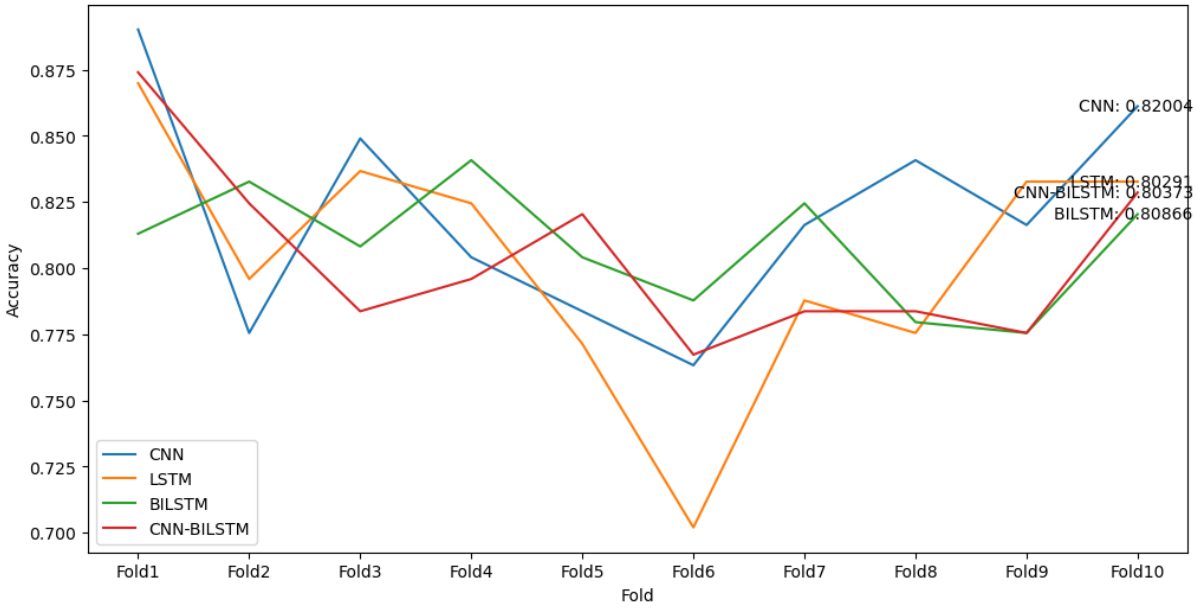


Figure 32. Accuracy validation curves for models using Count Vectorizer without code-switching

In the socio-political context, CNN and CNN-BILSTM's steady performance suggests their adeptness at capturing sentiment nuances efficiently, while LSTM's fluctuating results may point to difficulties in contextual understanding. CNN's consistency over CNN-BILSTM might be due to its simpler architecture, possibly preventing overfitting and enabling better feature capture in tweet sentiments.

In Experiment 3, both CNN and CNN-BILSTM maintain higher average accuracy, with CNN-BILSTM having a slight edge.

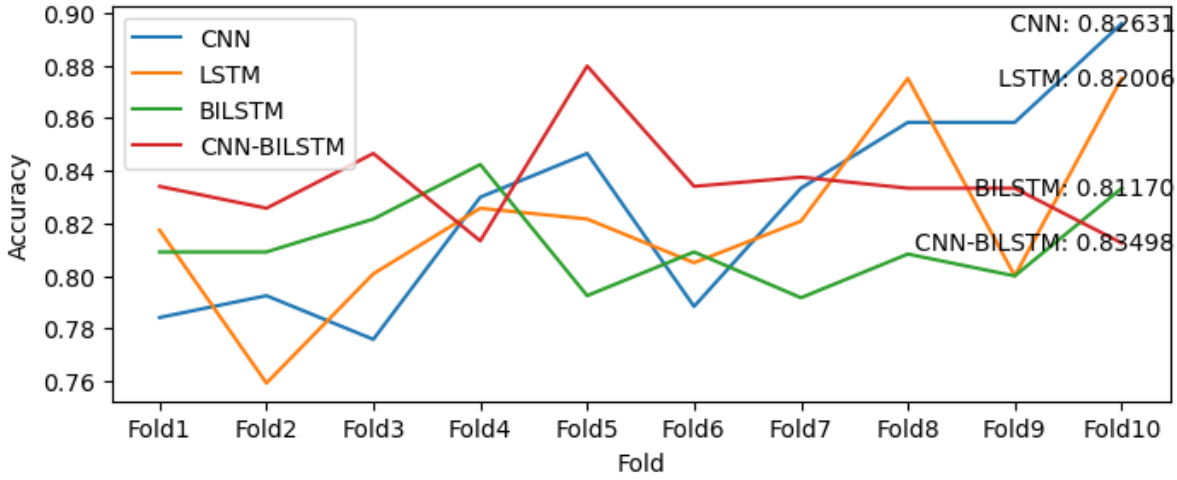


Figure 33. Accuracy validation curves for models using TF-IDF with code-switching

LSTM and BILSTM show comparable performance, albeit with fluctuations. This suggests CNN and CNN-BILSTM's suitability for sentiment classification in socio-political posts, possibly due to their effective capture of context-aware features. LSTM's improvement over Experiment 2 indicates adaptability, albeit still lagging behind CNN-based models. LSTM's simpler architecture contributes to its stability compared to BILSTM, which might struggle with overfitting or representation learning.

Experiment 4 highlights CNN-BILSTM as the top performer, with LSTM showing notable improvement over Experiment 3. CNN-BILSTM's success underscores the effectiveness of combining convolutional and recurrent structures in capturing sentiment nuances.

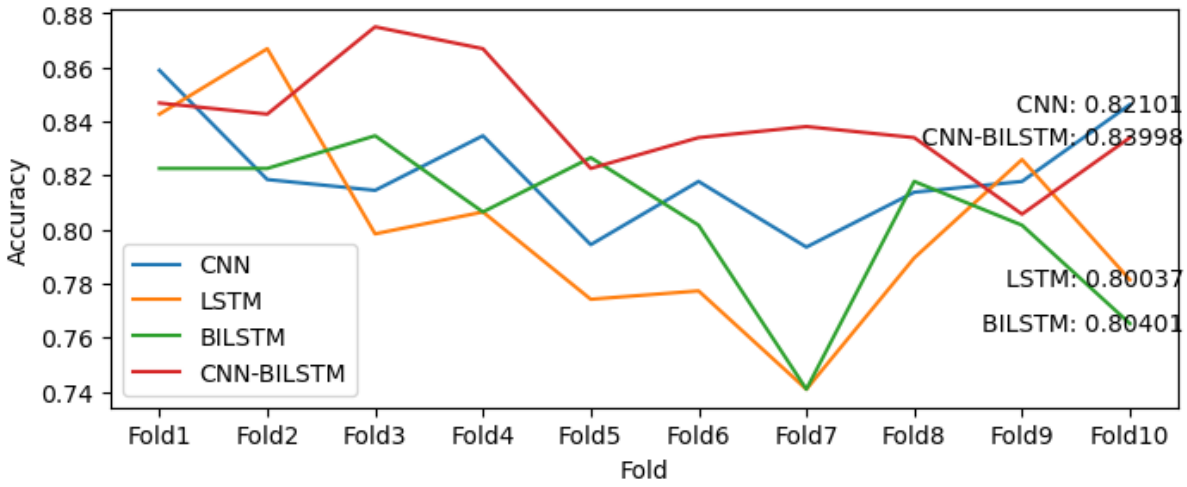


Figure 34. Accuracy validation curves for models using TF-IDF without code-switching

The comparison between CNN and CNN-BILSTM suggests CNN-BILSTM's ability to capture both local and global features effectively, leveraging the strengths of both architectures. CNN's simplicity may limit its ability to capture nuanced sentiment expressions, especially in code-mixed texts.

