TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The coexistence of multiple languages in digital communication, commonly known as code-mixing, presents a unique challenge for offensive language detection systems. This study focuses on developing an advanced Code-Mix Offensive Language Detection System specifically tailored for Kannada-English code-mixed text. The objective is to create a robust model that effectively identifies and categorizes offensive content in the linguistic context of BERT (Bidirectional Encoder Representations from Transformers) [1] Leveraging the power of pre-trained contextual embeddings, BERT offers a deep understanding of semantic relationships within code-mixed sentences, enhancing the system's ability to discern offensive language nuances Kannada and English.

Key Algorithms Utilized:

- **BERT (Bidirectional Encoder Representations from Transformers):** Leveraging the power of pre-trained contextual embeddings, BERT offers a deep understanding of semantic relationships within code-mixed sentences, enhancing the system's ability to discern offensive language nuances
- **Multinomial Naive Bayes (MultinomialNB):** A probabilistic algorithm that excels in handling text classification tasks, MultinomialNB is employed to capture the probabilistic distribution of offensive language features in the code-mixed dataset.
- **Support Vector Machines (SVM) with Various Kernels (Linear, Polynomial, Sigmoid):** SVMs are utilized for their versatility in handling both linear and non-linear decision boundaries. Different kernel functions, including linear, polynomial, and sigmoid, are explored to optimize offensive language detection performance
- **Logistic Regression:** Employed to model the relationship between linguistic features and offensive language, linear regression provides insights into the linear dependencies within the code-mixed text.

- **Random Forest:** A powerful ensemble learning algorithm, Random Forest, is implemented to create a robust and accurate offensive language detection model by aggregating predictions from multiple decision trees.

By integrating these diverse algorithms, this Code-Mix Offensive Language Detection System aims to overcome the challenges posed by linguistic diversity, offering a comprehensive solution for monitoring and moderating offensive language in Kannada-English code-mixed content.

1.1 Overview or background and motivation

The motivation for enhancing abusive language detection stems from the pressing need to create safer online spaces. Abusive language, including hate speech, insults, and profanity, can lead to harmful consequences, such as cyberbullying, harassment, and the spread of misinformation. Detecting and mitigating abusive content is crucial for maintaining a healthy and inclusive digital environment.

- **Social Well-being:** Improving abusive language detection contributes to the overall well-being of individuals using online platforms. It helps create a more positive and respectful digital discourse.
- **Combatting Cyberbullying:** Abusive language often leads to cyberbullying, affecting mental health and causing emotional distress. Advanced detection methods aim to identify and prevent such harmful behavior.
- **Preventing Spread of Misinformation:** Abusive content can be a vehicle for spreading misinformation and fake news. Robust detection systems play a vital role in curbing the dissemination of false information.
- **Ensuring Inclusivity:** A safer online space is essential for fostering inclusivity and ensuring that diverse voices feel welcome and protected on digital platforms.

1.1.1 Problem Statement

The research focuses on advancing offensive language identification in code-mixed social media content, specifically within Kannada and English. The existing work outlines the intent to classify code-mixed comments/posts, emphasizing the urgency to address offensive content within low-resourced linguistic environments.

To enhance the problem statement, the study aims to delve deeper into the complexities of offensive post detection in code-mixed languages such as kanglish (kannada–English mixed) . The challenge lies in developing an innovative solution that not only accurately identifies offensive language but also adapts to the linguistic nuances and variations present in social media discourse.

1.2 Research Questions

- **Algorithmic Performance Comparison:**

  How do the performance metrics (accuracy, precision, recall) of BERT, MultinomialNB, SVM (linear, polynomial, sigmoid), Linear Regression, and Random Forest compare in the context of CodeMix Offensive Language Detection in Kannada-English?

- **Effectiveness in Handling Out-of-Vocabulary Words:**

  To what extent do these algorithms effectively handle out-of-vocabulary words, including code-mixed and misspelled terms, considering their impact on offensive language detection accuracy?

- **Contextual Understanding with BERT:**

  How does BERT contribute to the contextual understanding of offensive language in the Kannada-English code-mixed dataset, and how does it compare to traditional algorithms in this aspect?

- **Generalization and Adaptability:**

  How well do the chosen algorithms generalize and adapt to variations in offensive language usage across different contexts and topics within the Kannada-English code-mixed content?

1.2.1 Research Objectives

- **Developing an Accurate Offensive Language Detection Algorithm for Code-Mixed Texts:**

  Offensive language detection in code-mixed texts presents a unique set of challenges due to the inherent complexity of language mixing. The primary objective is to design a sophisticated algorithm that can navigate the intricacies of multiple languages within a single context.

- **Creating a Diverse Dataset of Code-Mixed Content for Training and Evaluation:**
  The foundation of a robust offensive language detection system lies in the quality and diversity of the training dataset. Our project meticulously curates a dataset that mirrors real-world scenarios of code-mixing across different languages and domains. This diversity is crucial to expose the algorithm to a wide array of linguistic patterns, ensuring its adaptability and effectiveness in varied contexts.

- **Evaluating the System's Performance in Terms of Precision, Recall, and F1-Score:**

A comprehensive evaluation is paramount to assessing the offensive language detection system's efficacy. We employ well-established metrics such as precision, recall, and F1-score to provide a nuanced understanding of the algorithm's performance. Precision measures the accuracy of offensive language predictions, recall gauges the system's ability to capture all instances of offensive language, and F1-score balances these metrics, considering the trade-off between precision and recall.

1.2.2 Research Significance

- **Preserving Cultural Nuances**: Dravidian languages have unique linguistic and cultural nuances. Studying offensive language in these languages helps in preserving and understanding the subtleties of cultural expression in digital communication.

