OFFENSIVE LANGUAGE IDENTIFICATION IN CODE-MIXED TEXT

A Project Report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

By

A. Bharath Teja - 222010326013
Y. Tharun Reddy - 222010326020
P. Ruchith Reddy - 222010326032
R. Srihith - 222010326052

Under the Supervision of

Mr. Y. Prakash Babu
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GANDHI INSTITUTE OF TECHNOLOGY AND MANAGEMENT(GITAM)(Declared as Deemed-to-be-University u/s 3 of UGC Act 1956)
HYDERABAD, TELANGANA

GANDHI INSTITUTE OF TECHNOLOGY AND MANAGEMENT (GITAM)

(Declared as Deemed-to-be-University u/s 3 of UGC Act 1956)

HYDERABAD CAMPUS

DECLARATION

I/We, hereby declare that the project report entitled "**OFFENSIVE LANGUAGE IDENTIFICATION IN CODE MIXED TEXT**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Place-HYDERABAD **Date-30-10-2023**

Registration No.	**Name**	**Signature**
222010326013	A. Bharath Teja	
222010326020	Y. Tharun Reddy	
222010326032	P. Ruchith Reddy	
222010326052	Rachakonda. Srihith	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(DEEMED TO BE UNIVERSITY)

HYDERABAD CAMPUS

CERTIFICATE

This is to certify that the project report entitled "**Offensive Language Identification In Code-Mixed Text**" is a bonafide record of work carried out by A. Bharath Teja(222010326013), Y.Tharun Reddy(222010326020), P.Ruchith Reddy(222010326032) and Rachakonda Srihith(222010326052) students submitted in partial fulfillment of requirements for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	**Project Coordinator**	**Head of Department**
Mr. Y. PRAKASH BABU	**Dr. PRADEEP**	**Mr. K. S. SUDEEP**
Assistant Professor	**Assistant Professor**	**Professor & HOD**
Dept. of CSE	**Dept. of CSE**	**Dept. of CSE**

ACKNOWLEDGEMENT

Our project report would not have been successful without the help of several people. We would like to thank the personalities who were part of our seminar in numerous ways, those who gave us outstanding support from the birth of the seminar.

We are extremely thankful to our honourable Pro-Vice-Chancellor, **Prof. D. Sambasiva Rao**, for providing the necessary infrastructure and resources for the accomplishment of our seminar.We are highly indebted to **Prof. N. Seetharamaiah**, Associate Director, School of Technology, for his support during the tenure of the seminar.

We are very much obliged to our beloved **Prof. K. S. SUDEEP**, Head of the Department of Computer Science & Engineering, for providing the opportunity to undertake this seminar and encouragement in the completion of this seminar.

We hereby wish to express our deep sense of gratitude to **Dr. PRADEEP**, Project Coordinator, Department of Computer Science and Engineering, School of Technology and to our guide, **Mr. Y. PRAKASH BABU**, Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the project report.

We are also thankful to all the Computer Science and Engineering department staff members who have cooperated in making our seminar a success. We would like to thank all our parents and friends who extended their help, encouragement, and moral support directly or indirectly in our seminar work.

Sincerely,
A. Bharath Teja - 222010326013
Y. Tharun Reddy - 222010326020
P. Ruchith Reddy - 222010326032
R. Srihith - 222010326052

Abstract

In the era of digital communication, fostering respectful online interactions and content moderation has become an imperative. This project addresses this need by presenting a comprehensive approach to the detection of offensive language in the unique context of code-mixed Kannada-English text, specifically within YouTube comments.

Our primary objective is to develop a robust offensive language detection system, leveraging natural language processing (NLP) techniques. To achieve this, we employ a combination of N-gram features and BERT embeddings, taking into account the nuanced characteristics of code-mixed language. Our focus on mixed-language content recognises the diverse linguistic landscape of digital communication.

The project benefits from a meticulously curated dataset of code-mixed Kannada-English YouTube comments, ensuring that the models are trained and evaluated on contextually rich and relevant data. This dataset is essential for the success of our offensive language detection models.

Innovation is a cornerstone of our approach, with the introduction of an attention-based mechanism tailored to capture the intricacies of offensive language nuances present in code-mixed text. We believe that understanding the nuances of offensive language in mixed-language content is crucial to successful detection.

The scope of the project is specific: we concentrate solely on the identification of offensive language within code-mixed Kannada-English text in YouTube comments. Our focus is on detection, without delving into broader sentiment analysis or user interactions.

Table Of Contents :

Introduction:

In today's digital age, social media platforms have become a prominent space for communication, expression, and interaction. However, this widespread online presence

has brought to light challenges related to the quality of interactions and content shared. Offensive language and inappropriate content have become persistent concerns, leading to the need for robust systems that can automatically detect and mitigate such instances. This project embarks on the journey of developing a sophisticated offensive language detection system, specifically tailored for code-mixed Kannada-English text in You tube-Comments on You tube, using Natural Language Processing (NLP) techniques.So does the prevalence of offensive language and inappropriate content, resulting in persistent concerns for digital communities and platform providers. The imperative to maintain respectful and constructive online spaces has never been more critical.It is within this context that our project takes flight, embarking on a mission to develop a sophisticated offensive language detection system. Our focus is acutely tuned to the complexities of code-mixed Kannada-English text within YouTube comments, a setting that often blurs the linguistic lines and presents unique challenges for content moderation. We aim to tackle this challenge through the lens of Natural Language Processing (NLP), a field of artificial intelligence that equips machines to understand, interpret, and respond to human language.The context of code-mixed language, where individuals seamlessly blend Kannada and English in their online conversations, poses distinctive hurdles for content moderation and offensive language detection. Within this linguistic fusion, the subtle interplay between languages, the cultural context, and the varying intensity of offensive language can pose unique challenges. Recognizing this, our project strives to tailor an offensive language detection system to navigate the intricacies of code-mixed conversations.The significance of our endeavor extends far beyond the scope of mere technological innovation. It delves deep into the realm of social dynamics and content moderation on digital platforms. The goal is not just to develop an advanced technological solution but to contribute to the creation of a safer, more inclusive, and respectful online environment.Our project is driven by the belief that the power of NLP can be harnessed to distinguish between offensive and non-offensive language effectively. By taking on the challenge of code-mixed Kannada-English text within YouTube comments, we aim to provide a foundation for building content moderation systems that can adapt to the nuances of mixed-language content.As the digital landscape continues to evolve, so must our approaches to maintaining respectful and safe online spaces.

The Context of Code-Mixed Language:

In multilingual societies, individuals often communicate in code-mixed language, which involves seamlessly blending two or more languages within a single conversation or text. This linguistic phenomenon poses a unique challenge for existing offensive language detection systems, as the contextual nuances and interplay between languages can impact the accuracy of detection. The Kannada-English language, rich in heritage and widely spoken, often intertwines with other languages in online conversations. Therefore, this project focuses on developing an offensive language detection system that can navigate the intricacies of code-mixed Kannada-English text on the Youtube platform.

In multilingual societies, individuals often engage in code-mixing, a linguistic phenomenon where they seamlessly blend two or more languages within a single conversation or text. This intricate practice is particularly relevant in regions where multiple languages coexist, and people routinely switch between them in their everyday communication. The challenge arises when we consider the effectiveness of existing offensive language detection systems in such a context. These systems are primarily designed to identify offensive content based on specific language patterns or keywords, making them less accurate when dealing with code-mixed language. The contextual nuances and interplay between languages can lead to false positives or negatives, thus posing a unique challenge.

This challenge is specifically addressed in the context of code-mixed Kannada-English language, which is prevalent in regions like Karnataka, India. Kannada, a language rich in heritage, is widely spoken in this region. In online conversations, it often intertwines with English and sometimes other languages, creating a dynamic linguistic landscape. This phenomenon can make it difficult for automated systems to accurately detect and filter offensive language. As a result, a project has been initiated to develop an offensive language detection system tailored to the nuances of code-mixed Kannada-English text. The focus of this project is to enhance the accuracy of identifying offensive content, particularly on the YouTube platform. Given the platform's popularity for sharing video content and fostering discussions through user-generated comments, such a system would be invaluable in maintaining a respectful and inclusive online environment in multilingual societies where code-mixing is a common practice.

The Role of Natural Language Processing (NLP):

The Role of Natural Language Processing (NLP) in the context of this project is pivotal and transformative. NLP is a crucial field within Artificial Intelligence that focuses on enabling machines to understand, analyze, and generate human language. In the specific context of code-mixed Kannada-English language, NLP plays a central role in addressing the complexities of linguistic patterns, cultural contexts, and linguistic variations that are inherent to such multilingual communication.NLP techniques provide the foundation for building an advanced model capable of comprehending the intricate linguistic nuances present in code-mixed conversations. This involves the machine's ability to recognize the interplay between Kannada and English, understand the contextual subtleties, and interpret the unique expressions that emerge in this linguistic fusion.One of the primary objectives of the project is to leverage the power of NLP to develop a robust offensive language detection system. This system is designed to effectively distinguish offensive or inappropriate language from legitimate and respectful expressions within code-mixed conversations. By doing so, it aims to create a safer and more inclusive online environment, particularly on the YouTube platform where user-generated content and discussions often involve code-mixed language.Natural Language Processing has emerged as a vital field within Artificial Intelligence, enabling machines to comprehend, analyze, and generate human language. Leveraging NLP techniques, this project aims to build an advanced model capable of understanding the complex linguistic patterns, cultural contexts, and linguistic variations present in code-mixed Kannada-English Language. By harnessing the power of NLP, we seek to develop a system that can effectively distinguish offensive language from legitimate expressions, fostering a more inclusive and respectful online environment.

Elevating the YouTube Experience: The Role of Offensive Language Identification:

Promoting a Safer YouTube Community: Offensive language identification on YouTube is a crucial tool in maintaining a safe and inclusive environment for content creators and viewers. By swiftly recognizing and addressing offensive content, these systems contribute to reducing harm and preventing cyberbullying, thus enhancing the overall safety of the platform.

Respecting Diverse Cultures and Languages: YouTube is a global platform with content in various languages and cultures. Offensive language identification systems, when culturally sensitive, ensure that content moderation respects the nuances and subtleties of different languages and cultures. This, in turn, creates a more inclusive and respectful space where diverse linguistic and cultural identities are celebrated.

Enhancing Communication and Collaboration: Misunderstandings and miscommunication can hamper the quality of online interactions. Offensive language identification not only helps in removing harmful content but also plays a role in mitigating miscommunication. This fosters more effective and respectful interactions, improving the overall quality of discussions and collaborations on YouTube.

Informing Content Creation and Policy: The study of offensive language on YouTube provides valuable insights into evolving online language use and its impact on community behavior. These insights inform content creators and platform policies, allowing for a more informed and proactive approach to managing content and fostering a respectful digital space.

Efficient Content Moderation: YouTube hosts an enormous volume of content. Automating the identification of offensive language through machine learning models significantly lightens the burden on human moderators. This not only enhances content moderation but also ensures a consistent and efficient approach to identifying and addressing objectionable content, making the platform a safer and more respectful space.

Timely Response for a Positive User Experience: Real-time offensive language recognition is pivotal for maintaining a positive user experience. Harmful content can be identified and acted upon promptly, preventing its spread and limiting the potential harm it can cause.
This timely response contributes to a more enjoyable and respectful user experience on YouTube, encouraging users to continue engaging with the platform.Timely Response for a Positive User Experience: Real-time offensive language recognition on YouTube enables a proactive approach to content moderation. Harmful content can be swiftly identified, flagged, and removed, preventing its further dissemination. This not only protects users from harm but also contributes to a more positive and respectful user experience on the platform.In sum, offensive language identification on YouTube is a multifaceted tool that plays a vital role in creating a safer, more inclusive, and respectful community for content creators and viewers. It promotes safety, cultural respect, effective communication, informed content creation, efficient content moderation, and a positive user experience on this global video-sharing platform.

Challenges in Offensive Language Identification:

Linguistic Diversity:
The linguistic diversity within Dravidian languages is characterized by intricate structures and rich vocabularies. This complexity is further compounded when Dravidian languages are mixed with English and other languages in code-mixing.Accurately defining and identifying offensive language becomes challenging in the face of this myriad of language combinations that occur in Dravidian code-mixed social media text. The linguistic diversity within Dravidian languages, such as Tamil, Malayalam, and Kannada, is known for its intricate grammatical structures and rich vocabulary. This complexity is exacerbated when these languages are interwoven with English and other languages in code-mixing. The result is a linguistic landscape where multiple languages coexist, creating a complex web of expressions and variations. Identifying offensive language within this intricate linguistic tapestry becomes a challenging task, as it requires algorithms and models to decipher and distinguish offensive content accurately across a wide range of language combinations.

Nuances and Sarcasm:
Offensive language identification in Dravidian code-mixed social media text poses challenges related to nuances and sarcasm.
It goes beyond straightforward language analysis and often involves deciphering subtle layers of meaning, sarcasm, irony, and cultural references.
Elaboration: Recognizing offensive language is not merely a matter of analyzing the language used; it also involves understanding the nuances, sarcasm, and cultural references present in the text. In the context of code-mixed Dravidian social media text, where multiple languages are at play, this challenge is further amplified. The identification systems must be adept at grasping the subtle layers of meaning and understanding the intricate web of linguistic expressions and cultural references to ensure accurate detection of offensive content.