6.1.3. Performance comparison of Models

The bar graph representation of cross-validation accuracy is a valuable tool that provides a concise and informative way to summarize and compare the performance of machine learning models. In this graphical format, each bar corresponds to the average accuracy score calculated across the different folds or iterations of the cross-validation process, with the height of the bar representing the magnitude of the average accuracy.

In experiment 1, the Convolutional Neural Network (CNN) emerges as the top performer, displaying the highest average accuracy among all the models.

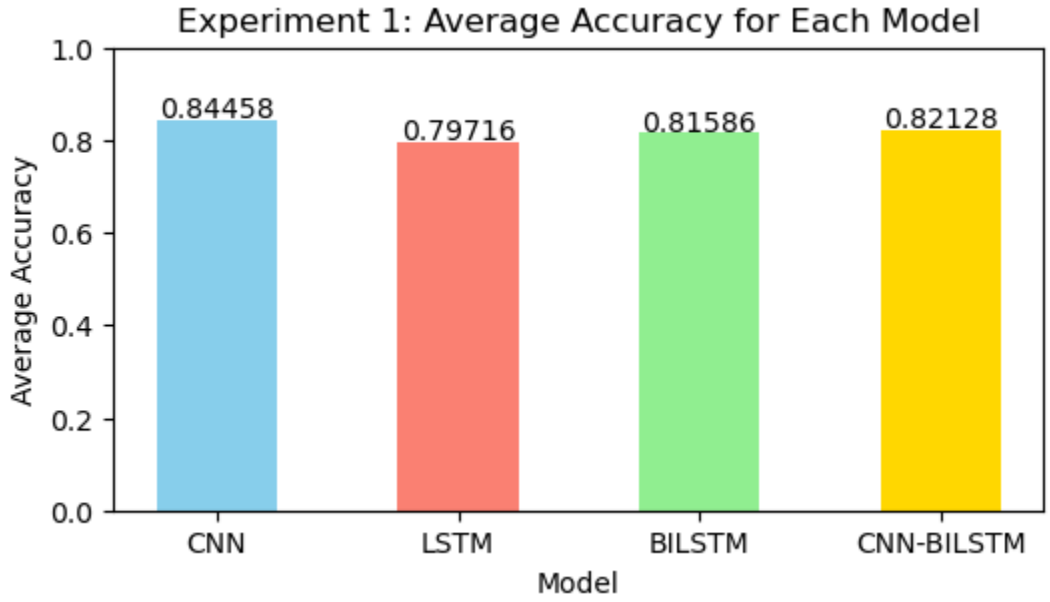


Figure 35. average accuracy for models using Count Vectorizer with code-switching

Closely following CNN are the Bidirectional Long Short-Term Memory (BILSTM) and the CNN-BILSTM hybrid model. Conversely, the Long Short-Term Memory (LSTM) model exhibits the lowest average accuracy. Notably, there are significant variations in accuracies across the different models, with CNN and BILSTM demonstrating comparatively superior performance.

In experiment 2, CNN and CNN-BILSTM showcase the highest average accuracies, indicating their robust performance.

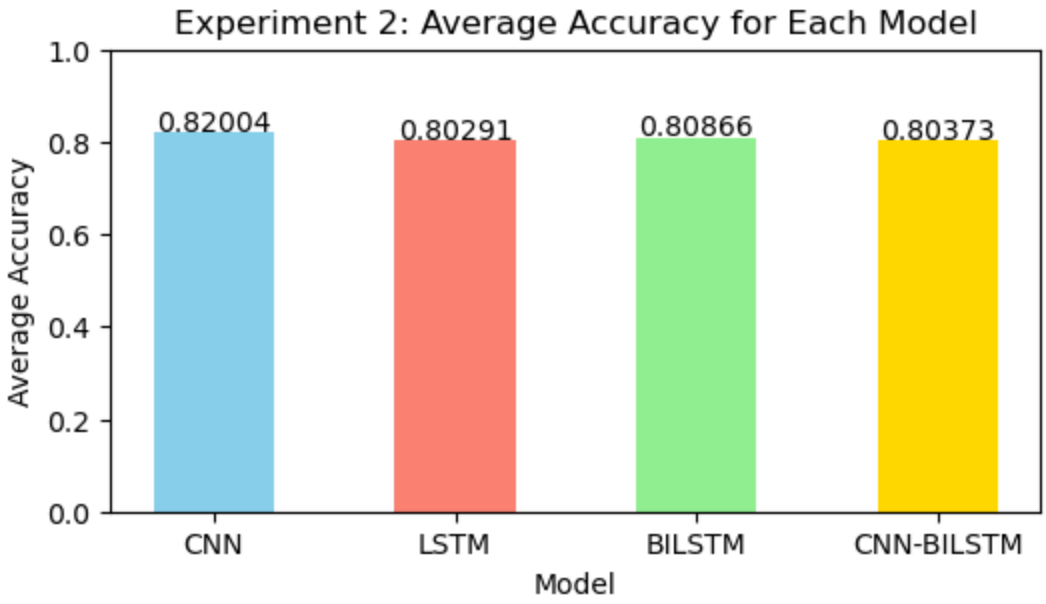


Figure 36. Average accuracy for models using Count Vectorizer without code-switching

While LSTM and BILSTM display slightly lower average accuracies, they remain competitive. Remarkably, the CNN-BILSTM hybrid model performs exceptionally well, matching CNN's performance consistently across the experiments.

Consistent with the findings from the previous experiments, in Experiment 3 the CNN and CNN-BiLSTM models continued to demonstrate reliable and steady performance. This reaffirms the robustness of these model architectures across the different experimental conditions.