- **Addressing Linguistic Diversity:** Dravidian languages often involve code-mixing, where different languages are used in combination. Researching offensive language in this context contributes to addressing linguistic diversity and challenges in multilingual communication.

- **Community Well-being:** Identifying offensive language is crucial for maintaining a positive online environment. It contributes to community well-being by preventing cyberbullying, harassment, and fostering respectful online interactions in Dravidian language communities.

- **Social Media Impact:** With the increasing use of social media in Dravidian regions, analyzing offensive language becomes pivotal for creating safer online spaces. This research helps in mitigating the negative impact of offensive content on social media platforms.

- **Language-Specific Solutions:** Dravidian languages face unique challenges, and research in offensive language identification provides language-specific solutions. Tailoring models and approaches to these languages ensures more accurate and culturally sensitive results.

1.3 Dataset Description

Bharathi Raja Chakravathi and et al.,[1] The development of a multilingual, manually annotated dataset for three under-resourced Dravidian languages generated from social media comments. The dataset was annotated for sentiment analysis and offensive language identification for a total of more than 60,000 YouTube comments. The dataset consists of around 7000 comments in Kannada-English. The data was manually annotated by volunteer annotators and has a high inter-annotator agreement

Table 1.1: Original Dataset Classes

Class 1	**Not offensive**
Class 2	**Offensive Targeted Insult Individual**
Class 3	**Offensive Targeted Insult Group**
Class 4	**Offensive Targeted Insult Other**
Class 5	**Offensive Untargeted**
Class 6	**Not Kannada**

Table 1.2: Sub-Set Classes Chosen from Original Dataset

Class 1	**Not Offensive**
Class 2	**Offensive**

in Krippendorff's alpha. The dataset contains all types of code-mixing phenomena since it comprises user-generated content from a multilingual country. We also present baseline experiments to establish benchmarks on the dataset using machine learning and deep learning methods. The dataset is available on Github and Zenodo.

In Table 1.1 shows the various classes involved in the original dataset

In Table 1.2 shows the classes that we have chosen to develop our project in order to concentrate only on Offensive and not-offensive Kannada-English Languages

Chapter 2

Literature Survey

Performing a Literature survey on Code-Mix Offensive Language Detection System in Kannada-English is essential for several reasons:

- Understanding Existing Approaches: A literature survey provides insights into existing methodologies, techniques, and models employed for Code-Mix Offensive Language Detection. This understanding helps in identifying the strengths and limitations of previous approaches.
- Benchmarking Models: Evaluating and comparing the performance of different models (BERT, MultinomialNB, SVM, Linear Regression, Random Forest) in the context of Kannada-English Code-Mix offensive language detection is crucial. This allows researchers to identify the most effective models for this specific language combination.
- Identifying Challenges: Literature reviews help in identifying challenges and gaps in the current research. It allows researchers to understand what issues have been addressed and what areas require further exploration, guiding the formulation of research questions.

The Literature Survey for this Project are:

Rajish Pandy and Jothi Prakash Singh [2] performed a comparative analysis of different Transformer-based language Mod- els pre-trained using unsupervised approaches. We have included the code-mixed models like HingBERT Mixed, mBERT, and non- code-mixed models like AlBERT, BERT, and RoBERTa for comparative analysis of code-mixed Hindi-English downstream tasks. We report state-of-the-art results on re- spective datasets using HingBERT-based models which are specifically pre-trained on real code-mixed text. Our HingBERT-based models provide significant improvements thus highlighting the poor performance of vanilla BERT models on code-mixed text.

Carmen Barroso and et al., [3] used Sarcasm which is the acerbic use of words to mock someone or something, mostly in a satirical way. Scandal or mockery is used harshly, often crudely and contemptuously, for destructive purposes in sarcasm. To extract the actual sentiment of a sentence for code-mixed language is complex because of the unavailability of sufficient clues for sarcasm. In this Paper, they have proposed a model consisting of Bidirectional Encoder Representations from Transformers (BERT) stacked with Long Short Term Memory (LSTM) (BERT-LSTM). A pre-trained BERT model is used to create em- bedding for the code-mixed dataset. These embedding vectors were used by an LSTM network consisting of a single layer to identify the nature of a sentence, i.e., sarcas- tic or non-sarcastic. The experiments show that the proposed BERT-LSTM model detects sarcastic sentences more effectively compared to other models on the code- mixed dataset, with an improvement of up to 6 percent in terms of F1-score.

Anirudh Srinivasan and et al., [4] performed a comparative analysis of different Transformer-based language Models pre-trained using unsupervised approaches. They have included the code-mixed models like HingBERT, HingRoBERTa, HingRoBERT -Mixed, mBERT, and non-code-mixed models like AlBERT, BERT, and RoBERTa for comparative analysis of code-mixed Hindi-English downstream tasks. Their report state of the art results on respective datasets using HingBERT-based models which are specifically pre-trained on real code-mixed text. Our HingBERT-based models provide signifi- cant improvements thus highlighting the poor performance of vanilla BERT models on code-mixed text.

Baraj B Kachru [5] describes the development of a multilingual, manual ally annotated dataset for three under-resourced Dravidian languages generated from social media comments. The dataset was annotated for sentiment analysis and offen- sive language identification for a total of more than 60,000 YouTube com- ments. The dataset consists of around 44,000 comments in Tamil-English, around 7000 comments in Kannada-English, and around 20,000 comments in Malayalam- English.