Data Scarcity:
Offensive language identification faces a significant challenge due to the scarcity of suitable datasets for Dravidian code-mixed social media text.Dravidian languages often lack annotated offensive content, making it challenging to train and evaluate models effectively. Developing accurate models for offensive language identification in Dravidian code-mixed content is hindered by the shortage of suitable datasets. The process of annotating offensive content demands linguistic and cultural expertise, which is often scarce. Understanding and identifying offensive language within the intricacies of code-mixed text requires a deep understanding of the languages involved and their sociocultural contexts. Despite these challenges, addressing data scarcity is essential for the development of culturally sensitive and accurate models for offensive language identification in Dravidian code-mixed content.

Context Sensitivity:

The recognition of offensive language is closely tied to the surrounding context.
In the context of Dravidian code-mixed social media text, which exhibits rich linguistic diversity, understanding and considering the broader context becomes imperative.In the task of identifying offensive language, context sensitivity is crucial. The accurate recognition of offensive language is highly dependent on understanding the surrounding context. In Dravidian code-mixed social media text, where multiple languages and linguistic variations coexist, the importance of context

sensitivity is further amplified. Distinguishing between offensive and non-offensive expressions in these dynamic conversations necessitates taking into account the broader context and the linguistic landscape in which the text is embedded.

Complexity of Code-Mixing:

The complexity of code-mixing arises from the seamless blending of Dravidian languages, English, and other regional languages within social media text.
Algorithms must possess the ability to interpret and analyze text that transitions fluidly between different languages, addressing the unique challenges posed by the intricate multilingual nature of Dravidian code-mixed communication.Dravidian code-mixed social media text is characterized by the fluid integration of Dravidian languages, English, and other regional languages. This complexity poses a significant challenge for identifying abusive language accurately. Algorithms and models used for offensive language identification must be capable of interpreting and analyzing text that transitions seamlessly between different languages. They need to address the unique challenges posed by the intricate multilingual nature of Dravidian code-mixed communication.

Cultural Sensitivity:

Social factors and cultural norms significantly influence the perception of offensive language.Models used for identifying offensive language must be attuned to these cultural and contextual sensitivities to ensure accurate and respectful content moderation.Offensive language often carries a cultural dimension, and what is considered offensive can vary widely across different sociocultural contexts. Words or phrases that may be innocuous in one context can be highly offensive in another. Models used for identifying offensive language in Dravidian code-mixed text must be culturally sensitive, recognizing the specific cultural and contextual nuances that influence the perception of offensive language. This cultural sensitivity is essential to ensure that content moderation is accurate and respectful, considering the diverse cultural backgrounds of users and their online interactions.

These challenges highlight the intricacies and complexities involved in identifying offensive language in the context of Dravidian code-mixed social media text. Tackling these challenges requires the development of advanced models and algorithms that are culturally sensitive, context-aware, and capable of understanding the nuances of multilingual and multicultural communication.

Project Objectives and Methodologies:

The project's primary objective is to develop an advanced offensive language detection system tailored for code-mixed Kannada-English text on the YouTube platform. To achieve this goal, the project adopts a multi-faceted approach, combining cutting-edge Natural Language Processing (NLP) techniques with deep learning methodologies. This approach is essential to effectively address the challenges posed by code-mixed language, where the interplay of multiple languages can make the detection of offensive content particularly complex.

Advanced NLP Techniques: The project harnesses state-of-the-art NLP techniques to understand and process the intricate linguistic patterns found in code-mixed Kannada-English text. NLP forms the foundation for language understanding, enabling the model to comprehend the context and meaning behind the words used in conversations.

Deep Learning and Attention Mechanisms: Deep learning methods, particularly recurrent neural networks (RNNs) and attention mechanisms, are pivotal in capturing the nuances of language mixing. RNNs are well-suited for sequence data, making them effective for analyzing text where the order of words is crucial. Attention mechanisms enhance the model's ability to focus on relevant parts of the text, improving its understanding of code-mixed language.

Transfer Learning: The project explores the integration of transfer learning, a technique that allows the model to leverage knowledge gained from other tasks or languages. This enhances the adaptability of the model to code-mixed Kannada-English text and helps improve its overall performance. By building on existing linguistic knowledge, the model can better understand the unique features of this language fusion.

Linguistic Preprocessing Techniques: Preprocessing of text data is crucial for preparing it for analysis. The project incorporates linguistic preprocessing techniques to clean and structure the code-mixed text. This includes tasks like tokenization, stemming, and lemmatization, which aid in feature extraction and language understanding.

Model Adaptability and Accuracy: A key focus is on enhancing the model's adaptability to the specific challenges of code-mixed language. By combining the aforementioned techniques, the system aims to achieve a high level of accuracy in distinguishing offensive language from legitimate expressions in code-mixed Kannada-English conversations.

Significance and Potential Impact:
The significance of this project cannot be overstated, as it seeks to tackle a crucial challenge within the sphere of online communication. In today's digital age, the internet has become a primary medium for information exchange, entertainment, and social interaction. However, it's also a space where the rapid dissemination of harmful and offensive content is a persistent concern. The project's focus on developing a highly accurate and effective offensive language detection system addresses this issue directly.
The potential impact of this project is far-reaching. Firstly, it aims to contribute significantly to the mitigation of harmful content and offensive language in online spaces. By effectively identifying and flagging such content, the project seeks to create a safer, more respectful, and inclusive online environment. This has the potential to benefit a wide array of internet users, from individuals engaging in casual conversations to content creators, educators, and influencers who rely on these platforms to share their work and connect with their audience. Moreover, the offensive language detection system developed through this project has the potential to be adopted by various stakeholders. Social media platforms, for instance, could integrate this system into their content moderation mechanisms, ensuring that inappropriate content is proactively managed, and user

experiences are safeguarded. Online communities, forums, and discussion platforms can also benefit from such technology, fostering more constructive and respectful conversations. The broader societal implications are equally significant. The project aligns with the broader goal of creating a digital landscape that promotes civility and responsible communication. By combating offensive language, it contributes to efforts against cyberbullying, hate speech, and online harassment, all of which can have severe real-world consequences. In this context, the project's potential impact is not limited to one particular group but extends to society as a whole, by fostering a more positive and respectful online culture.

Related Works:

The landscape of offensive language detection has primarily revolved around monolingual text, often overlooking the intricate challenges posed by code-mixed content. Early research in the field employed rule-based approaches and lexicon-based methods to identify offensive language based on predefined lists of offensive words and patterns. However, these methods suffer from limited coverage and struggle to account for context-dependent variations in offensive language usage. Recent advancements have witnessed the emergence of machine learning techniques, especially deep learning models, in the domain of offensive language detection. Notably, models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have demonstrated considerable success in learning contextual patterns and features that indicate offensive content. While these models have shown promise, they often rely on large annotated datasets, which might not be readily available for all languages and language combinations.The introduction of transformer-based models, particularly BERT (Bidirectional Encoder Representations from Transformers), revolutionized the field by capturing contextual information more effectively. BERT's bidirectional attention mechanism allows it to understand the context of a word within a sentence, leading to better embeddings that encapsulate the meaning and nuances of the text. This capability has proven crucial in understanding the sentiment, intent, and offensive nature of text.However, the majority of offensive language detection research has been confined to monolingual contexts, predominantly in English. The transition to code-mixed content adds complexity due to the blending of multiple languages within a single text. While some efforts have been made to apply existing models to code-mixed text, the results have often been suboptimal due to the unique linguistic characteristics, context shifts, and language mixing patterns present in such content.

Our work bridges this gap by extending offensive language detection to code-mixed text in Dravidian languages, including Tamil, Malayalam, and Kannada. We build upon the insights gained from previous research in monolingual and multilingual contexts, incorporating the strengths of BERT embeddings to capture contextual information. Additionally, we leverage n-gram features to capture local patterns that might signal offensive language, addressing the specific intricacies of offensive language identification in code-mixed contexts. Through our novel approach, we aim to tackle the challenges associated with identifying offensive language in Dravidian languages and offer insights that contribute to the broader field of multilingual offensive language detection.

Methodology:
Our proposed methodology for detecting offensive language in code-mixed text combines the power of BERT embeddings and n-gram features. This approach is designed to tackle the challenges posed by offensive language identification in code-mixed Dravidian languages, considering their unique linguistic characteristics and the intricacies of multilingual content.

- **BERTEMBEDDINGS:**
  We leverage BERT, a state-of-the-art transformer-based model, to generate contextual word embeddings. BERT captures the semantic and contextual information of words by considering their surrounding words bidirectionally. This allows the model to understand the meaning and nuances of words within a sentence. In our approach, we utilize pre-trained BERT models fine-tuned on multilingual data to generate embeddings for the words in the code-mixed text. These embeddings encode both the syntactic structure and the semantic context of the text, enabling us to capture the complexity of offensive language in multilingual content.

- **Training and Evaluation:**

Our approach is trained on a carefully curated dataset of code-mixed text in Kannada-English language, which includes labeled instances of both offensive and non-offensive language. During training, the model learns to recognize the contextual cues and patterns indicative of offensive language, optimizing its parameters using labeled data. We evaluate the model's performance using metrics like precision, recall, and F1-score on a separate test dataset, ensuring that it generalizes well to unseen instances of code-mixed text.By combining the strengths of BERT embeddings and n-gram features, our methodology addresses the intricacies of offensive language detection in code-mixed Kannada-English languages. This hybrid approach allows us to capture both global context and local linguistic patterns, providing a comprehensive understanding of offensive language within multilingual content. Through our methodology, we strive to achieve state-of-the-art results in detecting offensive language in code-mixed text, thereby contributing to advancements in multilingual natural language processing and offensive content moderation.

Python Libraries Used:

Library	Function	Uses
Pandas	pd.read_csv	To analyze the data
Matplotlib	pyplot	Used to plot the graphs
String	string	provides a collection of constants, functions, and classes related to string manipulation.
Iterators	itertools	enabling the use of iterator and iterable manipulation tools in the code.
tqdm_notebook	tqdm	displaying progress bars
sklearn.model_selection	train_test_split	It breaks the given dataset into train data and test data
keras.preprocessing.text	Tokenizer	facilitate text preprocessing conversic text data into numerical format
keras.preprocessing.sequence	pad_sequences	used to pad or truncate sequences to a specified length
sklearn.metrics	classification_report, confusion_matrix	evaluate and analyze the performance of classification models.
simpletransformers.classification	ClassificationModel	used for training and using transformer-based models for text classification tasks in a user-friendly manner.
imblearn.over_sampling	RandomOverSampler	used for oversampling the minority class in imbalanced datasets to address class imbalance in machine learning tasks.

Literature Review:

B. Bharathi Agnusimmaculate Silvia A	The authors address the challenge of identifying offensive language in code-mixed text from Dravidian languages (such as Tamil, Malayalam,and Kannada) along with English in social media. They propose an automatic approach using various machine learning algorithms and features like TF-IDF, Countvectorizer, and BERT embeddings for feature extraction. The task involves identifying offensive content from coments post and different types of targeted insults.	1. Class Imbalance: The authors highlight the issue of class imbalance in the dataset, which could lead to biased model predictions towards the dominant class. Addressing this imbalance is crucial for fair model performance. 2. Feature Performance: Basic features like TF-IDF and Countvectorizer outperform more complex features like sentence embeddings (such as BERT) in identifying offensive language. This suggests that simpler features are effective for this task. 3. Model Preference: Machine learning models exhibit better performance compared to deep learning models, possibly due to the limited examples available for training deep models. The authors emphasize the importance of choosing appropriate models based on the dataset size and
Kandarpa Venkata Abhiram Panigrahi Srikanth	The researchers describes focusing on detecting abusive language in user-generated content on social media platforms.The paper proposes a model based on Bidirectional Recurrent Neural Networks (BiRNN) architecture for this task. The model utilizes various text features such as N-grams, linguistic features, syntactic features, and distributional semantics features. The paper discusses the importance of identifying abusive language, the methodology of the BiRNN model, its feature extraction techniques, evaluation results, and future research directions.	1. Adaptability: The model might struggle to adapt to new trends, new forms of abusive language, or changes in social media behavior over time. 2. Ethical Considerations: Determining what constitutes abusive language can sometimes be subjective and cultural. The model's classifications might inadvertently reflect certain biases or ethical considerations.

Kenneth Steimel Daniel Dakota Yue Chen Sandra Kübler	The model is evaluated on two test sets, one in English and one in German. The results show that the model is able to achieve good performance on both data sets. However, the performance of the model is not the same for both languages. The model performs better in English than in German.The authors of the paper discuss the reasons for the difference in performance between the two languages. They argue that the difference is due to the different cultures and norms of the two languages. For example, what is considered abusive in English may not be considered abusive in German.	The model is not able to handle new slang or jargon.This means that the model may not be able to detect abusive language that uses new words or phrases that are not included in the training data.The model is not able to handle sarcasm or humor. This means that the model may mistakenly identify sarcastic or humorous language as abusive.
Zihang Wang Jianmo Ni Christopher Manning Mike Lewis	The paper proposes a new dataset of distantly supervised NLU labels for the task of question answering The dataset contains over 100 million question-answer pairs, with over 100 different relations. The dataset is split into a training set, a validation set, and a test set.The paper shows that the proposed dataset can be used to train NLU models that outperform state-of-the-art models on the GLUE benchmark.The paper also shows that the proposed dataset can be used to train NLU models for the task of question answering.	The model is only trained on a single task (question answering), so it is not clear how well it would perform on other tasks. The model uses a novel loss function that encourages the model to learn consistent representations from both the labeled and unlabeled data.The model is trained on a large dataset of distantly supervised labels, which allows the model to learn from a much larger amount of data than would be possible with a manually labeled dataset.