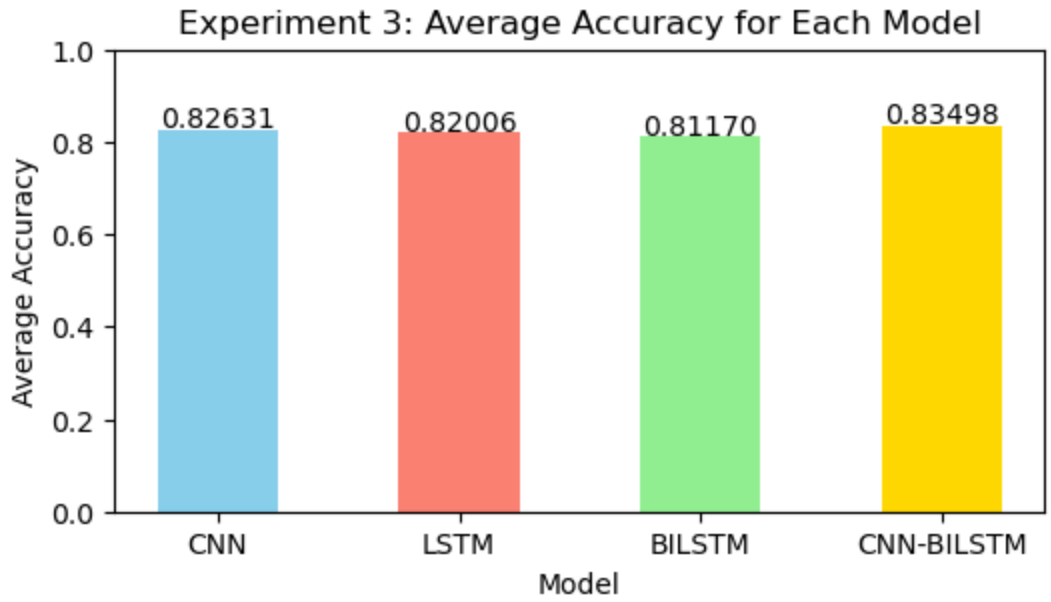


Figure 37. Average accuracy for models using TF-IDF with code-switching

While the LSTM model showed some improvement in performance compared to Experiment 2, it still trailed behind the CNN and CNN-BiLSTM approaches in terms of accuracy. This suggests that the more complex CNN-based models are better able to capture the relevant patterns in the data compared to the LSTM alone. In contrast, the BiLSTM model exhibited more variable performance, characterized by fluctuations in accuracy across the different experiments. This inconsistency indicates that the BiLSTM architecture may be more sensitive to the specific experimental setup or data characteristics, and may not provide the same level of reliable performance as the CNN-based models.

In experiment 4, the CNN-BILSTM hybrid model emerges as the standout performer, surpassing CNN in average accuracy.

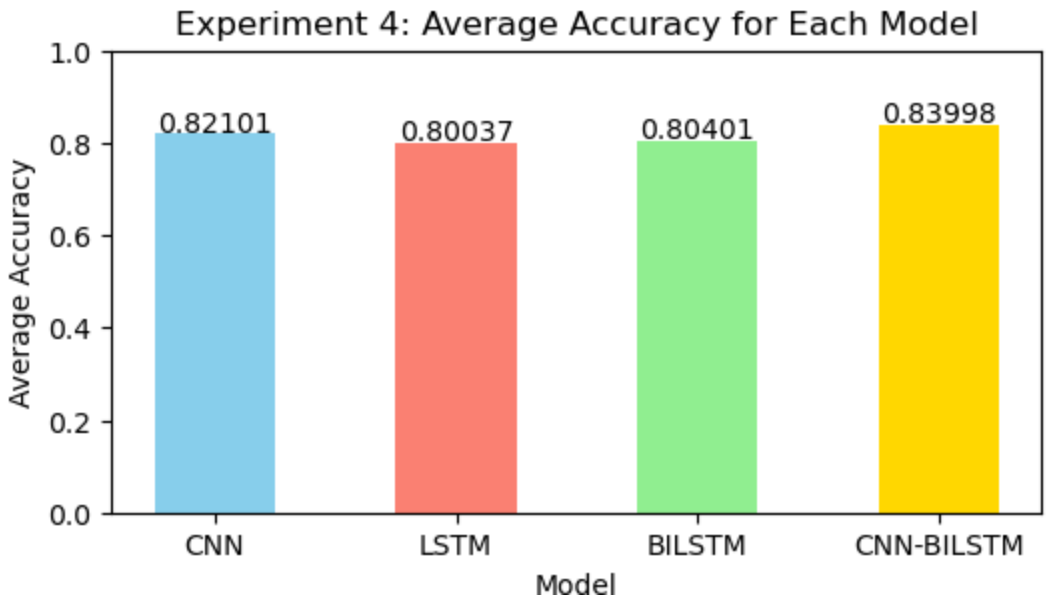


Figure 38. Average accuracy for models using TF-IDF without code-switching

LSTM also demonstrates a noteworthy improvement from Experiment 3, securing the position of the second-best performer. BILSTM and CNN exhibit similar average accuracies, slightly trailing behind CNN-BILSTM and LSTM.

6.2. Discussion

This study delved into evaluating the effectiveness of various preprocessing techniques, vectorization methods, and deep learning architectures for sentiment analysis on Amharic-English code-mixed texts, with a specific focus on Ethiopian socio-political issues. To tackle the challenges posed by the code-mixed nature of the text data, the researchers employed several preprocessing techniques.

The researchers recognized the importance of accurately identifying the primary language of each text segment to handle the code-mixed nature of the data effectively. They implemented language detection techniques to discern the dominant language in each text, be it Amharic or English. This was a crucial step as it allowed the researchers to apply appropriate tokenization and other processing methods tailored to the specific language of the text. Additionally, the researchers developed methods to identify and handle code-switching, which occur when a person alternates between multiple languages within a single text. By detecting and appropriately processing these code-switched segments, the researchers aimed to preserve the linguistic nuances and ensure accurate sentiment classification.

These preprocessing steps played a pivotal role in enhancing the model's understanding of the multilingual context and improving the overall sentiment classification accuracy. The language detection and code-switching handling techniques enabled the models to better capture the diverse linguistic patterns and sentiment expressions present in the code-mixed text data. Following the preprocessing steps, the researchers investigated the impact of different vectorization techniques on sentiment classification. They explored traditional approaches such as TF-IDF and count vectors, with varying n-gram ranges (e.g., 1-3), to represent the code-mixed text. These techniques allowed the models to capture the lexical and syntactic features of the text, which were crucial for sentiment analysis.

The researchers evaluated the performance of Convolutional Neural Network (CNN) and CNN-Bidirectional Long Short-Term Memory (CNN-BiLSTM) models on the new dataset. To ensure robustness and generalizability, cross-validation techniques were employed. Systematically partitioning the dataset and evaluating model performance across multiple folds allowed for more reliable and unbiased estimates of sentiment classification capabilities. The CNN model exhibited strong performance, effectively capturing the local patterns and spatial dependencies crucial for sentiment analysis in code-mixed texts. The parallel processing capabilities of CNNs facilitated efficient training on the large datasets required for this complex real-world task. However, the CNN-BiLSTM model, despite combining local and global feature extraction, did not match the performance of the standalone CNN, possibly due to challenges in accurately representing nuanced sentiment expressions common in short social media text segments.