Abhishek Phaltankar and et al.,[6] Models such as mBERT and XLMR have shown success in solving Code-Mixed NLP tasks even though they were not exposed to such text during pre training. Code- Mixed NLP models have relied on using synthetically generated data along with nat- urally occurring data to improve their performance. Finetuning mBERT on such data improves it's code- mixed performance, but the benefits of using the different types of Code-Mixed data aren't clear. In this paper, we study the impact of fine- tuning with different types of code-mixed data and outline the changes that occur to the model during such fine tuning. Their findings suggest that using naturally occurring code-mixed data brings in the best performance improvement after fine tuning and that fine tuning with any type of code-mixed text improves the respon sivity of it's attention heads to code-mixed text inputs. There is a hierarchy oflanguages in a multilingual society, and that each language is as igneda functional role in amul- tilingual individual srestrictedorext endedsphere soflinguistic interaction. In South Asia, language dependency has-resulted in linguistic conver gence of two types,conver gence within the iner language circle, that is, within South Asian languages. outer linguistic imposition, or dependency on languages outside the South Asian language periphery.This type of convergence is seninthe Persianization and the Anglicization of the in ercircle languages. This study is concerned with an aspect of.the convergence with English' code-mixing, or the use of one or more languages for consistent transfer of linguistic units from one language into another, resulting in a new code of linguistic interaction. Code-mixing is role-dependent because the religious, social, economic, and regional characteristics of the participants are- crucial in understanding the event. It is function-dependent because the specialized use to which a language is being put determines code-mixing. The study explores the formal manisfestations and motivations for code-mixing;the aceptability constraintsonit,and its influence on South Asian languages.

Chapter 3

Project Design

3.1 Proposed Methodology

Figure 3.1 Shows the Data Block Diagram of the System which consists of mainly 6 steps namely:

- Data Collection and Annotation
- Data Translation
- Data Processing and Tokenization
- Pre-Trained BERT model
- Fine Tuning on Offensiive lnguage
- Model evaluation and Hyperparameter Tuning

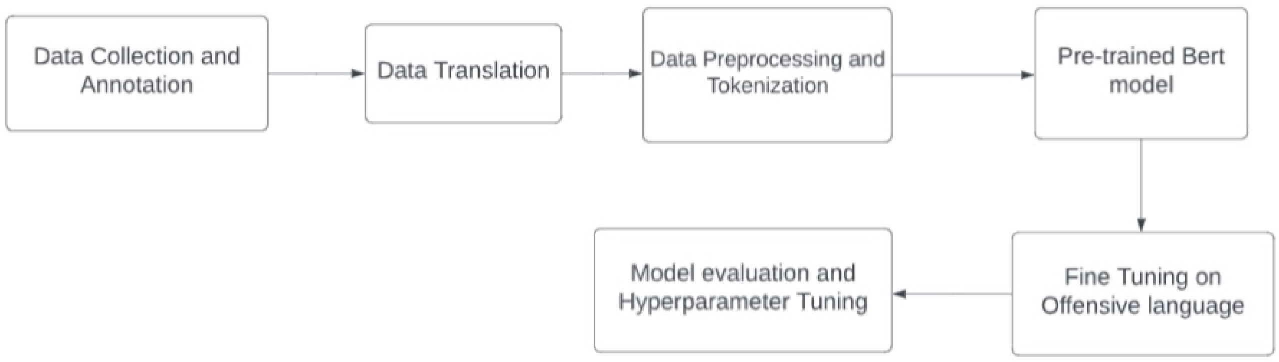

Data Collection and Annotation
Data Translation
Data Preprocessing and Tokenization
Pre-trained Bert model
Fine Tuning on Offensive language
Model evaluation and Hyperparameter Tuning

Figure 3.1: Data Block Diagram

- **Data Collection:** Objective: Gather a dataset comprising ,codemixed text containing offensive language. Rationale: The collection aims to have a diverse dataset that accurately represents codemixed language with offensive content. Diverse data enhances model robustness.

- **Translation Using Google Translate API:** Objective: Utilize the Google Translate API to translate the da taset into target languages.
- **Preprocessing Steps:** Objective: Clean, tokenize, and prepare the translated dataset for model training. Rationale: Data preprocessing ensures uniformity, removes noise, handles special characters, and standardizes the text format across languages for effective model learning.
- **Feature Engineering:** Objective: Extract relevant features from the translated and preprocessed codemixed text. Rationale: Techniques like TF-IDF, word embeddings, or language-specific features capture linguistic nuances and aid in representing the data effectively for model learning.
- **Model Selection:** Objective: Choose suitable algorithms (BERT, SVM, SGDC, LR, RF, MultinomialNB) for offensive language detection. Rationale: Each algorithm offers unique strengths (contextual understanding, statistical methods, etc.) that can address the complexity of offensive language in codemixed text.
- **Model Evaluation:** Objective:Assess the performance of selected models using specific metrics. Rationale: Evaluation metrics like accuracy, precision, recall, and F1-score quantify model effectiveness in detecting offensive language across various languages. Rationale Behind Methodology Choices: Diversity: The inclusion of multiple languages in the dataset and translation reflects the diverse nature of codemixed content online. Robustness: Preprocessing and feature engineering aim to create a standardized representation of codemixed text for robust model training. Versatility: The selection of multiple models allows for the evaluation of diverse approaches, enhancing the project's insights into effective offensive language detection in codemixed data.