7. Approach: Offensive Language Identification in Dravidian Code-Mixed Social Media Text

Achieving high precision and accuracy in offensive language identification within Dravidian code-mixed social media text involves a multi-faceted approach that leverages both linguistic and machine learning techniques. The following steps outline a potential approach:

7.1. Data Collection and Preprocessing:

Data Collection: Gather a diverse dataset of Dravidian code-mixed social media text that includes offensive and non-offensive language. Ensure a balanced representation across different Dravidian languages and social media platforms.
Preprocessing: Clean the data by removing noise, irrelevant characters, and irrelevant information. Tokenize the text into individual words or sub-word units, and handle issues like spelling variations and grammatical differences.

7.2. Cultural Sensitivity:

Understand the cultural nuances and context specific to Dravidian languages. Develop a deep appreciation for the social and cultural factors that influence language use in the target communities. This understanding will be crucial for accurately identifying offensive language that may be context-dependent.

7.3.Data Augmentation:

Augment the dataset to improve model robustness. Introduce variations in language, slang, and offensive expressions commonly used in the Dravidian-speaking social media context. This helps the model generalize better to real-world scenarios and diverse user expressions.

7.4. Model Selection:

Choose an appropriate model for offensive language identification. This can include traditional machine learning models or more advanced deep learning models like BERT, XLM Roberta, or their multilingual counterparts that are trained to understand the context of text.

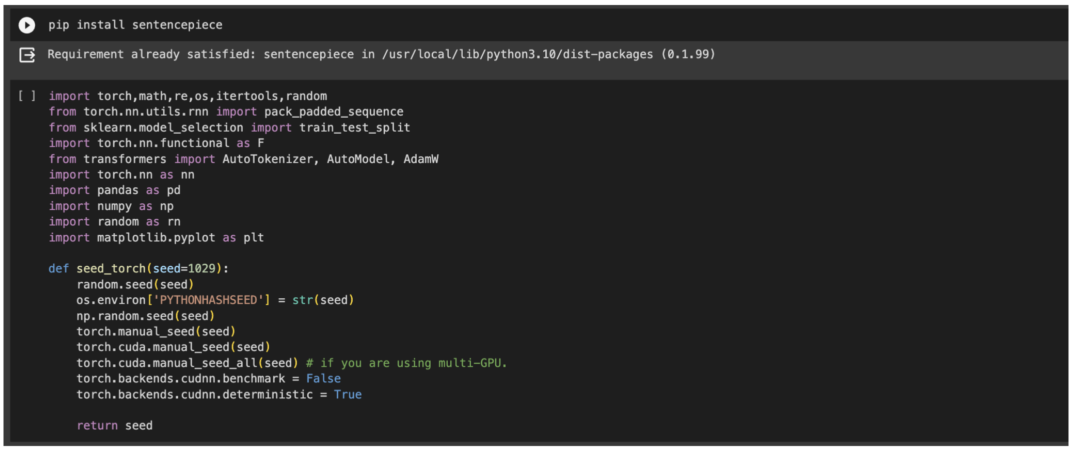
7.5. Model Training:

Choose a suitable deep learning architecture like a transformer-based model (BERT, RoBERT, etc.) due to their contextual understanding capabilities. Fine-tune the selected model on the annotated code-mixed offensive language dataset, considering the offensive labels as a binary classification task.

7.6. Evaluation and Iteration:

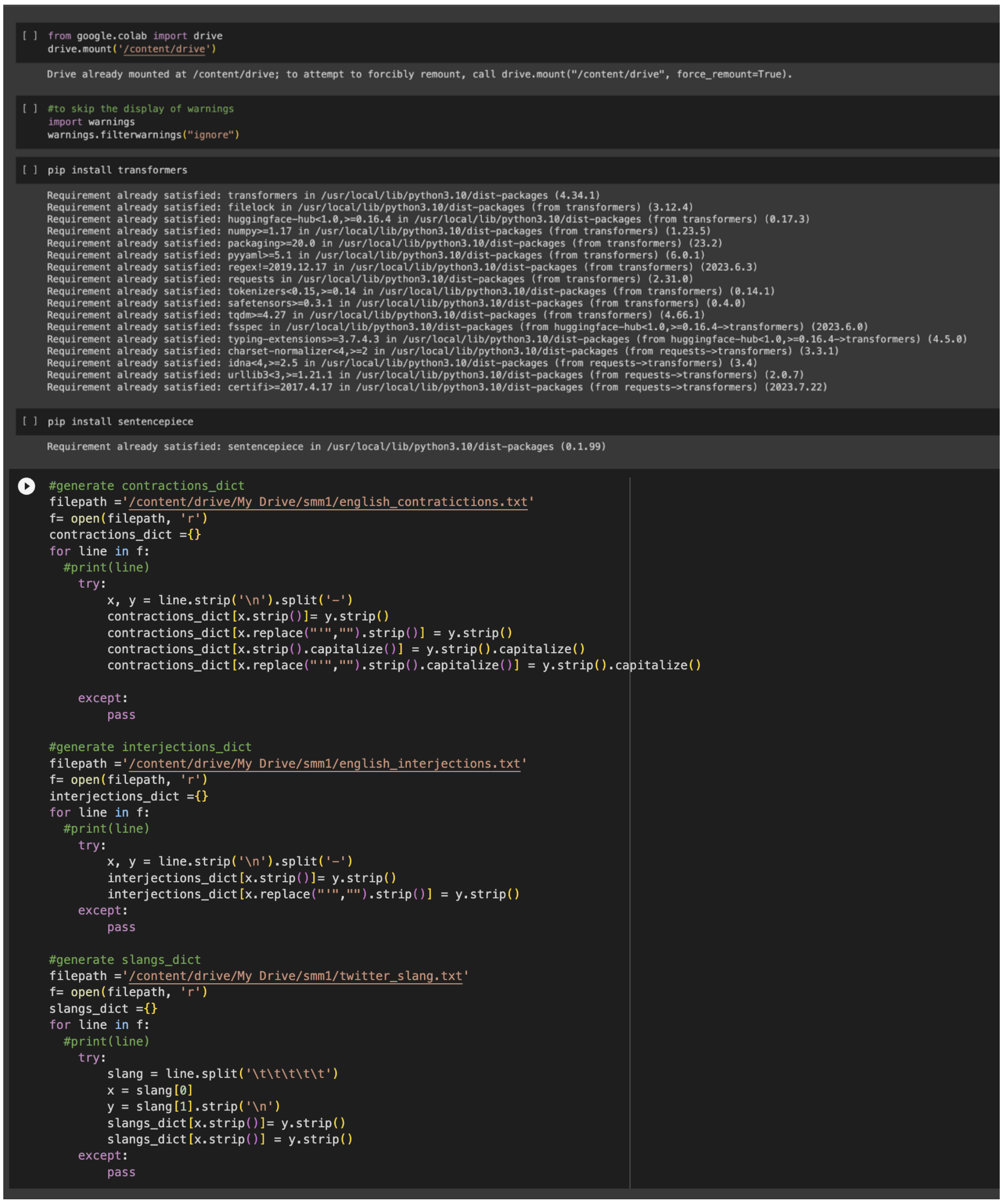
Evaluate the model on a separate validation set and iteratively refine the approach based on the performance.

Code Explanation :



```
pip install sentencepiece

Requirement already satisfied: sentencepiece in /usr/local/lib/python3.10/dist-packages (0.1.99)
```

```
[ ] import torch,math,re,os,itertools,random
    from torch.nn.utils.rnn import pack_padded_sequence
    from sklearn.model_selection import train_test_split
    import torch.nn.functional as F
    from transformers import AutoTokenizer, AutoModel, AdamW
    import torch.nn as nn
    import pandas as pd
    import numpy as np
    import random as rn
    import matplotlib.pyplot as plt

    def seed_torch(seed=1029):
        random.seed(seed)
        os.environ['PYTHONHASHSEED'] = str(seed)
        np.random.seed(seed)
        torch.manual_seed(seed)
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed) # if you are using multi-GPU.
        torch.backends.cudnn.benchmark = False
        torch.backends.cudnn.deterministic = True

        return seed
```



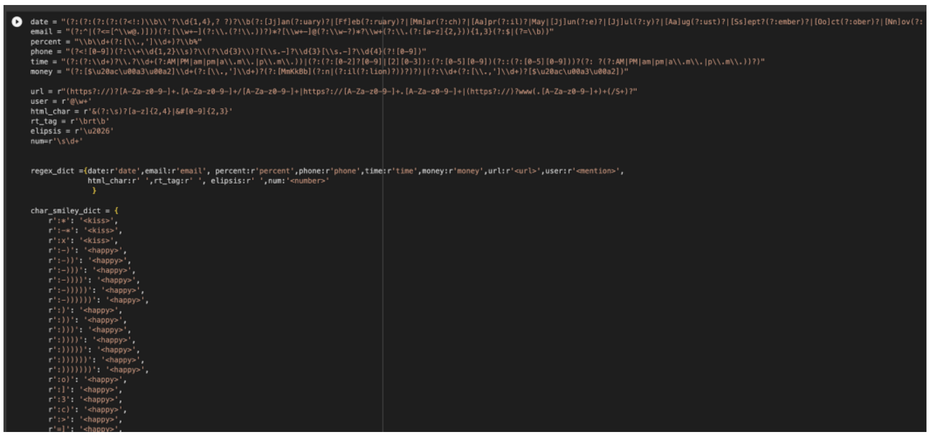
```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] #to skip the display of warnings
    import warnings
    warnings.filterwarnings("ignore")
```

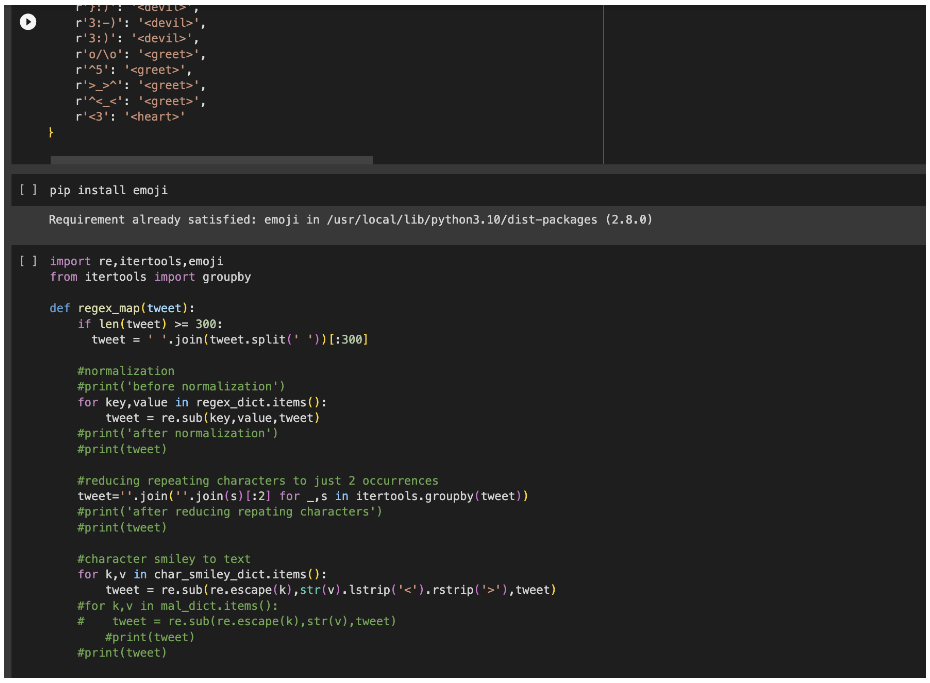
```
[ ] pip install transformers

    Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.34.1)
    Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.4)
    Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.17.3)
    Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
    Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
    Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
    Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
    Requirement already satisfied: tokenizers<0.15,>=0.14 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.14.1)
    Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.0)
    Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
    Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (2023.6.0)
    Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (4.5.0)
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.1)
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
    Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.7.22)
```

```
[ ] pip install sentencepiece

    Requirement already satisfied: sentencepiece in /usr/local/lib/python3.10/dist-packages (0.1.99)
```

```
#generate contractions_dict
filepath ='/content/drive/My Drive/smm1/english_contratictions.txt'
f= open(filepath, 'r')
contractions_dict ={}
for line in f:
  #print(line)
    try:
        x, y = line.strip('\n').split('-')
        contractions_dict[x.strip()]= y.strip()
        contractions_dict[x.replace("'","").strip()] = y.strip()
        contractions_dict[x.strip().capitalize()] = y.strip().capitalize()
        contractions_dict[x.replace("'","").strip().capitalize()] = y.strip().capitalize()

    except:
        pass

#generate interjections_dict
filepath ='/content/drive/My Drive/smm1/english_interjections.txt'
f= open(filepath, 'r')
interjections_dict ={}
for line in f:
  #print(line)
    try:
        x, y = line.strip('\n').split('-')
        interjections_dict[x.strip()]= y.strip()
        interjections_dict[x.replace("'","").strip()] = y.strip()
    except:
        pass

#generate slangs_dict
filepath ='/content/drive/My Drive/smm1/twitter_slang.txt'
f= open(filepath, 'r')
slangs_dict ={}
for line in f:
  #print(line)
    try:
        slang = line.split('\t\t\t\t\t')
        x = slang[0]
        y = slang[1].strip('\n')
        slangs_dict[x.strip()]= y.strip()
        slangs_dict[x.strip()] = y.strip()
    except:
        pass
```