Throughout the experiments, the CNN model consistently outperformed other architectures, demonstrating superior ability to identify and extract sentiment-related features from socio-political posts. Limitations of the LSTM in handling long-term dependencies and complexity issues with BiLSTM likely contributed to its weaker performance. The researchers suggest that the simpler CNN architecture led to greater stability and better feature extraction, emphasizing the importance of balancing model complexity with task requirements.

Proper preprocessing techniques, such as language detection and code-switching handling, were critical for improving model performance on the multilingual dataset. These preprocessing steps enabled the models to better capture the nuanced sentiment expressions

and linguistic complexities present in the code-mixed text, leading to enhanced sentiment classification accuracy.

These findings offer valuable insights for sentiment analysis, particularly in the context of socio-political discourse. The implications extend to real-world applications like sentiment monitoring in political discussions and social media analysis to inform policy-making. Further research is warranted to better understand the strengths and weaknesses of deep learning architectures, as well as the effectiveness of different preprocessing and vectorization techniques, for sentiment classification in Twitter data related to socio-political issues.

CHAPTER SEVEN

7. Conclusion and Future Work

7.1. Conclusion

The primary objective of this study was to perform multiclass sentiment classification on code-mixed texts using deep learning algorithms. We leveraged a dataset of 8,819 instances extracted from social media platforms, including Twitter and Facebook, categorized into positive, negative, and neutral sentiments. Various preprocessing techniques were employed, such as cleaning, language detection, normalization, tokenization, and stop-words removal, to prepare the data for analysis.

Following preprocessing, we applied vectorization techniques, including count vectorization and TF-IDF vectorization with unigram and trigram features, to convert the text into numerical representations suitable for machine learning. These techniques were chosen for their simplicity, efficacy, and compatibility with deep learning architectures, enabling the extraction of meaningful features from the text. We constructed models with various parameter configurations and evaluated them using 10-fold cross-validation. Our findings revealed that the CNN demonstrated superior performance, with an accuracy of 84.458% using count vectorization, outperforming the CNN-BiLSTM model, which attained an accuracy of 83.498% with TF-IDF vectorization.

The study highlights the effectiveness of CNNs in capturing local patterns and spatial dependencies, which are crucial for sentiment analysis in code-mixed texts. The inherent parallel processing capability of CNNs facilitated efficient training on large datasets, essential for complex real-world applications. In contrast, the CNN-BiLSTM model, despite its ability to capture both local and global features, struggled with nuanced sentiment expressions in shorter text segments, typical in socio-political discussions on social media.

Our observations suggest that enhancing the quality and quantity of the dataset could lead to improved model accuracy in future iterations, emphasizing the importance of robust datasets for training deep learning models for sentiment analysis. In conclusion, employing CNN with count vectorization, along with language detection, code-switching techniques, and thorough dataset cleaning, proved highly effective for multiclass sentiment analysis of code-mixed

social media texts in Amharic and English. These findings provide valuable insights into sentiment analysis, particularly in socio-political discourse, and have significant implications for real-world applications such as sentiment monitoring in political discussions and social media analysis for informed policy-making. The study advocates for continued research to further optimize model performance and address the specific challenges posed by code-mixed texts in diverse socio-political contexts.

7.2. Outcome of this Study

This study makes significant strides in sentiment analysis of code-mixed texts, specifically focusing on English and Amharic contexts. By addressing key challenges in multilingual societies and digital platforms, it develops tailored preprocessing techniques and introduces a specialized deep learning model, filling a critical research gap. The study's findings are crucial for researchers involved in sentiment analysis and social media analytics, offering valuable support for future research and encouraging further exploration of Amharic-English code-mixed sentiment analysis.

The study evaluates sentiment analysis performance on code-mixed with a focus on Amharic-English communication, highlighting the effectiveness of proposed models and techniques in handling code-mixing. This examination provides insights into the challenges and opportunities in sentiment analysis for code-mixed scenarios, informing future research and methodological advancements. Additionally, it explores how linguistic diversity and code-mixing patterns affect sentiment classification accuracy, contributing to our understanding of language variation's impact on sentiment analysis.

Moreover, the study identifies and evaluates various vectorization techniques for sentiment classification in code-mixed texts, enhancing our understanding of strategies that capture linguistic complexities. It also compares multiple deep learning algorithms for multiclass sentiment classification, offering insights into the suitability of specific architectures for sentiment analysis in code-mixed scenarios. This comprehensive evaluation provides valuable guidance for future researchers in selecting effective algorithms for sentiment analysis tasks in multilingual environments.

7.3. Future Work

In this study, we proposed a multiclass sentiment analysis model for code-mixed social media texts using a deep learning approach. While our results are promising, further research is required to fully develop sentiment analysis for Amharic-English code-mixed texts. To address the identified limitations and enhance the accuracy and reliability of English-Amharic code-mixed sentiment analysis, the following recommendations are suggested for future work:

1) Acquiring more labeled data specifically for the English-Amharic language pair is crucial to augment training resources and facilitate the development of robust models.
2) Developing dedicated tools tailored for code-mixed sentiment analysis, considering the linguistic complexities and variations in code-mixed text, can lead to improved performance.
3) Although we used term weighting techniques such as Tfidf and Count Vectorizer for dataset vectorization, future research should explore alternative approaches like word embedding.
4) Future work could explore extending the sentiment analysis to include emoticons, transliteration and part-of-speech tagging, for code-mixed texts, while addressing the challenges posed by the availability of special symbols, phrases, and nouns in this domain.
5) Lastly, future work should involve studying different language pairs and code-mixing scenarios to understand their unique challenges and devise models and techniques tailored to specific contexts.

References

[1] K. Lakhwani and J. Bala, “Deep Learning for Sentiment Analysis,” vol. 5, no. 10, 2018.

[2] K. Raviya and S. M. Vennila, “DEEP CNN WITH SVM - HYBRID MODEL FOR SENTENCE-BASED DOCUMENT LEVEL SENTIMENT ANALYSIS USING SUBJECTIVITY DETECTION,” *ICTACT J. SOFT Comput.*, vol. 11, no. 03, 2021.

[3] W. Philemon and W. Mulugeta, “A Machine Learning Approach to Multi-Scale Sentiment Analysis of Amharic Online Posts,” vol. 2, no. 2.

[4] A. Konate and R. Du, “Sentiment Analysis of Code-Mixed Bambara-French Social Media Text Using Deep Learning Techniques,” *Wuhan Univ. J. Nat. Sci.*, vol. 23, no. 3, pp. 237–243, Jun. 2018, doi: 10.1007/s11859-018-1316-z.

[5] Y. Y. Desai, “To Study The Social Media Sentimental Analysis Using Facebook As Platform,” no. 10, 2017.