Chapter 4

System Requirements

4.1 Hardware Reuirements

- Processor:

  Minimum: 1 GHz Recommended: 2 GHz or more

- RAM (Random Access Memory):

  Minimum: 4 GB Recommended: 8 GB or more

- Storage:

  Minimum: 60 GB of hard disk space

4.2 Software Requirements

- Operating System:

  Any modern operating system, such as Windows, Linux, or macOS

- Programming Language:

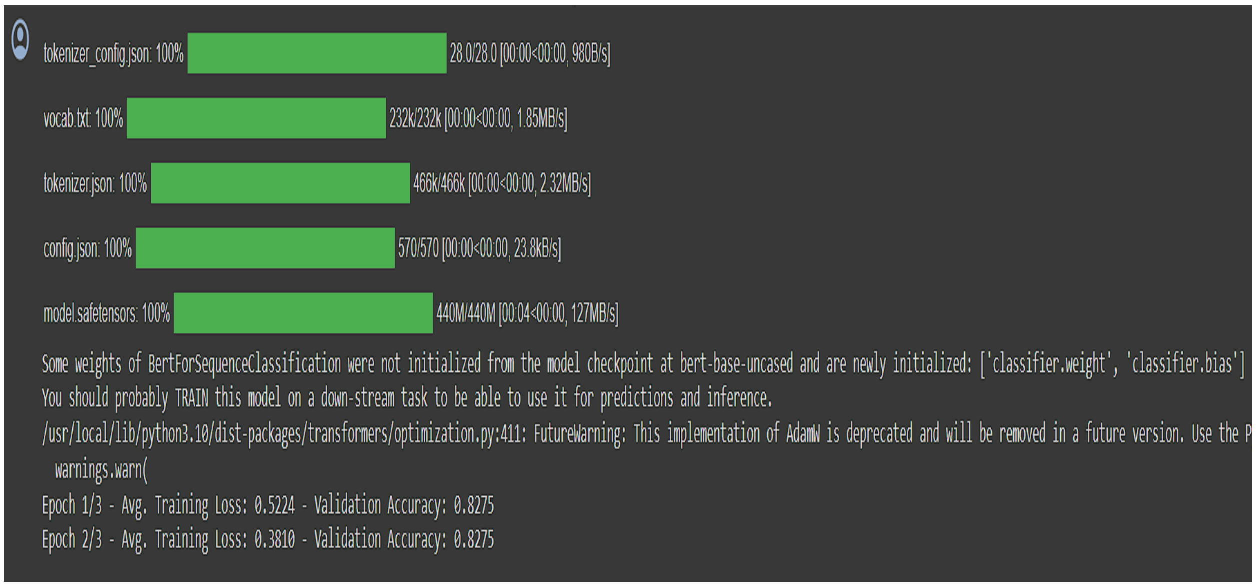
  Python is commonly used for natural language processing tasks.

4.3 Algorithms required

- Suport Vector Machine (SVM)-Linear,Polynomial,Sigmoid
- Logistic Regression
- Random Forest
- Stochastic Gradient Descent Classifier (SGDC)
- Bidirectional Encoder Representations from Transformers (BERT)
- Multinomial Naive Bayes Clasifier (MultinomialNB)

Chapter 5

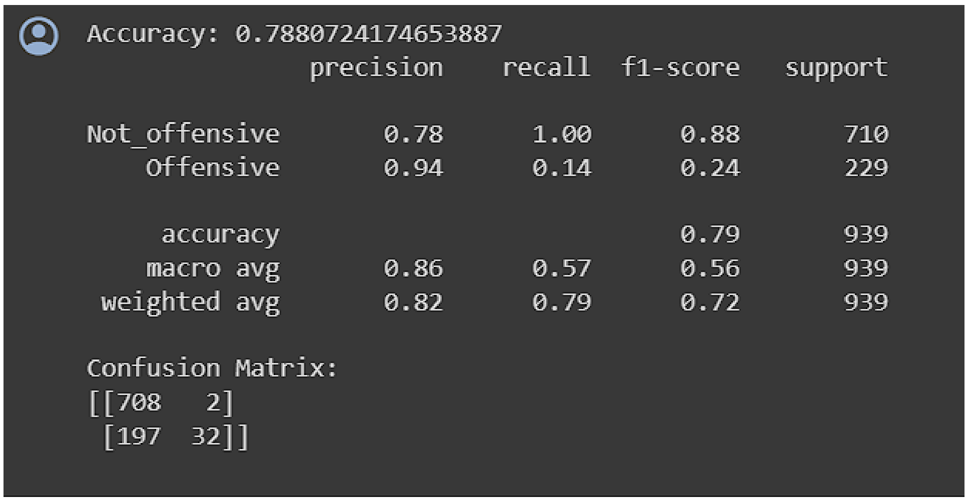
Results



```
tokenizer_config.json: 100%                28.0/28.0 [00:00<00:00, 980B/s]
vocab.txt: 100%                232k/232k [00:00<00:00, 1.85MB/s]
tokenizer.json: 100%                466k/466k [00:00<00:00, 2.32MB/s]
config.json: 100%                570/570 [00:00<00:00, 23.8kB/s]
model.safetensors: 100%                440M/440M [00:04<00:00, 127MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the P
  warnings.warn(
Epoch 1/3 - Avg. Training Loss: 0.5224 - Validation Accuracy: 0.8275
Epoch 2/3 - Avg. Training Loss: 0.3810 - Validation Accuracy: 0.8275
```