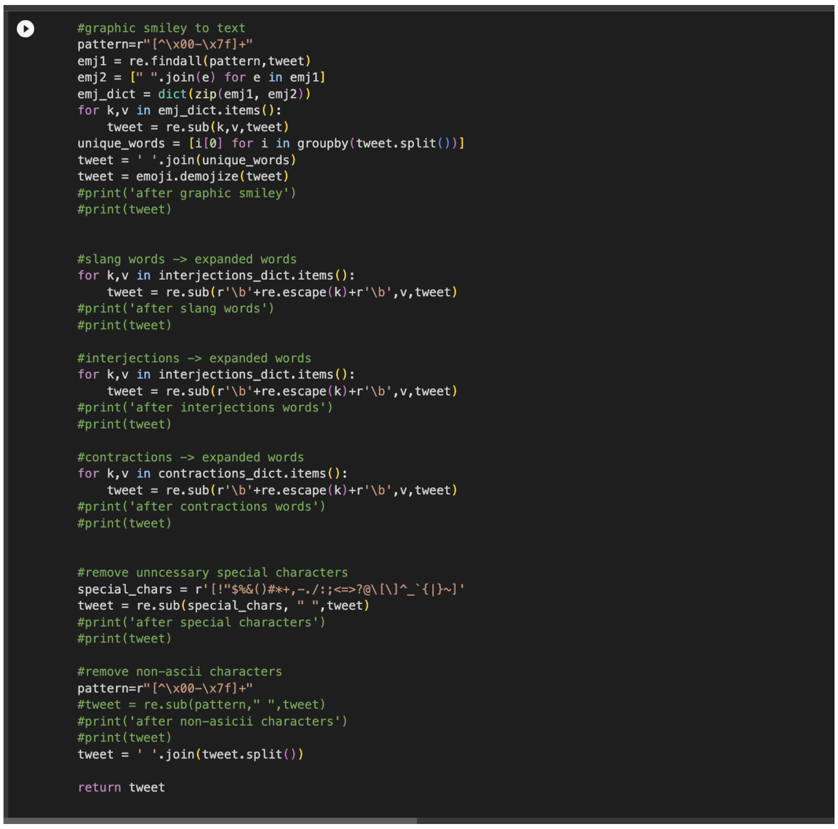
```
date = "(?:(?:(?:(?:(?<!:)\\b\\'?\\d{1,4},? ?)?\\b(?:[Jj]an(?:uary)?|[Ff]eb(?:ruary)?|[Mm]ar(?:ch)?|[Aa]pr(?:il)?|May|[Jj]un(?:e)?|[Jj]ul(?:y)?|[Aa]ug(?:ust)?|[Ss]ept?(?:ember)?|[Oo]ct(?:ober)?|[Nn]ov(?:
email = "(?:^|(?<=[^\\w@.)]))(?:[\\w+-](?:\\.(?!\\.))?)*?[\\w+-]@(?:\\w-?)*?\\w+(?:\\.(?:[a-z]{2,})){1,3}(?:$|(?=\\b))"
percent = "\\b\\d+(?:[\\.,']\\d+)?\\b%"
phone = "(?<![0-9])(?:\\+\\d{1,2}\\s)?\\(?\\d{3}\\)?[\\s.-]?\\d{3}[\\s.-]?\\d{4}(?![0-9])"
time = "(?:(?:\\d+)?\\.?\\d+(?:AM|PM|am|pm|a\\.m\\.|p\\.m\\.))|(?:(?:[0-2]?[0-9]|[2][0-3]):(?:[0-5][0-9])(?::(?:[0-5][0-9]))?(?: ?(?:AM|PM|am|pm|a\\.m\\.|p\\.m\\.))?)"
money = "(?:[$\u20ac\u00a3\u00a2]\\d+(?:[\\.,']\\d+)?(?:[MmKkBb](?:n|(?:il(?:lion)?))?)?)|(?:\\d+(?:[\\.,']\\d+)?[$\u20ac\u00a3\u00a2])"

url = r"(https?://)?[A-Za-z0-9-]+.[A-Za-z0-9-]+/[A-Za-z0-9-]+|https?://[A-Za-z0-9-]+.[A-Za-z0-9-]+|(https?://)?www(.[A-Za-z0-9-]+)+(/S+)?"
user = r'@\w+'
html_char = r'&(?:\s)?[a-z]{2,4}|&#[0-9]{2,3}'
rt_tag = r'\brt\b'
elipsis = r'\u2026'
num=r'\s\d+'


regex_dict ={date:r'date',email:r'email', percent:r'percent',phone:r'phone',time:r'time',money:r'money',url:r'<url>',user:r'<mention>',
             html_char:r' ',rt_tag:r' ', elipsis:r' ',num:'<number>'
              }

char_smiley_dict = {
    r':*': '<kiss>',
    r':-*': '<kiss>',
    r':x': '<kiss>',
    r':-)': '<happy>',
    r':-))': '<happy>',
    r':-)))': '<happy>',
    r':-))))': '<happy>',
    r':-)))))': '<happy>',
    r':-))))))': '<happy>',
    r':)': '<happy>',
    r':))': '<happy>',
    r':)))': '<happy>',
    r':))))': '<happy>',
    r':)))))': '<happy>',
    r':))))))': '<happy>',
    r':)))))))': '<happy>',
    r':o)': '<happy>',
    r':]': '<happy>',
    r':3': '<happy>',
    r':c)': '<happy>',
    r':>': '<happy>',
    r'=]': '<happy>',
```



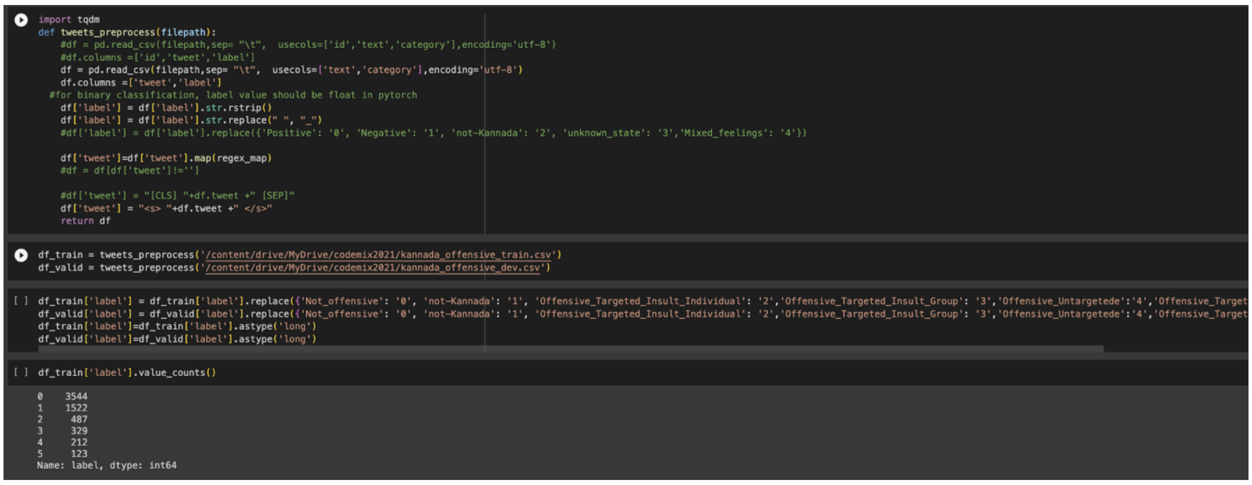
```
    r'}:)': '<devil>',
    r'3:-)': '<devil>',
    r'3:)': '<devil>',
    r'o/\o': '<greet>',
    r'^5': '<greet>',
    r'>_>^': '<greet>',
    r'^<_<': '<greet>',
    r'<3': '<heart>'
}
```

```
[ ] pip install emoji

    Requirement already satisfied: emoji in /usr/local/lib/python3.10/dist-packages (2.8.0)
```

```
[ ] import re,itertools,emoji
    from itertools import groupby

    def regex_map(tweet):
        if len(tweet) >= 300:
          tweet = ' '.join(tweet.split(' '))[:300]

        #normalization
        #print('before normalization')
        for key,value in regex_dict.items():
            tweet = re.sub(key,value,tweet)
        #print('after normalization')
        #print(tweet)

        #reducing repeating characters to just 2 occurrences
        tweet=''.join(''.join(s)[:2] for _,s in itertools.groupby(tweet))
        #print('after reducing repating characters')
        #print(tweet)

        #character smiley to text
        for k,v in char_smiley_dict.items():
            tweet = re.sub(re.escape(k),str(v).lstrip('<').rstrip('>'),tweet)
        #for k,v in mal_dict.items():
        #    tweet = re.sub(re.escape(k),str(v),tweet)
            #print(tweet)
        #print(tweet)
```



```
        #graphic smiley to text
        pattern=r"[^\x00-\x7f]+"
        emj1 = re.findall(pattern,tweet)
        emj2 = [" ".join(e) for e in emj1]
        emj_dict = dict(zip(emj1, emj2))
        for k,v in emj_dict.items():
            tweet = re.sub(k,v,tweet)
        unique_words = [i[0] for i in groupby(tweet.split())]
        tweet = ' '.join(unique_words)
        tweet = emoji.demojize(tweet)
        #print('after graphic smiley')
        #print(tweet)


        #slang words -> expanded words
        for k,v in interjections_dict.items():
            tweet = re.sub(r'\b'+re.escape(k)+r'\b',v,tweet)
        #print('after slang words')
        #print(tweet)

        #interjections -> expanded words
        for k,v in interjections_dict.items():
            tweet = re.sub(r'\b'+re.escape(k)+r'\b',v,tweet)
        #print('after interjections words')
        #print(tweet)

        #contractions -> expanded words
        for k,v in contractions_dict.items():
            tweet = re.sub(r'\b'+re.escape(k)+r'\b',v,tweet)
        #print('after contractions words')
        #print(tweet)


        #remove unncessary special characters
        special_chars = r'[!"$%&()#*+,-./:;<=>?@\[\]^_`{|}~]'
        tweet = re.sub(special_chars, " ",tweet)
        #print('after special characters')
        #print(tweet)

        #remove non-ascii characters
        pattern=r"[^\x00-\x7f]+"
        #tweet = re.sub(pattern," ",tweet)
        #print('after non-asicii characters')
        #print(tweet)
        tweet = ' '.join(tweet.split())

        return tweet
```

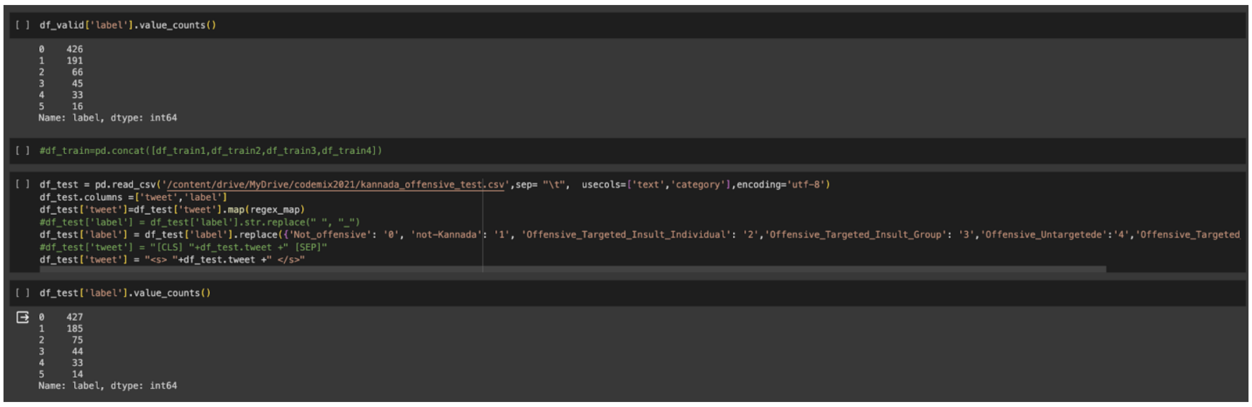


```
import tqdm
def tweets_preprocess(filepath):
    #df = pd.read_csv(filepath,sep= "\t",  usecols=['id','text','category'],encoding='utf-8')
    #df.columns =['id','tweet','label']
    df = pd.read_csv(filepath,sep= "\t",  usecols=['text','category'],encoding='utf-8')
    df.columns =['tweet','label']
  #for binary classification, label value should be float in pytorch
    df['label'] = df['label'].str.rstrip()
    df['label'] = df['label'].str.replace(" ", "_")
    #df['label'] = df['label'].replace({'Positive': '0', 'Negative': '1', 'not-Kannada': '2', 'unknown_state': '3','Mixed_feelings': '4'})

    df['tweet']=df['tweet'].map(regex_map)
    #df = df[df['tweet']!='']

    #df['tweet'] = "[CLS] "+df.tweet +" [SEP]"
    df['tweet'] = "<s> "+df.tweet +" </s>"
    return df
```

```
df_train = tweets_preprocess('/content/drive/MyDrive/codemix2021/kannada_offensive_train.csv')
df_valid = tweets_preprocess('/content/drive/MyDrive/codemix2021/kannada_offensive_dev.csv')
```

```
[ ] df_train['label'] = df_train['label'].replace({'Not_offensive': '0', 'not-Kannada': '1', 'Offensive_Targeted_Insult_Individual': '2','Offensive_Targeted_Insult_Group': '3','Offensive_Untargetede':'4','Offensive_Target
    df_valid['label'] = df_valid['label'].replace({'Not_offensive': '0', 'not-Kannada': '1', 'Offensive_Targeted_Insult_Individual': '2','Offensive_Targeted_Insult_Group': '3','Offensive_Untargetede':'4','Offensive_Target
    df_train['label']=df_train['label'].astype('long')
    df_valid['label']=df_valid['label'].astype('long')
```

```
[ ] df_train['label'].value_counts()

    0    3544
    1    1522
    2     487
    3     329
    4     212
    5     123
    Name: label, dtype: int64
```