[6] M. Mihret, “Sentiment Analysis Model for Opinionated Awngi Text : Case of Music Reviews,” pp. 2–4, 2017.

[7] G. Mutanov, V. Karyukin, and Z. Mamykova, “Multi-Class Sentiment Analysis of Social Media Data with Machine Learning Algorithms,” *Comput. Mater. Contin.*, vol. 69, no. 1, pp. 913–930, 2021, doi: 10.32604/cmc.2021.017827.

[8] A. D’Andrea, F. Ferri, P. Grifoni, and T. Guzzo, “Approaches, Tools and Applications for Sentiment Analysis Implementation,” *Int. J. Comput. Appl.*, vol. 125, no. 3, pp. 26–33, 2015, doi: 10.5120/ijca2015905866.

[9] C. D. Mengesha, “PUBLIC SENTIMENT ANALYSIS FOR AMHARIC NEWS”.

[10] L. W. Astuti, Y. Sari, and S. -, “Code-Mixed Sentiment Analysis using Transformer for Twitter Social Media Data,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 10, 2023, doi: 10.14569/IJACSA.2023.0141053.

[11] A. Priyanshu, A. Vardhan, S. Sivakumar, S. Vijay, and N. Chhabra, “ExCode-Mixed: Explainable Approaches towards Sentiment Analysis on Code-Mixed Data using BERT models.” arXiv, Sep. 25, 2021. Accessed: Dec. 25, 2023. [Online]. Available: http://arxiv.org/abs/2109.03200

[12] G. Singh, “Sentiment Analysis of Code-Mixed Social Media Text (Hinglish)”.

[13] Y. P. Babu, R. Eswari, and K. Nimmi, “CIA_NITT@Dravidian-CodeMix-FIRE2020: Malayalam-English Code Mixed Sentiment Analysis Using Sentence BERT And Sentiment Features.”.

[14] L. L. Maceda, A. A. Satuito, and M. B. Abisado, “Sentiment Analysis of Code-mixed Social Media Data on Philippine UAQTE using Fine-tuned mBERT Model,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 7, 2023, doi: 10.14569/IJACSA.2023.0140777.

[15] N. H. M. Et.al, “Sentiment Analysis of Code-Mixed Text: A Review,” *Turk. J. Comput. Math. Educ. TURCOMAT*, vol. 12, no. 3, pp. 2469–2478, Apr. 2021, doi: 10.17762/turcomat.v12i3.1239.

[16] P. Singh and E. Lefever, “Sentiment Analysis for Hinglish Code-mixed Tweets by means of Cross-lingual Word Embeddings”.

[17] N. Farra, “Cross-Lingual and Low-Resource Sentiment Analysis”.

[18] M. Errami, M. A. Ouassil, R. Rachidi, B. Cherradi, S. Hamida, and A. Raihani, “Sentiment Analysis on Moroccan Dialect based on ML and Social Media Content Detection,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 3, 2023, doi: 10.14569/IJACSA.2023.0140347.

[19] N. Wayessa and S. Abas, “Multi-Class Sentiment Analysis from Afaan Oromo Text Based On Supervised Machine Learning Approaches”.

[20] B. Liu, “Sentiment Analysis and Opinion Mining”.

[21] F. Alemayehu, M. Meshesha, and J. Abate, “Amharic Political Sentiment Analysis Using Deep Learning Approaches,” In Review, preprint, Jun. 2023. doi: 10.21203/rs.3.rs-3060010/v1.

[22] M. A. Ansari and S. Govilkar, “Sentiment Analysis of Mixed Code for The Transliterated Hindi and Marathi Texts,” *Int. J. Nat. Lang. Comput.*, vol. 7, no. 2, pp. 15–28, Apr. 2018, doi: 10.5121/ijnlc.2018.7202.

[23] A. Khodaei, A. Bastanfard, H. Saboohi, and H. Aligholizadeh, “Deep Emotion Detection Sentiment Analysis of Persian Literary Text,” In Review, preprint, Jul. 2022. doi: 10.21203/rs.3.rs-1796157/v1.

[24] B. Omarov and Z. Zhumanov, “Bidirectional Long-Short-Term Memory with Attention Mechanism for Emotion Analysis in Textual Content,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 6, 2023, doi: 10.14569/IJACSA.2023.0140615.

[25] A. A. Q. Aqlan, B. Manjula, and R. Lakshman Naik, “A Study of Sentiment Analysis: Concepts, Techniques, and Challenges,” in *Proceedings of International Conference on Computational Intelligence and Data Engineering*, vol. 28, N. Chaki, N. Devarakonda, A. Sarkar, and N. C. Debnath, Eds., in Lecture Notes on Data Engineering and Communications Technologies, vol. 28. , Singapore: Springer Singapore, 2019, pp. 147–162. doi: 10.1007/978-981-13-6459-4_16.

[26] D. Tang, B. Qin, and T. Liu, “Deep learning for sentiment analysis: successful approaches and future challenges,” *WIREs Data Min. Knowl. Discov.*, vol. 5, no. 6, pp. 292–303, Nov. 2015, doi: 10.1002/widm.1171.

[27] O. Habimana, Y. Li, R. Li, X. Gu, and G. Yu, “Sentiment analysis using deep learning approaches: an overview,” *Sci. China Inf. Sci.*, vol. 63, no. 1, p. 111102, Jan. 2020, doi: 10.1007/s11432-018-9941-6.

[28] R. Pawar, S. Salunkhe, S. Dhongdi, V. Waghamare, and T. A. Rane, “Sentiment Analysis of Code Mixed Text,” vol. 08, no. 12, 2021.

[29] K. Mehta and S. P. Panda, “Customer Reviews’ Sentiments Analysis using Deep Learning,” *Int. J. Comput. Appl.*, vol. 175, no. 30, pp. 27–31, Nov. 2020, doi: 10.5120/ijca2020920842.

[30] P. Singhal and P. Bhattacharyya, “Sentiment Analysis and Deep Learning: A Survey”.

[31] Girma Neshir Alemneh, A. Rauber, and S. Atnafu, “Semi supervised Amharic Sentiment clasification,” Jan. 2021, doi: 10.5281/ZENODO.4429734.

[32] T. Fikre, “ADDIS ABABA UNIVERSITY ADDIS ABABA INSTITUTE OF TECHNOLOGY SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING”.

[33] M. O. Rase, “Sentiment Analysis of Afaan Oromoo Facebook Media Using Deep Learning Approach,” vol. 90, pp. 7–22, 2020, doi: 10.7176/NMMC/90-02.

[34] Y. Mekonnen, “Thesis Title: DICTIONARY AND RULE BASED (HYBRID) APPROACH OF”.

[35] A. Kaura and E. G. Sharma, “Sentimental Analysis of Hotel Reviews Using Statistical Analysis Technique,” vol. 15, no. 5, 2017.

[36] M. Oljira, “Sentiment Analysis of Afaan Oromo using Machine learning Approach”.