Figure 5.1: BERT

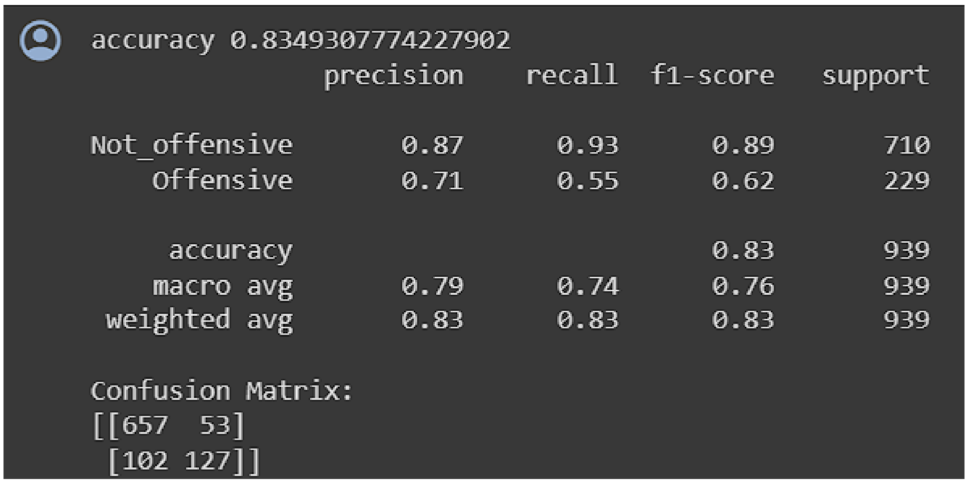
In Figure 5.1 shows an accuracy of 79 percrnt, showing competitive precision and recall scores, particularly for the "Not-Off" class. It captures bidirectional contextual information to understand the context of words in a sentence.



```
Accuracy: 0.7880724174653887
               precision    recall  f1-score   support

Not_offensive       0.78      1.00      0.88       710
    Offensive       0.94      0.14      0.24       229

     accuracy                           0.79       939
    macro avg       0.86      0.57      0.56       939
 weighted avg       0.82      0.79      0.72       939

Confusion Matrix:
[[708   2]
 [197  32]]
```

Figure 5.2: MultinomialNB

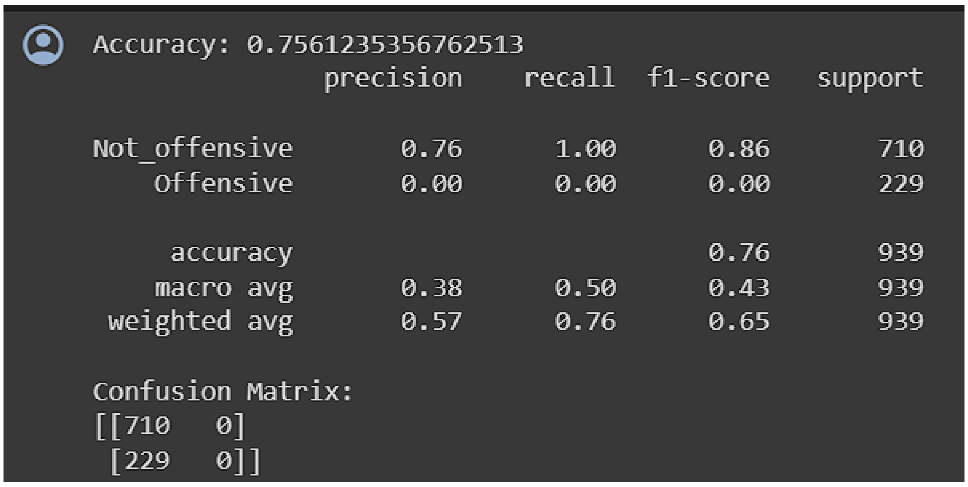
In Figure 5.2 MultinomialNB exhibited strong performance with an accuracy of 78 percent, high precision for both classes, and significant recall for the ”Off” class.Multinomial Naive Bayes assumes that the features follow a multinomial distribution and is often employed for text classification, where the input features represent word counts or term frequencies in documents.



```
accuracy 0.8349307774227902
               precision    recall  f1-score   support

Not_offensive       0.87      0.93      0.89       710
    Offensive       0.71      0.55      0.62       229

     accuracy                           0.83       939
    macro avg       0.79      0.74      0.76       939
 weighted avg       0.83      0.83      0.83       939

Confusion Matrix:
[[657  53]
 [102 127]]
```

Figure 5.3: Logstic Regression

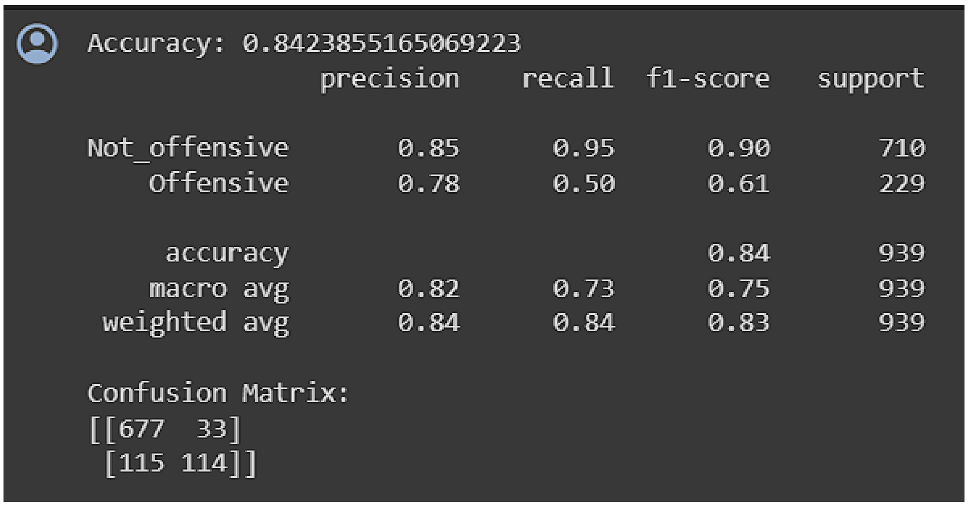
In Figure 5.3 Logistic Regression achieved a high overall accuracy of 83 percent with notable precision,recall, and F1 scores. It Applies the logistic (sigmoid) function to model the probability of a binary outcome and it is suitable for problems where the dependent variable is categorical



```
Accuracy: 0.7561235356762513
               precision    recall  f1-score   support

Not_offensive       0.76      1.00      0.86       710
    Offensive       0.00      0.00      0.00       229

     accuracy                           0.76       939
    macro avg       0.38      0.50      0.43       939
 weighted avg       0.57      0.76      0.65       939

Confusion Matrix:
[[710   0]
 [229   0]]
```