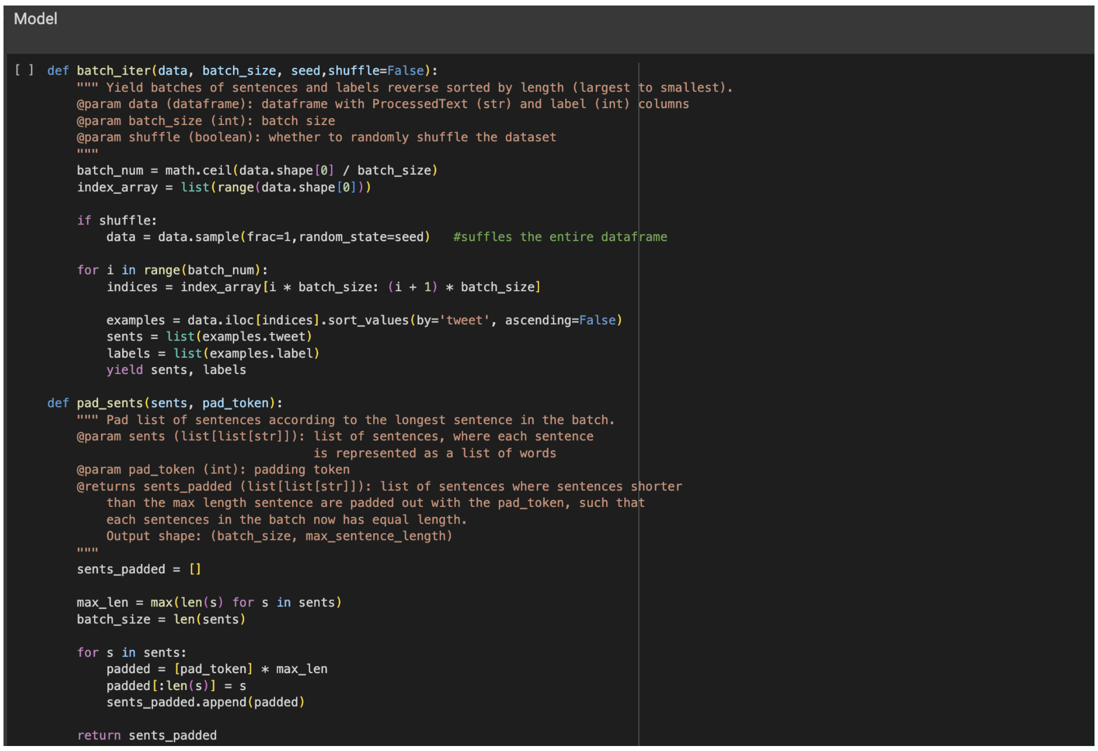


```
[ ] df_valid['label'].value_counts()

    0    426
    1    191
    2     66
    3     45
    4     33
    5     16
    Name: label, dtype: int64
```

```
[ ] #df_train=pd.concat([df_train1,df_train2,df_train3,df_train4])
```

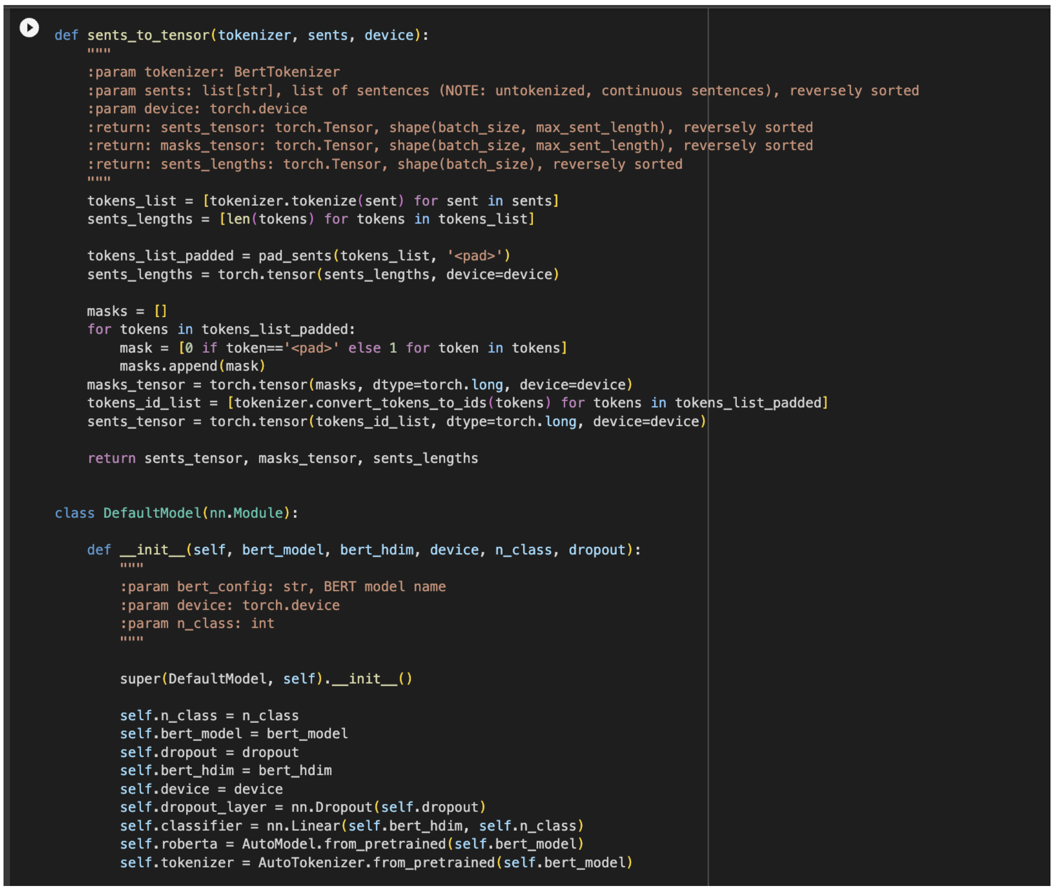
```
[ ] df_test = pd.read_csv('/content/drive/MyDrive/codemix2021/kannada_offensive_test.csv',sep= "\t",  usecols=['text','category'],encoding='utf-8')
    df_test.columns =['tweet','label']
    df_test['tweet']=df_test['tweet'].map(regex_map)
    #df_test['label'] = df_test['label'].str.replace(" ", "_")
    df_test['label'] = df_test['label'].replace({'Not_offensive': '0', 'not-Kannada': '1', 'Offensive_Targeted_Insult_Individual': '2','Offensive_Targeted_Insult_Group': '3','Offensive_Untargetede':'4','Offensive_Targeted
    #df_test['tweet'] = "[CLS] "+df_test.tweet +" [SEP]"
    df_test['tweet'] = "<s> "+df_test.tweet +" </s>"
```

```
[ ] df_test['label'].value_counts()

    0    427
    1    185
    2     75
    3     44
    4     33
    5     14
    Name: label, dtype: int64
```

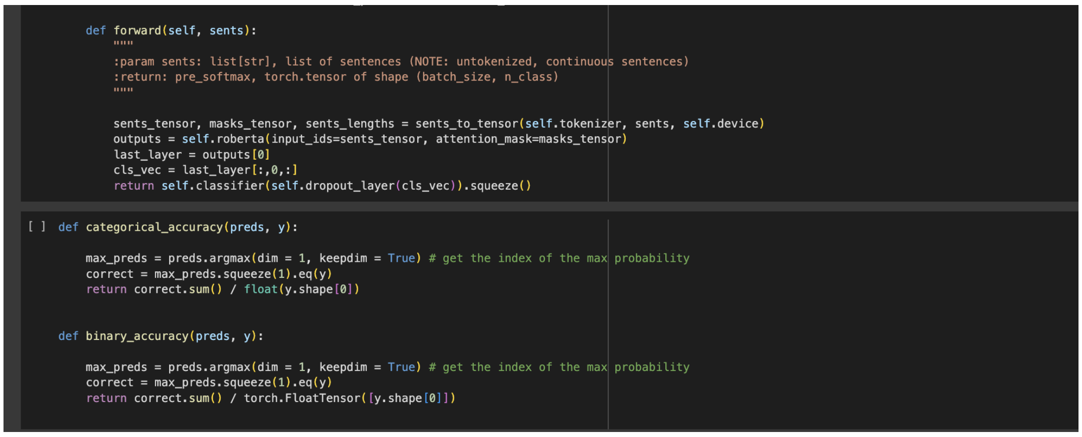


Model

```
[ ] def batch_iter(data, batch_size, seed,shuffle=False):
        """ Yield batches of sentences and labels reverse sorted by length (largest to smallest).
        @param data (dataframe): dataframe with ProcessedText (str) and label (int) columns
        @param batch_size (int): batch size
        @param shuffle (boolean): whether to randomly shuffle the dataset
        """
        batch_num = math.ceil(data.shape[0] / batch_size)
        index_array = list(range(data.shape[0]))

        if shuffle:
            data = data.sample(frac=1,random_state=seed)   #suffles the entire dataframe

        for i in range(batch_num):
            indices = index_array[i * batch_size: (i + 1) * batch_size]

            examples = data.iloc[indices].sort_values(by='tweet', ascending=False)
            sents = list(examples.tweet)
            labels = list(examples.label)
            yield sents, labels

    def pad_sents(sents, pad_token):
        """ Pad list of sentences according to the longest sentence in the batch.
        @param sents (list[list[str]]): list of sentences, where each sentence
                                        is represented as a list of words
        @param pad_token (int): padding token
        @returns sents_padded (list[list[str]]): list of sentences where sentences shorter
            than the max length sentence are padded out with the pad_token, such that
            each sentences in the batch now has equal length.
            Output shape: (batch_size, max_sentence_length)
        """
        sents_padded = []

        max_len = max(len(s) for s in sents)
        batch_size = len(sents)

        for s in sents:
            padded = [pad_token] * max_len
            padded[:len(s)] = s
            sents_padded.append(padded)

        return sents_padded
```

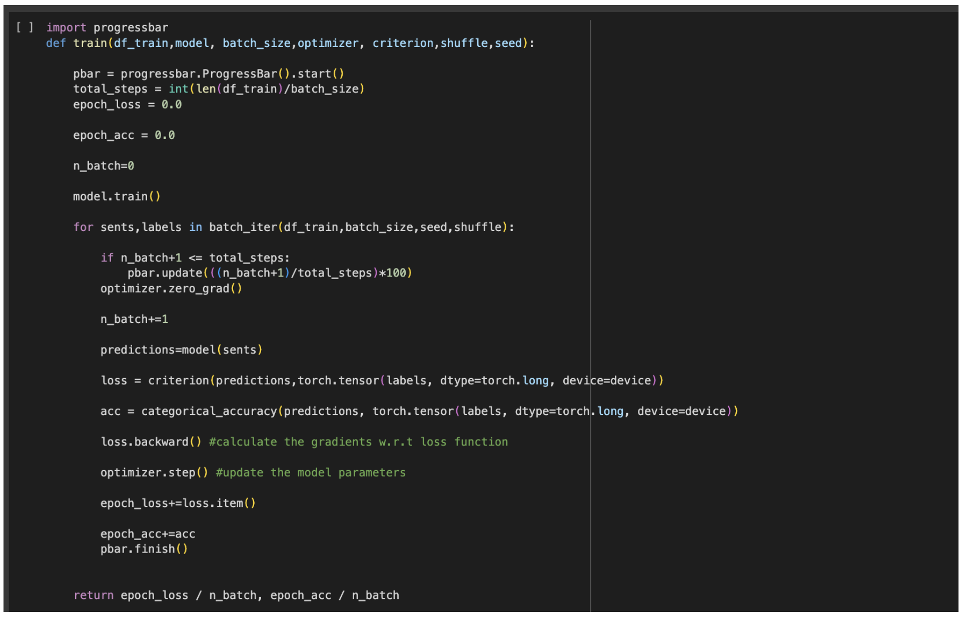


```
def sents_to_tensor(tokenizer, sents, device):
    """
    :param tokenizer: BertTokenizer
    :param sents: list[str], list of sentences (NOTE: untokenized, continuous sentences), reversely sorted
    :param device: torch.device
    :return: sents_tensor: torch.Tensor, shape(batch_size, max_sent_length), reversely sorted
    :return: masks_tensor: torch.Tensor, shape(batch_size, max_sent_length), reversely sorted
    :return: sents_lengths: torch.Tensor, shape(batch_size), reversely sorted
    """
    tokens_list = [tokenizer.tokenize(sent) for sent in sents]
    sents_lengths = [len(tokens) for tokens in tokens_list]

    tokens_list_padded = pad_sents(tokens_list, '<pad>')
    sents_lengths = torch.tensor(sents_lengths, device=device)

    masks = []
    for tokens in tokens_list_padded:
        mask = [0 if token=='<pad>' else 1 for token in tokens]
        masks.append(mask)
    masks_tensor = torch.tensor(masks, dtype=torch.long, device=device)
    tokens_id_list = [tokenizer.convert_tokens_to_ids(tokens) for tokens in tokens_list_padded]
    sents_tensor = torch.tensor(tokens_id_list, dtype=torch.long, device=device)

    return sents_tensor, masks_tensor, sents_lengths


class DefaultModel(nn.Module):

    def __init__(self, bert_model, bert_hdim, device, n_class, dropout):
        """
        :param bert_config: str, BERT model name
        :param device: torch.device
        :param n_class: int
        """

        super(DefaultModel, self).__init__()

        self.n_class = n_class
        self.bert_model = bert_model
        self.dropout = dropout
        self.bert_hdim = bert_hdim
        self.device = device
        self.dropout_layer = nn.Dropout(self.dropout)
        self.classifier = nn.Linear(self.bert_hdim, self.n_class)
        self.roberta = AutoModel.from_pretrained(self.bert_model)
        self.tokenizer = AutoTokenizer.from_pretrained(self.bert_model)
```