[37] Y. Teshome, “DEBRE BERHAN UNIVERSITY COLLEGE OF COMPUTING DEPARTMENT OF INFORMATION SYSTEMS”.

[38] G. N. Alemneh, “ADDIS ABABA UNIVERSITY SCHOOL OF GRADUATE STUDIES”.

[39] P. Patil and P. Yalagi, “Sentiment Analysis Levels and Techniques : A Survey,” *Int. J. Innov. Eng. Technol.*, vol. 6, no. 4, pp. 523–528, 2016.

[40] S. G. Tesfagergish, R. Damasevicius, and J. Kapociūtė-Dzikienė, “Deep learning-based sentiment classification in Amharic using multi-lingual datasets,” *Comput. Sci. Inf. Syst.*, vol. 20, no. 4, pp. 1459–1481, 2023, doi: 10.2298/CSIS230115042T.

[41] L. Khan, A. Amjad, K. M. Afaq, and H.-T. Chang, “Deep Sentiment Analysis Using CNN-LSTM Architecture of English and Roman Urdu Text Shared in Social Media,” *Appl. Sci.*, vol. 12, no. 5, p. 2694, Mar. 2022, doi: 10.3390/app12052694.

[42] G. Neshir, A. Rauber, and S. Atnafu, “Meta-Learner for Amharic Sentiment Classification,” *Appl. Sci.*, vol. 11, no. 18, p. 8489, Sep. 2021, doi: 10.3390/app11188489.

[43] A. Younas, R. Nasim, S. Ali, G. Wang, and F. Qi, “Sentiment Analysis of Code-Mixed Roman Urdu-English Social Media Text using Deep Learning Approaches,” in *2020 IEEE 23rd International Conference on Computational Science and Engineering (CSE)*, Dec. 2020, pp. 66–71. doi: 10.1109/CSE50738.2020.00017.

[44] L. Khan, A. Amjad, N. Ashraf, and H.-T. Chang, “Multi-class sentiment analysis of urdu text using multilingual BERT,” *Sci. Rep.*, vol. 12, no. 1, p. 5436, Mar. 2022, doi: 10.1038/s41598-022-09381-9.

[45] P. Borele and D. A. Borikar, “An Approach to Sentiment Analysis using Artificial Neural Network with Comparative Analysis of Different Techniques”.

[46] A. Tedla and R. Tsegaye, “Content Based Multi-Class Amharic Short Message Service Classification using Machine Learning”.

[47] A. Hassan, M. R. Amin, A. K. A. Azad, and N. Mohammed, “Sentiment Analysis on Bangla and Romanized Bangla Text (BRBT) using Deep Recurrent models”.

[48] N. Choudhary, R. Singh, I. Bindlish, and M. Shrivastava, “Sentiment Analysis of Code-Mixed Languages Leveraging Resource Rich Languages,” in *Computational Linguistics and Intelligent Text Processing*, vol. 13397, A. Gelbukh, Ed., in Lecture Notes in Computer Science, vol. 13397. , Cham: Springer Nature Switzerland, 2023, pp. 104–114. doi: 10.1007/978-3-031-23804-8_9.

Appendix 1: Sample Source Code

Appendix A. A Sample Code for Normalization

```
def normalize_char_level_missmatch(input_token):
    replacements = {
            '[ሃኅኃሐሓኻ]': 'ሀ',
            '[ሑኁኹ]': 'ሁ',
            '[ኂሒኺ]': 'ሂ',
            '[ኌሔኼ]': 'ሄ',
            '[ሕኅኽ]': 'ህ',
            '[ኆሖኾ]': 'ሆ',
            '[ሠ]': 'ሰ',
            '[ሡ]': 'ሱ',
            '[ሢ]': 'ሲ',
            '[ሣ]': 'ሳ',
            '[ሤ]': 'ሴ',
            '[ሥ]': 'ስ',
            '[ሦ]': 'ሶ',
            '[ዓኣዐ]': 'አ',
            '[ዑ]': 'ኡ',
            '[ዒ]': 'ኢ',
            '[ዔ]': 'ኤ',
            '[ዕ]': 'እ',
            '[ዖ]': 'ኦ',
            '[ጸ]': 'ፀ',
            '[ጹ]': 'ፁ',
            '[ጺ]': 'ፂ',
            '[ጻ]': 'ፃ',
            '[ጼ]': 'ፄ',
            '[ጽ]': 'ፅ',
            '[ጾ]': 'ፆ',
            '(ሉ[ዋአ])': 'ሏ',
            '(ሙ[ዋአ])': 'ሟ',
            '(ቱ[ዋአ])': 'ቷ',
            '(ሩ[ዋአ])': 'ሯ',
            '(ሱ[ዋአ])': 'ሷ',
            '(ሹ[ዋአ])': 'ሿ',
            '(ቁ[ዋአ])': 'ቋ',
            '(ቡ[ዋአ])': 'ቧ',
            '(ቹ[ዋአ])': 'ቿ',
            '(ሁ[ዋአ])': 'ኋ',
            '(ኑ[ዋአ])': 'ኗ',
            '(ኙ[ዋአ])': 'ኟ',
            '(ኩ[ዋአ])': 'ኳ',
            '(ዙ[ዋአ])': 'ዟ',
            '(ጉ[ዋአ])': 'ጓ',
            '(ደ[ዋአ])': 'ዷ',
```

```
                    '(ጡ[ዋአ])': 'ጧ',
                    '(ጩ[ዋአ])': 'ቋ',
                    '(ጱ[ዋአ])': 'ፏ',
                    '(ፉ[ዋአ])': 'ፏ',
                    '[ቍ]': 'ቁ',
                    '[ኍ]': 'ኩ'
                }
for pattern, replacement in replacements.items():
    input_token = re.sub(pattern, replacement, input_token)
```

Appendix B: lists of Amharic Stopwords

ነዉ	ነዋ	የእኛ	መካከል	ኋላ
ለምን	ነገሮች	አንቺ	ከዚህ በፊት	ሁኔታ
ሌላ	አረ	ያንተ	ወደከ	ሆኑ
ማለት	አንተ	ራስህን	ወደ ላይ	ሆኖም
ማነው	አንች	እሱ	ተጨማሪ	ሁሌ
ምነው	አዎ	የእሱ	ከዚያ	ሁልጊዜ
ሰአት	እርስዎ	ራሱ	አንድ ጊዜ	ሁሉንም
ሲናገር	እሽ	እሷ	እዚያ	ላይ
ስም	እናም	የእሷ	መቼ	ማን
ስራ	እኔም	እራሷ	የት	ማንም
ቀን	እንዴ	እነሱ	እንዴት	ሰሞኑን
ቁጥር	እንድህ	እነሱን	አይ	ሲሆን
በለው	እኮ	የእነሱ	የራሱ	ጎን
በል	ኧረ	ራሳቸው	ተመሳሳይ	ሲል
በቃ	ከ	ምንድን	ይልቅ	እዚህ
በአል	ከቶ	የትኛው	እንዲሁ	እንጂ
በይ	የኔ	የአለም ጤና ድርጅት	በጣም	ከዚህ
ቢቢሲ	ይሄ	የሚል ነው	እ.ኤ.አ.	ከኋላ
ብለህ	ይሆን	እነዚያ	እኤአ	ከላይ
ብር	ዉስጥ	ነኝ	ዓ.ም	ከመሃል
ተባለ	ከላይ	አለው	ዓም	ከመካከል
ነህ	የአንተ	ከሆነ	ያደርጋል	ከታች
ነሽ	በ	የ	ሁሉ	ከውስጥ
ነበረ	እኔ	ለ	ሁሉም	ከጋራ
ነች	እኛ			ከፊት