Figure 5.4: Random Forest

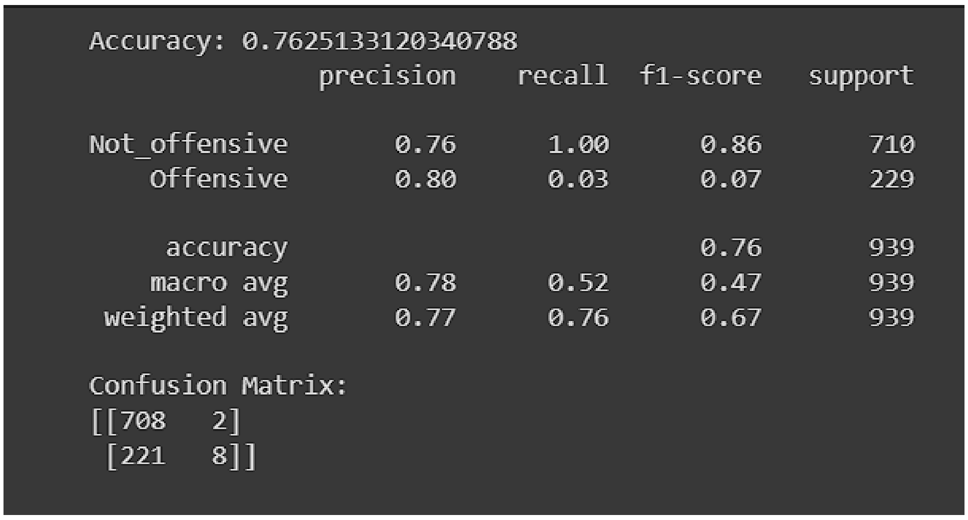
In Figure 5.4 Random forest acquired lower accuracy at 75 percent, particularly due to zero precision and recall for the ”Off” class.Random Forest is an ensemble learning method that builds multiple decision trees during training,here each tree ”votes” for the most popular class, and the mode of the classes is the final prediction.



```
Accuracy: 0.8423855165069223
               precision    recall  f1-score   support

Not_offensive       0.85      0.95      0.90       710
    Offensive       0.78      0.50      0.61       229

     accuracy                           0.84       939
    macro avg       0.82      0.73      0.75       939
 weighted avg       0.84      0.84      0.83       939

Confusion Matrix:
[[677  33]
 [115 114]]
```

Figure 5.5: SVM Linear

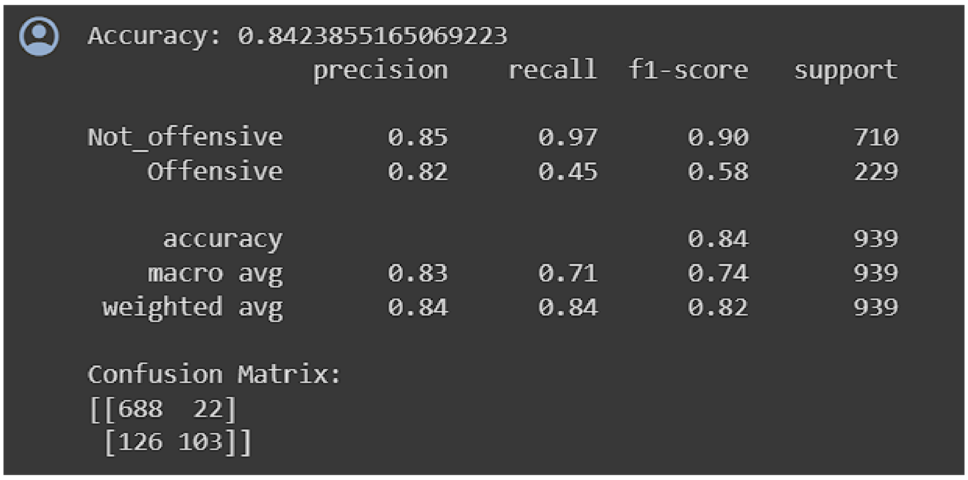
In Figure 5.5 SVM Linear Stood out with an accuracy of 84 percent , combining high precision, recall, and F1 scores for both classes. This SVM is a supervised machine learning algorithm used for classification or regression tasks.



```
Accuracy: 0.7625133120340788
               precision    recall  f1-score   support

Not_offensive       0.76      1.00      0.86       710
    Offensive       0.80      0.03      0.07       229

     accuracy                           0.76       939
    macro avg       0.78      0.52      0.47       939
 weighted avg       0.77      0.76      0.67       939

Confusion Matrix:
[[708   2]
 [221   8]]
```

Figure 5.6: SVM Polynomial

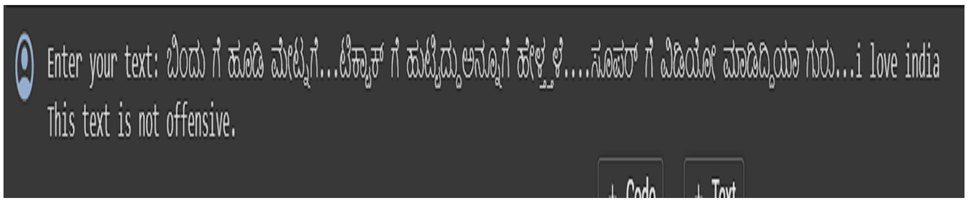
In Figure 5.6 SVM Polynomial shows Lower accuracy of 76 percent, primarily due to low precision and recall for the "Off" class.It utilizes a hyperplane to separate data points into different classes.



```
Accuracy: 0.8423855165069223
               precision    recall  f1-score   support

Not_offensive       0.85      0.97      0.90       710
    Offensive       0.82      0.45      0.58       229

     accuracy                           0.84       939
    macro avg       0.83      0.71      0.74       939
 weighted avg       0.84      0.84      0.82       939

Confusion Matrix:
[[688  22]
 [126 103]]
```