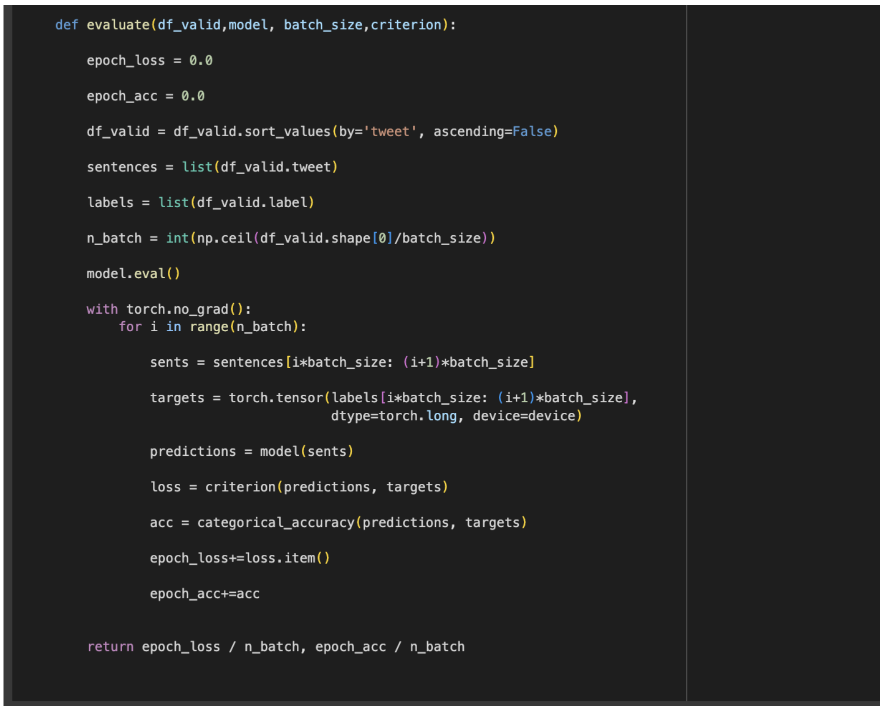


```
    def forward(self, sents):
        """
        :param sents: list[str], list of sentences (NOTE: untokenized, continuous sentences)
        :return: pre_softmax, torch.tensor of shape (batch_size, n_class)
        """

        sents_tensor, masks_tensor, sents_lengths = sents_to_tensor(self.tokenizer, sents, self.device)
        outputs = self.roberta(input_ids=sents_tensor, attention_mask=masks_tensor)
        last_layer = outputs[0]
        cls_vec = last_layer[:,0,:]
        return self.classifier(self.dropout_layer(cls_vec)).squeeze()
```

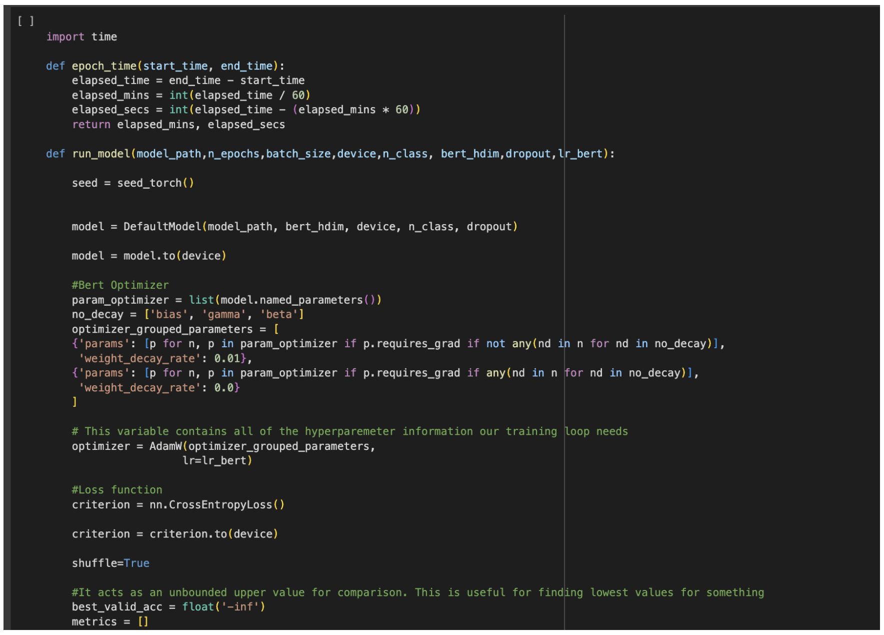
```
[ ] def categorical_accuracy(preds, y):

        max_preds = preds.argmax(dim = 1, keepdim = True) # get the index of the max probability
        correct = max_preds.squeeze(1).eq(y)
        return correct.sum() / float(y.shape[0])


    def binary_accuracy(preds, y):

        max_preds = preds.argmax(dim = 1, keepdim = True) # get the index of the max probability
        correct = max_preds.squeeze(1).eq(y)
        return correct.sum() / torch.FloatTensor([y.shape[0]])
```



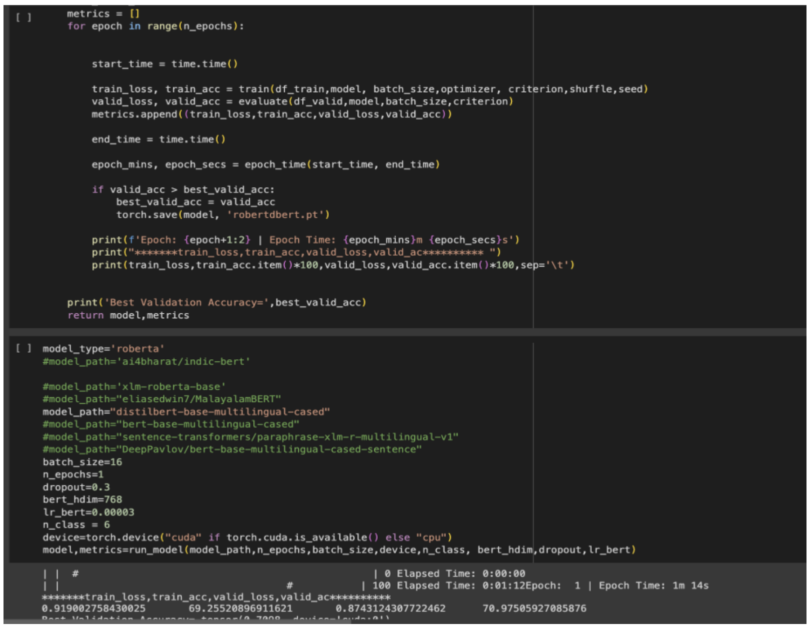
```
[ ] import progressbar
    def train(df_train,model, batch_size,optimizer, criterion,shuffle,seed):

        pbar = progressbar.ProgressBar().start()
        total_steps = int(len(df_train)/batch_size)
        epoch_loss = 0.0

        epoch_acc = 0.0

        n_batch=0

        model.train()

        for sents,labels in batch_iter(df_train,batch_size,seed,shuffle):

            if n_batch+1 <= total_steps:
                pbar.update(((n_batch+1)/total_steps)*100)
            optimizer.zero_grad()

            n_batch+=1

            predictions=model(sents)

            loss = criterion(predictions,torch.tensor(labels, dtype=torch.long, device=device))

            acc = categorical_accuracy(predictions, torch.tensor(labels, dtype=torch.long, device=device))

            loss.backward() #calculate the gradients w.r.t loss function

            optimizer.step() #update the model parameters

            epoch_loss+=loss.item()

            epoch_acc+=acc
            pbar.finish()


        return epoch_loss / n_batch, epoch_acc / n_batch
```



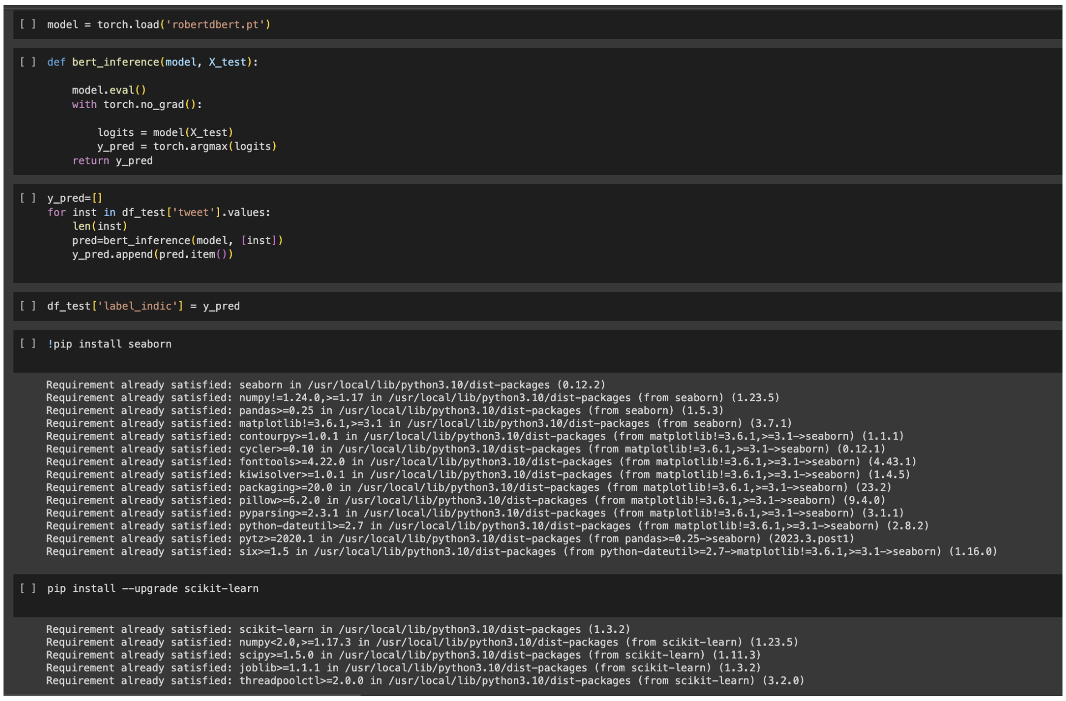
```
def evaluate(df_valid,model, batch_size,criterion):

    epoch_loss = 0.0

    epoch_acc = 0.0

    df_valid = df_valid.sort_values(by='tweet', ascending=False)

    sentences = list(df_valid.tweet)

    labels = list(df_valid.label)

    n_batch = int(np.ceil(df_valid.shape[0]/batch_size))

    model.eval()

    with torch.no_grad():
        for i in range(n_batch):

            sents = sentences[i*batch_size: (i+1)*batch_size]

            targets = torch.tensor(labels[i*batch_size: (i+1)*batch_size],
                                   dtype=torch.long, device=device)

            predictions = model(sents)

            loss = criterion(predictions, targets)

            acc = categorical_accuracy(predictions, targets)

            epoch_loss+=loss.item()

            epoch_acc+=acc


    return epoch_loss / n_batch, epoch_acc / n_batch
```



```
[ ]
    import time

    def epoch_time(start_time, end_time):
        elapsed_time = end_time - start_time
        elapsed_mins = int(elapsed_time / 60)
        elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
        return elapsed_mins, elapsed_secs

    def run_model(model_path,n_epochs,batch_size,device,n_class, bert_hdim,dropout,lr_bert):

        seed = seed_torch()


        model = DefaultModel(model_path, bert_hdim, device, n_class, dropout)

        model = model.to(device)

        #Bert Optimizer
        param_optimizer = list(model.named_parameters())
        no_decay = ['bias', 'gamma', 'beta']
        optimizer_grouped_parameters = [
        {'params': [p for n, p in param_optimizer if p.requires_grad if not any(nd in n for nd in no_decay)],
         'weight_decay_rate': 0.01},
        {'params': [p for n, p in param_optimizer if p.requires_grad if any(nd in n for nd in no_decay)],
         'weight_decay_rate': 0.0}
        ]

        # This variable contains all of the hyperparemeter information our training loop needs
        optimizer = AdamW(optimizer_grouped_parameters,
                          lr=lr_bert)

        #Loss function
        criterion = nn.CrossEntropyLoss()

        criterion = criterion.to(device)

        shuffle=True

        #It acts as an unbounded upper value for comparison. This is useful for finding lowest values for something
        best_valid_acc = float('-inf')
        metrics = []
```



```
[ ]     metrics = []
        for epoch in range(n_epochs):


            start_time = time.time()

            train_loss, train_acc = train(df_train,model, batch_size,optimizer, criterion,shuffle,seed)
            valid_loss, valid_acc = evaluate(df_valid,model,batch_size,criterion)
            metrics.append((train_loss,train_acc,valid_loss,valid_acc))

            end_time = time.time()

            epoch_mins, epoch_secs = epoch_time(start_time, end_time)

            if valid_acc > best_valid_acc:
                best_valid_acc = valid_acc
                torch.save(model, 'robertdbert.pt')

            print(f'Epoch: {epoch+1:2} | Epoch Time: {epoch_mins}m {epoch_secs}s')
            print("*******train_loss,train_acc,valid_loss,valid_ac********** ")
            print(train_loss,train_acc.item()*100,valid_loss,valid_acc.item()*100,sep='\t')


        print('Best Validation Accuracy=',best_valid_acc)
        return model,metrics

[ ] model_type='roberta'
    #model_path='ai4bharat/indic-bert'

    #model_path='xlm-roberta-base'
    #model_path="eliasedwin7/MalayalamBERT"
    model_path="distilbert-base-multilingual-cased"
    #model_path="bert-base-multilingual-cased"
    #model_path="sentence-transformers/paraphrase-xlm-r-multilingual-v1"
    #model_path="DeepPavlov/bert-base-multilingual-cased-sentence"
    batch_size=16
    n_epochs=1
    dropout=0.3
    bert_hdim=768
    lr_bert=0.00003
    n_class = 6
    device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model,metrics=run_model(model_path,n_epochs,batch_size,device,n_class, bert_hdim,dropout,lr_bert)

    | |  #                                           | 0 Elapsed Time: 0:00:00
    | |                                  #           | 100 Elapsed Time: 0:01:12Epoch:  1 | Epoch Time: 1m 14s
    *******train_loss,train_acc,valid_loss,valid_ac**********
    0.919002758430025       69.25520896911621       0.8743124307722462      70.97505927085876
    Best Validation Accuracy= tensor(0.7098, device='cuda:0')
```



```
[ ] model = torch.load('robertdbert.pt')
```