ጋራ ስለዚህ ያለውን በሚል በዚሁ

ወዘተ ግን መሆኑን በፊት የሆኑ

ወይም ግዜ በመሆኑ ስለሆነም በኋላ

ወደ ጊዜ መሆን እንኳን ከሌላ

ዋና ጥቂት አንድ ነው በሌላ

ወደፊት ዛሬ እንኳ የስራ አሉ

ውስጥ ጋር እስከ እና የኛ

ውጭ ችግር እዚሁ በዚያ የለዩ

ውጪ ታች እንደ እነ ደግሞ

ያለ ትናንት ወቅት ይሁንና ይህንን

ሲሉ ነበረች እንዲሁም እረ ሆነ

ስለ ነበሩ የሚል በልዩ የሆነ

ቢሆን ነይ በላይ የእኔ ካለ

ብቻ ነገር ሳይሆን ከነ ይላሉ

ቦታ ናት ሆነው ላይም ዜሮ

በሰሞኑ ናቸው ምን ለዚሁ ነን

በታች አሁን ምንም በውስጡ ከያዘ

በኩል አለ ለንም ከነዚህም ለዚህ

በውስጥ እስካሁን ዓይነት ከእነዚህ ከእነ

ይህን ስለሆነ ይሁን ውስጥም

የታች አቶ በዚህ እኚህ

የውስጥ ይህም ያህል ኛ

የውጭ በአንድ እነዚህ ከኛው

የጋራ ጉዳይ በእነዚህ ኛው

ያ ነበር ታዲያ ወሰኑ

ይህ አይደለም እንዲህ የዚህ

ድረስ ያለው ይቻላል ለኛ

Appendix 2: Sample Outputs

Experiment1												
Using count vector with various Preprocessing Techniques involving Language Detection and Code-Switching												
		Folds										
Model	**Metrics**	**Fold1**	**Fold2**	**Fold3**	**Fold4**	**Fold5**	**Fold6**	**Fold7**	**Fold8**	**Fold9**	**Fold10**	**Average**
	Accuracy	0.8963	0.7801	0.8548	0.8506	0.8299	0.8299	0.8417	0.85	0.8625	0.85	**0.84458**
	Precision	0.8986	0.7793	0.8545	0.8507	0.8302	0.8289	0.8408	0.8496	0.8618	0.8513	0.84457
	Recall	0.8963	0.7801	0.8548	0.8506	0.8299	0.8299	0.8417	0.85	0.8625	0.85	0.84458
CNN	F1 Score	0.8968	0.7775	0.8545	0.8499	0.8292	0.829	0.8399	0.8495	0.8617	0.8493	0.84373
	Accuracy	0.8797	0.7884	0.805	0.7676	0.8257	0.7801	0.7792	0.7792	0.8	0.7667	0.79716
	Precision	0.8838	0.787	0.8066	0.7771	0.8306	0.7844	0.7779	0.7788	0.8116	0.7749	0.80127
	Recall	0.8797	0.7884	0.805	0.7676	0.8257	0.7801	0.7792	0.7792	0.8	0.7667	0.79716
LSTM	F1 Score	0.8792	0.7873	0.8034	0.7597	0.8223	0.7653	0.7731	0.771	0.791	0.7545	0.79068
	Accuracy	0.8548	0.7884	0.7469	0.8672	0.8506	0.8382	0.7917	0.7833	0.8083	0.8292	0.81586
	Precision	0.8534	0.7921	0.7828	0.866	0.859	0.8379	0.7924	0.78	0.8101	0.8337	0.82074
	Recall	0.8548	0.7884	0.7469	0.8672	0.8506	0.8382	0.7917	0.7833	0.8083	0.8292	0.81586
BILSTM	F1 Score	0.8537	0.7895	0.7438	0.8664	0.8521	0.8336	0.7846	0.7802	0.8049	0.8259	0.81347
	Accuracy	0.8423	0.7759	0.8133	0.8423	0.8465	0.7925	0.8542	0.7958	0.8333	0.8167	0.82128
	Precision	0.8479	0.7781	0.8129	0.8428	0.8468	0.7938	0.8539	0.7984	0.8354	0.8178	0.82278
	Recall	0.8423	0.7759	0.8133	0.8423	0.8465	0.7925	0.8542	0.7958	0.8333	0.8167	0.82128
CNN-BILSTM	F1 Score	0.8386	0.7764	0.8117	0.8392	0.846	0.7855	0.8532	0.7923	0.8336	0.8136	0.81901

Figure 39. Result of count vector with Language Detection and Code-Switching

Experiment2												
Using count vector with various Preprocessing Techniques without Language Detection and Code-Switching												
		Folds										
Model	**Metrics**	**Fold1**	**Fold2**	**Fold3**	**Fold4**	**Fold5**	**Fold6**	**Fold7**	**Fold8**	**Fold9**	**Fold10**	**Average**
	Accuracy	0.8902	0.7755	0.849	0.8041	0.7837	0.7633	0.8163	0.8408	0.8163	0.8612	**0.82004**
	Precision	0.8966	0.786	0.8508	0.8031	0.7822	0.7603	0.8161	0.841	0.8181	0.8633	0.82175
	Recall	0.8902	0.7755	0.849	0.8041	0.7837	0.7633	0.8163	0.8408	0.8163	0.8612	0.82004
CNN	F1 Score	0.891	0.7621	0.8496	0.7999	0.7761	0.758	0.8109	0.8408	0.8157	0.8577	0.81618
	Accuracy	0.8699	0.7959	0.8367	0.8245	0.7714	0.702	0.7878	0.7755	0.8327	0.8327	0.80291
	Precision	0.8728	0.7927	0.8377	0.8233	0.7704	0.7092	0.7868	0.7738	0.8319	0.837	0.80356
	Recall	0.8699	0.7959	0.8367	0.8245	0.7714	0.702	0.7878	0.7755	0.8327	0.8327	0.80291
LSTM	F1 Score	0.8704	0.7928	0.837	0.8233	0.7664	0.6896	0.7817	0.7695	0.832	0.8262	0.79889
	Accuracy	0.813	0.8327	0.8082	0.8408	0.8041	0.7878	0.8245	0.7796	0.7755	0.8204	0.80866
	Precision	0.8178	0.8321	0.8072	0.8466	0.8066	0.7915	0.8261	0.7775	0.773	0.8207	0.80991
	Recall	0.813	0.8327	0.8082	0.8408	0.8041	0.7878	0.8245	0.7796	0.7755	0.8204	0.80866
BILSTM	F1 Score	0.8104	0.8318	0.8076	0.8421	0.8018	0.7875	0.8251	0.7775	0.7728	0.8145	0.80711
	Accuracy	0.874	0.8245	0.7837	0.7959	0.8204	0.7673	0.7837	0.7837	0.7755	0.8286	0.80373
	Precision	0.8757	0.8237	0.7802	0.7953	0.8191	0.7673	0.7806	0.78	0.775	0.83	0.80269
	Recall	0.874	0.8245	0.7837	0.7959	0.8204	0.7673	0.7837	0.7837	0.7755	0.8286	0.80373
CNN-BILSTM	F1 Score	0.874	0.8207	0.779	0.7936	0.8174	0.764	0.7788	0.7769	0.7719	0.8233	0.79996