Figure 5.7: SVM Sigmoid

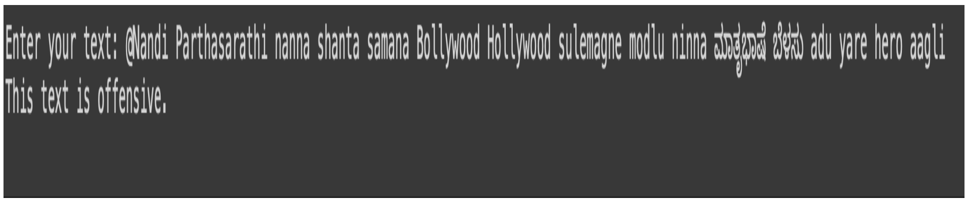
In Figure 5.7 SVM Sigmoid achived strong performance with an accuracy of 84 percent , particularly notable for precision and recall in the "Not-Off" class. Kernel function sigmoid can be applied for non-linear mapping



```
Enter your text: ಬಿಂದು ಗೆ ಹೂಡಿ ಮೇಟ್ನಿಗೆ...ಟಿಕ್ಟಾಕ್ ಗೆ ಹುಟ್ಟಿದ್ದುಅನ್ನೋಗೆ ಹೇಳ್ತ್ಳೆ....ಸೂಪರ್ ಗೆ ವಿಡಿಯೋ ಮಾಡಿದ್ದಿಯಾ ಗುರು...i love india
This text is not offensive.
```

Figure 5.8: sample 1

In Figure 5.8 is an sample which is offensive but shows not offensive because of accuracy level in the project



```
Enter your text: @Nandi Parthasarathi nanna shanta samana Bollywood Hollywood sulemagne modlu ninna ಮಾತೃಭಾಷೆ ಬೆಳಸು adu yare hero aagli
This text is offensive.
```

Figure 5.9: sample 2

In Figure 5.9 shows the sample which is an offensive language, it detects properly

Table 5.1: Accuracy Table for the Desired Project

	Precision		F1 Score		Recall		Accuracy
	Not-Off	Off	Not-Off	Off	Not-Off	Off	
LR	0.87	0.71	0.89	0.62	0.93	0.55	0.83
RF	0.76	0.00	0.86	0.00	1.00	0.00	0.75
MNB	0.78	0.94	0.88	0.24	1.00	0.24	0.78
SGDC	0.87	0.68	0.89	0.63	0.91	0.59	0.83
SVM LINEAR	0.85	0.78	0.90	0.61	0.95	0.50	0.84
SVM POLY	0.76	0.80	0.86	0.07	1.00	0.03	0.76
SVM SIGMOID	0.85	0.82	0.90	0.58	0.97	0.45	0.84
BERT	0.81	0.73	0.88	0.40	0.97	0.28	0.79

In Table 5.1 shows the Accuracy, Precission and Recall of each Algorithms with respect to Not-Offensive and Offensive classes

Chapter 6

Conclusion

The CodeMix Offensive Language Detection System in Kannada-English, employing various algorithms such as BERT, MultinomialNB, SVM (linear, polynomial, sigmoid), Linear Regression, and Random Forest, showcases a multifaceted approach to tackle the challenges of identifying offensive content in code-mixed text.

- **Algorithmic Diversity:**

  The inclusion of a diverse set of algorithms, including machine learning classics (MultinomialNB, SVM, Logistic Regression, Random Forest) and advanced deep learning models (BERT), brings versatility to the CodeMix offensive language detection framework .

- **Performance Insights:**
  The SVM based classifiers (Linear and Sigmoid) have demonstrated promising results upto 84 percent in the Kannada-English CodeMix shared task, emphasizing the effectiveness of these algorithms in handling offensive language detection in Kannada-English code-mixed data

In conclusion, the CodeMix Offensive Language Detection System, incorporating a mix of traditional and state-of-the-art algorithms, emerges as a comprehensive solution for addressing the complexities of offensive language identification in Kannada-English code-mixed content.

APPENDIX A

Program

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials= GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

downloaded = drive.CreateFile({'id':'1ekpeYQydpd1nhVXAIu77YBDi_rHKjFq8'})
downloaded.GetContentFile('original_dataset.csv')

from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
from googletrans import Translator
import time
import pandas as pd
import torch
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from transformers import BertTokenizer, BertForSequenceClassification
from torch.utils.data import TensorDataset, DataLoader, RandomSampler
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split

# Function to translate a chunk of data and save it to a CSV file
def tas(dataframe, start_index, end_index, translator, output_file):
    chunk = dataframe.loc[start_index:end_index]  # Select a chunk of data
    translated_texts = []

    # Translate the chunk
    for index, row in chunk.iterrows():
      translated = translator.translate(row['codemix_text'], dest='en').text
        translated_texts.append(translated)
        time.sleep(1)
    # Add translated texts to the chunk
    chunk['translated_text'] = translated_texts

    # Save translated chunk to a CSV file
    mode = 'w' if start_index == 0 else 'a'  # If it's the first chunkfile
    header = start_index == 0  # Write header only for the first chunk

    chunk.to_csv(output_file, mode=mode, header=header, index=False)