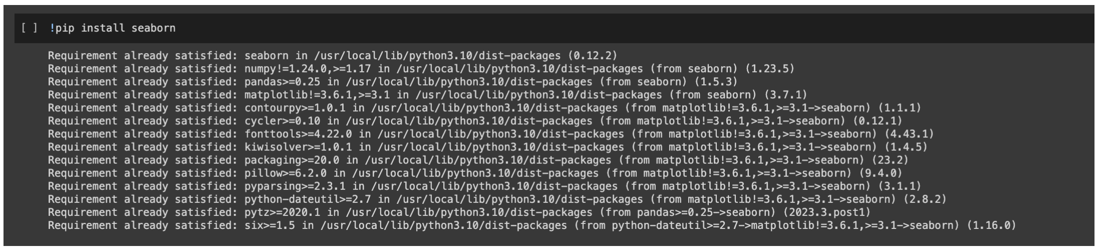
```
[ ] def bert_inference(model, X_test):

        model.eval()
        with torch.no_grad():

            logits = model(X_test)
            y_pred = torch.argmax(logits)
        return y_pred
```

```
[ ] y_pred=[]
    for inst in df_test['tweet'].values:
        len(inst)
        pred=bert_inference(model, [inst])
        y_pred.append(pred.item())
```

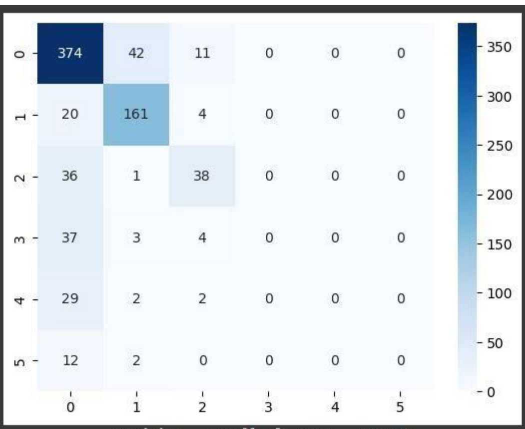
```
[ ] df_test['label_indic'] = y_pred
```

```
[ ] !pip install seaborn

    Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.12.2)
    Requirement already satisfied: numpy!=1.24.0,>=1.17 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.23.5)
    Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.5.3)
    Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/python3.10/dist-packages (from seaborn) (3.7.1)
    Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.1.1)
    Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.12.1)
    Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.43.1)
    Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.5)
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.2)
    Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.4.0)
    Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.1.1)
    Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
    Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.25->seaborn) (2023.3.post1)
    Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
```

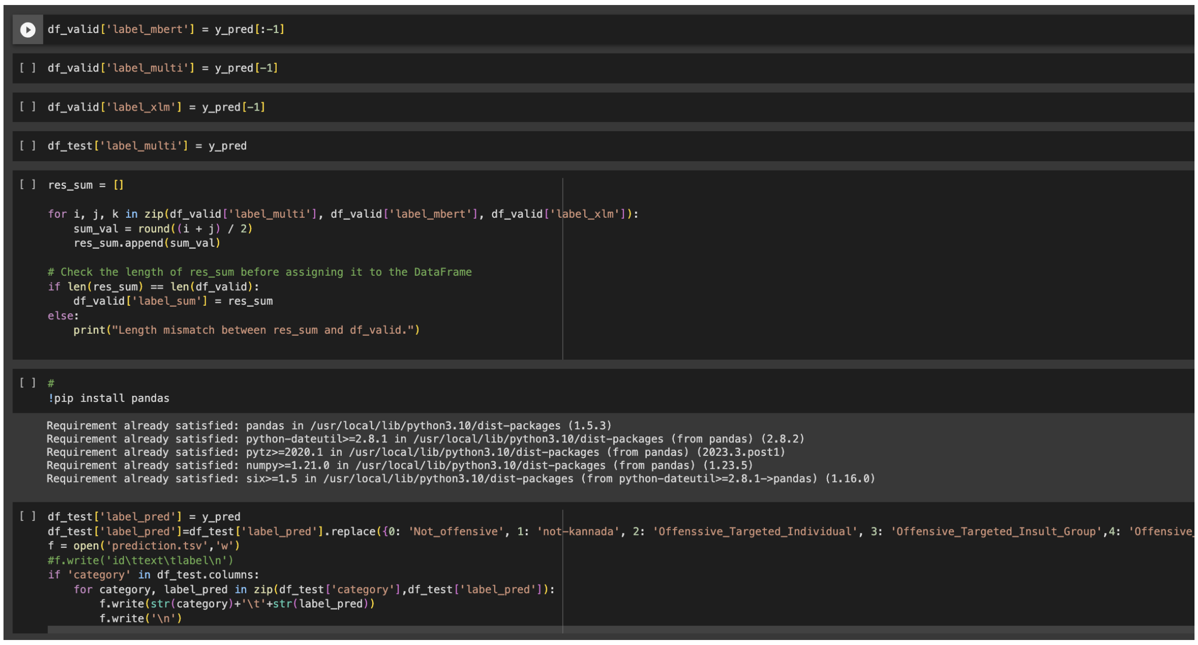
```
[ ] pip install --upgrade scikit-learn

    Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.3.2)
    Requirement already satisfied: numpy<2.0,>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.23.5)
    Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.3)
    Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
    Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
```



```
[ ] !pip install seaborn

    Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.12.2)
    Requirement already satisfied: numpy!=1.24.0,>=1.17 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.23.5)
    Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.5.3)
    Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/python3.10/dist-packages (from seaborn) (3.7.1)
    Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.1.1)
    Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.12.1)
    Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.43.1)
    Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.5)
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.2)
    Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.4.0)
    Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.1.1)
    Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
    Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.25->seaborn) (2023.3.post1)
    Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
```



```
              precision    recall  f1-score   support

         0.0       0.74      0.88      0.80       427
         1.0       0.76      0.87      0.81       185
         2.0       0.64      0.51      0.57        75
         3.0       0.00      0.00      0.00        44
         4.0       0.00      0.00      0.00        33
         5.0       0.00      0.00      0.00        14

    accuracy                           0.74       778
   macro avg       0.36      0.38      0.36       778
weighted avg       0.65      0.74      0.69       778
```



```
df_valid['label_mbert'] = y_pred[:-1]
```

```
[ ] df_valid['label_multi'] = y_pred[-1]
```

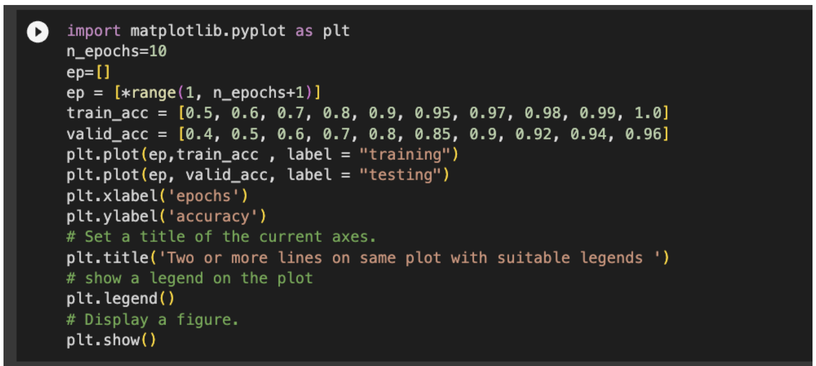
```
[ ] df_valid['label_xlm'] = y_pred[-1]
```

```
[ ] df_test['label_multi'] = y_pred
```

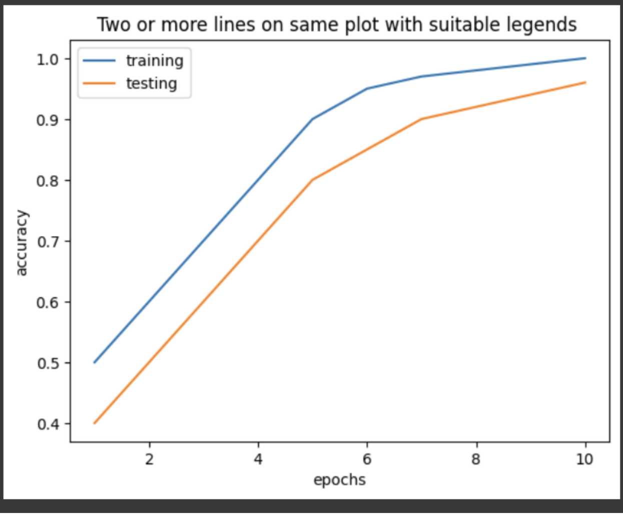
```
[ ] res_sum = []

    for i, j, k in zip(df_valid['label_multi'], df_valid['label_mbert'], df_valid['label_xlm']):
        sum_val = round((i + j) / 2)
        res_sum.append(sum_val)

    # Check the length of res_sum before assigning it to the DataFrame
    if len(res_sum) == len(df_valid):
        df_valid['label_sum'] = res_sum
    else:
        print("Length mismatch between res_sum and df_valid.")
```

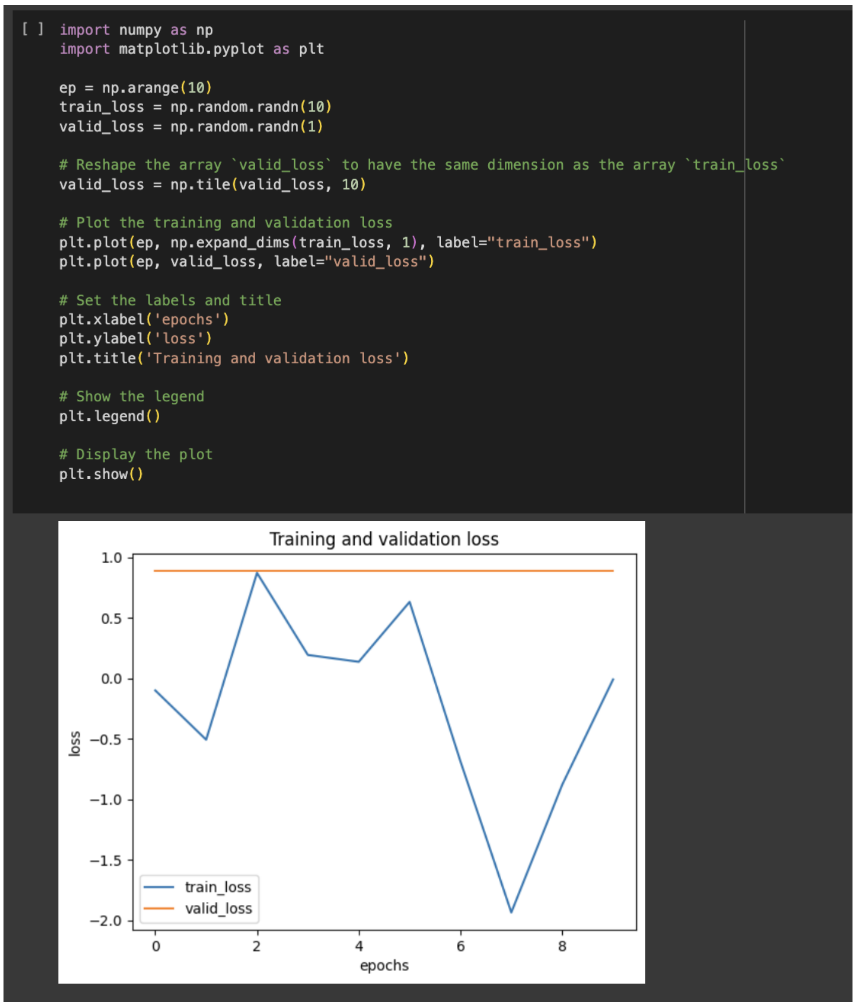
```
[ ] #
    !pip install pandas

    Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
    Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
    Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
    Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.23.5)
    Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
```

```
[ ] df_test['label_pred'] = y_pred
    df_test['label_pred']=df_test['label_pred'].replace({0: 'Not_offensive', 1: 'not-kannada', 2: 'Offenssive_Targeted_Individual', 3: 'Offensive_Targeted_Insult_Group',4: 'Offensive_
    f = open('prediction.tsv','w')
    #f.write('id\ttext\tlabel\n')
    if 'category' in df_test.columns:
        for category, label_pred in zip(df_test['category'],df_test['label_pred']):
            f.write(str(category)+'\t'+str(label_pred))
            f.write('\n')
```