Figure 40. Result of count vector without Language Detection and Code-Switching

Experiment3												
TF-IDF with various Preprocessing Techniques involving Language Detection and Code-Switching												
		Folds										
Model	**Metrics**	**Fold1**	**Fold2**	**Fold3**	**Fold4**	**Fold5**	**Fold6**	**Fold7**	**Fold8**	**Fold9**	**Fold10**	**Average**
	Accuracy	0.7842	0.7925	0.7759	0.8299	0.8465	0.7884	0.8333	0.8583	0.8583	0.8958	0.82631
	Precision	0.7846	0.8039	0.7764	0.8301	0.8543	0.7899	0.8426	0.8612	0.8592	0.8957	0.82979
	Recall	0.7842	0.7925	0.7759	0.8299	0.8465	0.7884	0.8333	0.8583	0.8583	0.8958	0.82631
CNN	F1 Score	0.7774	0.7955	0.7721	0.83	0.8423	0.7841	0.8273	0.8571	0.8562	0.8958	0.82378
	Accuracy	0.8174	0.7593	0.8008	0.8257	0.8216	0.805	0.8208	0.875	0.8	0.875	0.82006
	Precision	0.8189	0.7589	0.7999	0.8308	0.8228	0.8028	0.8211	0.8767	0.802	0.8756	0.82095
	Recall	0.8174	0.7593	0.8008	0.8257	0.8216	0.805	0.8208	0.875	0.8	0.875	0.82006
LSTM	F1 Score	0.8147	0.7544	0.7995	0.8218	0.8177	0.8033	0.8153	0.8732	0.7942	0.8736	0.81677
	Accuracy	0.8091	0.8091	0.8216	0.8423	0.7925	0.8091	0.7917	0.8083	0.8	0.8333	0.8117
	Precision	0.8065	0.8126	0.8245	0.8434	0.7977	0.8069	0.7958	0.81	0.8016	0.8386	0.81376
	Recall	0.8091	0.8091	0.8216	0.8423	0.7925	0.8091	0.7917	0.8083	0.8	0.8333	0.8117
BILSTM	F1 Score	0.8055	0.8102	0.8219	0.8411	0.7839	0.8056	0.7837	0.8036	0.7925	0.8293	0.80773
	Accuracy	0.834	0.8257	0.8465	0.8133	0.8797	0.834	0.8375	0.8333	0.8333	0.8125	**0.83498**
	Precision	0.8319	0.8397	0.8489	0.8149	0.8801	0.8322	0.8363	0.8327	0.8361	0.8245	0.83773
	Recall	0.834	0.8257	0.8465	0.8133	0.8797	0.834	0.8375	0.8333	0.8333	0.8125	0.83498
CNN-BILSTM	F1 Score	0.8317	0.8282	0.8466	0.8108	0.8792	0.8328	0.836	0.8305	0.8299	0.8026	0.83283

Figure 41. Result of TF-IDF with Language Detection and Code-Switching

Experiment4												
Using TF-IDF with various Preprocessing Techniques without Language Detection and Code-Switching												
		Folds										
Model	**Metrics**	**Fold1**	**Fold2**	**Fold3**	**Fold4**	**Fold5**	**Fold6**	**Fold7**	**Fold8**	**Fold9**	**Fold10**	**Average**
	Accuracy	0.8589	0.8185	0.8145	0.8347	0.7944	0.8178	0.7935	0.8138	0.8178	0.8462	0.82101
	Precision	0.8603	0.8235	0.8224	0.8337	0.7962	0.8172	0.8	0.8125	0.8205	0.8463	0.82326
	Recall	0.8589	0.8185	0.8145	0.8347	0.7944	0.8178	0.7935	0.8138	0.8178	0.8462	0.82101
CNN	F1 Score	0.859	0.8119	0.8086	0.8313	0.7947	0.8153	0.7911	0.8095	0.8156	0.8438	0.81808
	Accuracy	0.8427	0.8669	0.7984	0.8065	0.7742	0.7773	0.7409	0.7895	0.8259	0.7814	0.80037
	Precision	0.8431	0.867	0.8025	0.8146	0.7734	0.7788	0.7461	0.7897	0.8247	0.7853	0.80252
	Recall	0.8427	0.8669	0.7984	0.8065	0.7742	0.7773	0.7409	0.7895	0.8259	0.7814	0.80037
LSTM	F1 Score	0.8418	0.8664	0.7957	0.7996	0.7663	0.7746	0.7272	0.7839	0.8243	0.7724	0.79522
	Accuracy	0.8226	0.8226	0.8347	0.8065	0.8266	0.8016	0.7409	0.8178	0.8016	0.7652	0.80401
	Precision	0.8238	0.8215	0.8374	0.8085	0.8253	0.8002	0.7393	0.8214	0.8031	0.7658	0.80463
	Recall	0.8226	0.8226	0.8347	0.8065	0.8266	0.8016	0.7409	0.8178	0.8016	0.7652	0.80401
BILSTM	F1 Score	0.823	0.8189	0.8315	0.7999	0.8257	0.7961	0.7284	0.8112	0.7948	0.7549	0.79844
	Accuracy	0.8468	0.8427	0.875	0.8669	0.8226	0.834	0.8381	0.834	0.8057	0.834	**0.83998**
	Precision	0.8456	0.8448	0.8766	0.8665	0.8267	0.8354	0.8375	0.8354	0.805	0.8322	0.84057
	Recall	0.8468	0.8427	0.875	0.8669	0.8226	0.834	0.8381	0.834	0.8057	0.834	0.83998
CNN-BILSTM	F1 Score	0.8458	0.8423	0.8753	0.8658	0.8239	0.8341	0.8377	0.8323	0.8039	0.8312	0.83923

Figure 42. Result of TF-IDF without Language Detection and Code-Switching