# Load your dataset (assuming it's a CSV file named 'original_dataset.csv')
data = pd.read_csv('original_dataset.csv')

# Split the dataset into chunks of 1000 lines and translate each chunk
translator = Translator()

start_index = 0
chunk_size = 1000
end_index = min(chunk_size, len(data))

output_file = 'translated_output.csv'

while start_index < len(data):
    translate_and_save(data, start_index, end_index, translator, output_file)
    start_index = end_index
    end_index = min(start_index + chunk_size, len(data))
```

```
df_train = pd.read_csv(r'original_dataset.csv')

# Mapping categorical labels to integers
label_dict = {'Not_offensive': 0, 'Offensive': 1}
df_train['offensive_label'] = df_train['offensive_label'].map(label_dict)

# Fill NaN values with -1
df_train['offensive_label'] = df_train['offensive_label'].fillna(-1).astype(int)

label_list_train = list(df_train['offensive_label'])
word_list_train = list(df_train['codemix_text'])

# Printing for verification
print(word_list_train)
print(label_list_train)

train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)

train_texts = train_data['codemix_text'].values.astype('U')
train_labels = train_data['offensive_label'].values

test_texts = test_data['codemix_text'].values.astype('U')
test_labels = test_data['offensive_label'].values

nb = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 4), min_df=1)),
    ('tfidf', TfidfTransformer(sublinear_tf=True)),
    ('clf', MultinomialNB()),
])



sgd = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 2), min_df=1)),
    ('tfidf', TfidfTransformer(sublinear_tf=True)),
    ('clf', SGDClassifier(loss='hinge', penalty='l2', alpha=1e-4, random_state=42
])



logreg = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 3), min_df=1)),
    ('tfidf', TfidfTransformer(sublinear_tf=True)),
    ('clf', LogisticRegression(n_jobs=1, C=1e3)),
])


rf = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 3), min_df=1)),
    ('tfidf', TfidfTransformer(sublinear_tf=True)),
    ('clf', RandomForestClassifier(max_depth=10, random_state=65)),
])



svm_sigmoid = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 4), min_df=1)),
    ('tfidf', TfidfTransformer(sublinear_tf=True)),
    ('clf', SVC(kernel='sigmoid'))  # Using a sigmoid kernel
])

svm_poly = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 4), min_df=1)),
    ('tfidf', TfidfTransformer(sublinear_tf=True)),
    ('clf', SVC(kernel='poly', degree=3))
svm_linear = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 4), min_df=1)),
    ('tfidf', TfidfTransformer(sublinear_tf=True)),
    ('clf', SVC(kernel='linear'))  # Using a linear kernel
])


model_name = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(
    model_name,
    num_labels=len(set(translated_df['offensive_label'])),
    output_attentions=False,
    output_hidden_states=False
)



# Create TensorDataset for train and validation sets
```

```
train_dataset = TensorDataset(
    torch.tensor(train_encodings['input_ids']),
    torch.tensor(train_encodings['attention_mask']),
    torch.tensor(train_labels)
)
val_dataset = TensorDataset(
    torch.tensor(val_encodings['input_ids']),
    torch.tensor(val_encodings['attention_mask']),
    torch.tensor(val_labels)
)

# Define batch size and create DataLoaders
batch_size = 16
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Define optimizer and learning rate
optimizer = AdamW(model.parameters(), lr=2e-5)

# Training loop
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
epochs = 3  # You can adjust the number of epochs
for epoch in range(epochs):
    model.train()
    total_loss = 0
    for batch in train_loader:
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        optimizer.step()

    avg_train_loss = total_loss / len(train_loader)

    # Validation loop
    model.eval()
    val_preds = []
    val_true = []
    for batch in val_loader:
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device)

        with torch.no_grad():
            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs.logits

        val_preds.extend(torch.argmax(logits, axis=1).cpu().numpy())
        val_true.extend(labels.cpu().numpy())

def predict_offensiveness(input_text, trained_model):
    # Make a prediction using the trained model
    predicted_label = trained_model.predict([input_text])[0]
    return predicted_label

# Assuming ‘svm_linear‘ is your trained SVM model
user_input = input("Enter your text: ")

# Call the prediction function
result = predict_offensiveness(user_input, svm_linear)

# Mapping predicted labels to their corresponding meanings
label_meanings = {
    'Offensive': 'This text is offensive.',
    'Not_offensive': 'This text is not offensive.',
    'not-Kannada': 'This text is not in Kannada.'
}

# Display the predicted result
if result in label_meanings:
    print(label_meanings[result])
else:
    print("Label not recognized.")
```

References

[1] Bharathi Raja Chakravarthi, Ruba Priyadharshini, Vigneshwaran Muralidaran, and Navya Jose. Dravidiancodemix: sentiment analysis and offensive language identification dataset for dravidian languages in code-mixed text. 2022.

[2] Rajnish Pandey and Jyoti Prakash Singh. Bert-capsule model for cyberbullying detection in code-mixed indian languages. 2022.

[3] Acedo Francisco Jose, Carmen Barroso, and Cristobal Casanueva. Bert-lstm model for sarcasm detection in code-mixed social media post. 2006.

[4] Sebastin Santy, Anirudh Srinivasan, and Monojit Choudhury. *BERTTologi Code Mix.* 2021.

[5] Braj B. Kachru. Towards structuring code-mixing. 2020.

[6] Aryan Patil, Varad Patwardhan, Abhishek Phaltankar, Gauri Takawane, and Raviraj Joshi. *Comparative Study of Pre-Trained BERT Models for Code-Mixed Hindi-English Data.* 2023 IEEE 8th International Conference for Convergence in Technology (I2CT), 2023.