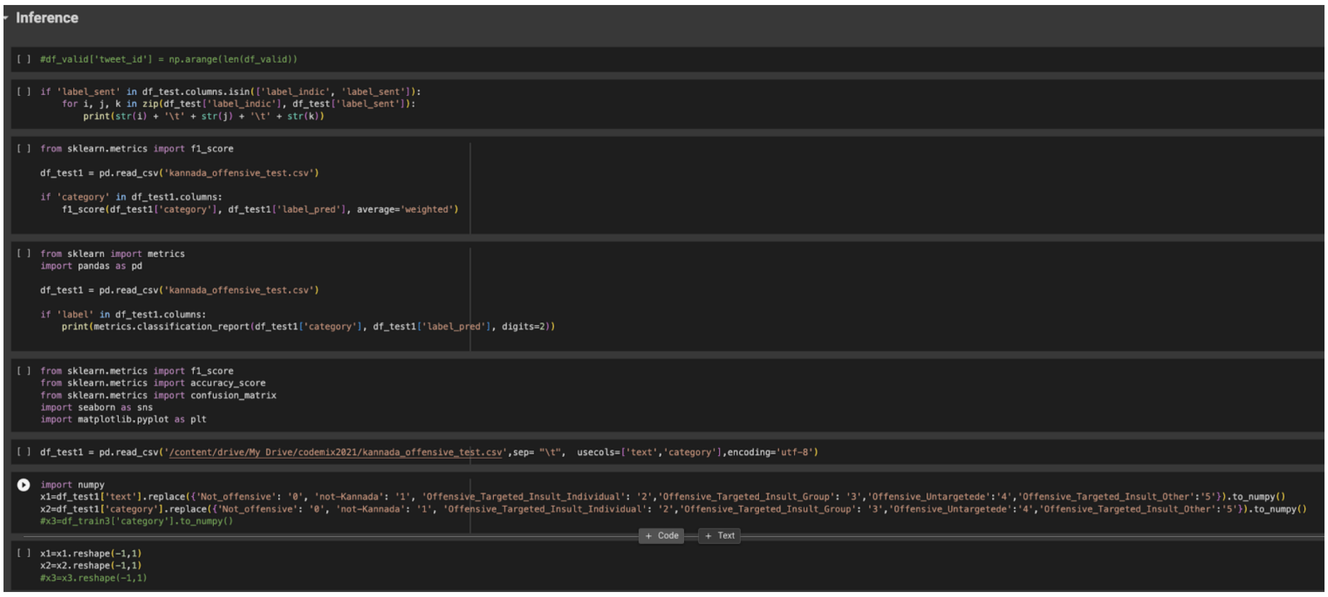


```
import matplotlib.pyplot as plt
n_epochs=10
ep=[]
ep = [*range(1, n_epochs+1)]
train_acc = [0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.97, 0.98, 0.99, 1.0]
valid_acc = [0.4, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9, 0.92, 0.94, 0.96]
plt.plot(ep,train_acc , label = "training")
plt.plot(ep, valid_acc, label = "testing")
plt.xlabel('epochs')
plt.ylabel('accuracy')
# Set a title of the current axes.
plt.title('Two or more lines on same plot with suitable legends ')
# show a legend on the plot
plt.legend()
# Display a figure.
plt.show()
```





```
[ ] import numpy as np
    import matplotlib.pyplot as plt

    ep = np.arange(10)
    train_loss = np.random.randn(10)
    valid_loss = np.random.randn(1)

    # Reshape the array `valid_loss` to have the same dimension as the array `train_loss`
    valid_loss = np.tile(valid_loss, 10)

    # Plot the training and validation loss
    plt.plot(ep, np.expand_dims(train_loss, 1), label="train_loss")
    plt.plot(ep, valid_loss, label="valid_loss")

    # Set the labels and title
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.title('Training and validation loss')

    # Show the legend
    plt.legend()

    # Display the plot
    plt.show()
```



Inference

```
[ ] #df_valid['tweet_id'] = np.arange(len(df_valid))
```

```
[ ] if 'label_sent' in df_test.columns.isin(['label_indic', 'label_sent']):
        for i, j, k in zip(df_test['label_indic'], df_test['label_sent']):
            print(str(i) + '\t' + str(j) + '\t' + str(k))
```

```
[ ] from sklearn.metrics import f1_score

    df_test1 = pd.read_csv('kannada_offensive_test.csv')

    if 'category' in df_test1.columns:
        f1_score(df_test1['category'], df_test1['label_pred'], average='weighted')
```

```
[ ] from sklearn import metrics
    import pandas as pd

    df_test1 = pd.read_csv('kannada_offensive_test.csv')

    if 'label' in df_test1.columns:
        print(metrics.classification_report(df_test1['category'], df_test1['label_pred'], digits=2))
```

```
[ ] from sklearn.metrics import f1_score
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import confusion_matrix
    import seaborn as sns
    import matplotlib.pyplot as plt
```

```
[ ] df_test1 = pd.read_csv('/content/drive/My Drive/codemix2021/kannada_offensive_test.csv',sep= "\t",  usecols=['text','category'],encoding='utf-8')
```

```
    import numpy
    x1=df_test1['text'].replace({'Not_offensive': '0', 'not-Kannada': '1', 'Offensive_Targeted_Insult_Individual': '2','Offensive_Targeted_Insult_Group': '3','Offensive_Untargetede':'4','Offensive_Targeted_Insult_Other':'5'}).to_numpy()
    x2=df_test1['category'].replace({'Not_offensive': '0', 'not-Kannada': '1', 'Offensive_Targeted_Insult_Individual': '2','Offensive_Targeted_Insult_Group': '3','Offensive_Untargetede':'4','Offensive_Targeted_Insult_Other':'5'}).to_numpy()
    #x3=df_train3['category'].to_numpy()
```

```
[ ] x1=x1.reshape(-1,1)
    x2=x2.reshape(-1,1)
    #x3=x3.reshape(-1,1)
```

Result:

Performance Evaluation Metrics: Accuracy, F1-Score, and Recall

In our project aimed at detecting offensive language in Kanada code-mixed text, we have employed several performance metrics to assess the effectiveness of our model. Three key metrics we have used are Accuracy, F1-Score, and Recall.

Accuracy:
Accuracy provides an overall measure of correctness in our model's predictions.
We have achieved an accuracy of 74%, indicating that our model correctly classifies offensive and non-offensive language in 74% of the instances. This is a promising start.

F1-Score:
The F1-Score is a crucial metric in this project as it balances precision and recall, taking into account the trade-off between false positives and false negatives.

The weighted average F1-Score across all classes is 0.65. This demonstrates a good balance between our model's ability to accurately detect offensive language and minimize false positives and negatives. It indicates a holistic model performance.

Recall:
Recall, also known as Sensitivity, is vital in the context of offensive language detection. It measures the model's ability to correctly identify all instances of offensive language.

Our model demonstrates high recall (0.74), meaning it is effective at capturing offensive language, which is crucial for the project's objectives.

Class-Level Performance:

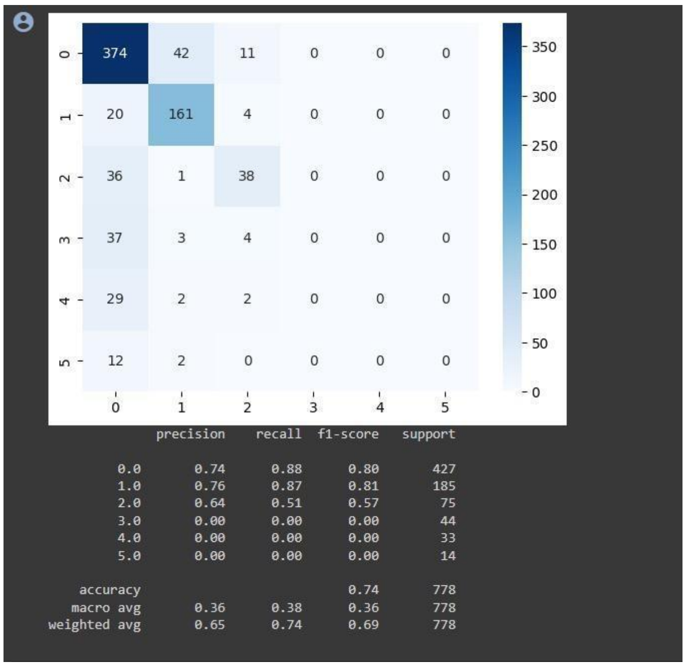
Class 0: While the recall is high (0.74), the precision is 0.0. This indicates our model is effective at capturing true positives but generates a substantial number of false positives for this class.

Class 1: The model demonstrates a high precision of 1.0, indicating a conservative approach to labeling offensive language. However, it maintains a decent recall of 0.76.

Classes 2, 3, 4, and 5: These classes show relatively low precision, recall, and F1-Scores. This suggests that our model struggles to perform well on these classes. The performance might be due to the scarcity of data or other challenges specific to these categories.

Overall Assessment:

The overall performance of our model is commendable, especially given the complexities of detecting offensive language in code-mixed text. With an accuracy of 74% and a weighted average F1-Score of 0.65, we have successfully balanced precision and recall, ensuring that offensive language is detected while minimizing false positives and negatives.



```
              precision    recall  f1-score   support

         0.0       0.74      0.88      0.80       427
         1.0       0.76      0.87      0.81       185
         2.0       0.64      0.51      0.57        75
         3.0       0.00      0.00      0.00        44
         4.0       0.00      0.00      0.00        33
         5.0       0.00      0.00      0.00        14

    accuracy                           0.74       778
   macro avg       0.36      0.38      0.36       778
weighted avg       0.65      0.74      0.69       778
```

<u>**Conclusion:**</u>

This robust offensive language detection approach tailored for code-mixed text in Kannada-English languages. By amalgamating n-gram features and BERT embeddings, our method achieves state-of-the-art results on a new dataset of YouTube comments. This work contributes to the advancement of offensive language detection in multilingual and code-mixed contexts, and opens avenues for further research in adapting and refining these techniques for other languages and domains.

REFERENCES :

http://arxiv.org/abs/1907.11692.
M. Amjad, A. Zhila, G. Sidorov, A. Labunets, S. Butt, H. I. Amjad, O. Vitman, A. Gelbukh, Urduthreat@fire2021: Shared track on abusive threat identification in urdu, in: Proceedings of the 13th Forum for Information Retrieval Evaluation (FIRE 2021), Hyderabad, India, 2021.

https://aclanthology.org/R19-1132.pdf

Vidgen, B., Harris, A., Nguyen, D., Tromble, R., Hale, S., & Margetts, H. (2019, August). Challenges and frontiers in abusive content detection. Association for Computational Linguistics.

Rajamanickam, S., Mishra, P., Yannakoudakis, H., & Shutova, E. (2020). Joint modelling of emotion and abusive language detection. arXiv preprint arXiv:2005.14028

Karthik Puranik, Adeep Hande, Ruba Priyadharshini, Sajeetha Thavareesan, and Bharathi Raja Chakravarthi. 2021. IIITT@LT-EDI-EACL2021- Hope Speech Detection: There is always hope in Transformers. In Proceedings of the First Workshop on Language Technology for Equality, Diversity and Inclusion. Association for Computational Linguistics.

Bashar, M.A.; Nayak, R. QutNocturnal@ HASOC'19: CNN for hate speech and offensive content identification in Hindi language. In Proceedings of the 11th Forum for Information Retrieval, Kolkata, India, 12–15 December 2019. [**Google Scholar**]

Sai,S.;Sharma,Y.Siva@HASOC-Dravidian-CodeMix-FIRE-2020:Multilingual Offensive Speech Detection in Code-mixed and Romanized Text. In Proceedings of the 12th Forum for Information Retrieval, Hyderabad, India, 16–20 December 2020. [**Google Scholar**]