National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Diwakar Muthuraj
Student ID: X23106824

School of Computing
National College of Ireland

Supervisor: Aaloka Anant

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Diwakar Muthuraj
Student ID:	X23106824
Programme:	MSc - Data Analytics **Year:** 2024
Module:	Research Project
Lecturer:	Aaloka Anant
Submission Due Date:	12/12/2024
Project Title:	Sentiment Analysis in Tamil-English Code-Mixed Data Using Hybrid Deep Learning Techniques
Word Count:	1658 **Page Count:** 16

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Diwakar Muthuraj

Date: 12/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Diwakar Muthuraj
Student ID: X23106824

1 Introduction

This research project looks to classify sentiments found in Tamil-English code-mixed data from YouTube comments using a pre-trained model with Text Graph Convolutional Network such as IndicBART+TextGCN (Dowlagar, 2021). This configuration manual discusses the step-by-step implementation procedure, by providing an overview of setting up the working environment in order to implement efficiently. It details the hardware, software, programming language, and libraries used in this working environment. And also shows the various experiments done during this research project and its evaluation results.

2 System Configuration

The hardware and software resources that were used are provided in this section.

2.1 Hardware Configuration

The Hardware configuration of the computing device:

- **Computing Device Name:** HP Victus
- **Operating System:** Windows 11
- **Processor:** AMD Ryzen 5 5600H – 3301 MHz – 6 Core(s)
- **RAM:** 16 GB
- **Number of Core(s):** 6
- **Dedicated Graphic Memory:** NVIDIA GeForce RTX 3050
- **Storage:** 500GB NVMe SSD

2.2 Software Configuration

The Software configuration of the computing device:

- **Web Browser:** Google Chrome
- **Platform As a Service:** Google Colabatory (Colab Pro)
- **Runtime Type Name:** T4 High RAM
- **Runtime Type Configuration:** 51GB – RAM / 15GB – GPU / 240GB Storage
- **Programming Language:** Python 3
- **Documentation:** MS Word

3 Project Implementation

The overall implementation of this research project is provided in this section including steps such as data collection, data preprocessing, feature extraction, experimental model and proposed model's training together with its evaluation.

The first step is setting up the working environment. Hence, Google Colabatory Pro with T4 High-RAM was chosen for developing the code as shown in figure 1. In order to access the environment, a Google account is needed to be signed in.

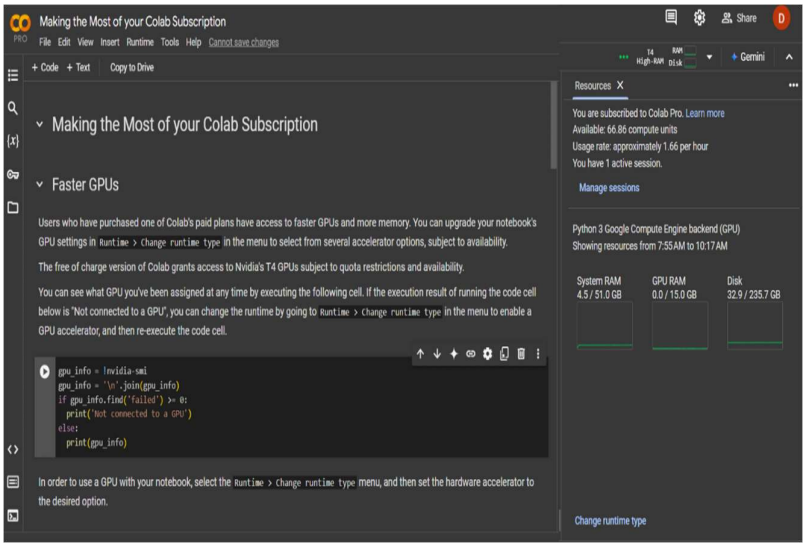


Figure 1: Setting up Google Colab Pro

After that as shown in Figure 2 all the necessary libraries that were installed and imported into the colab working environment.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datasets import load_dataset
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from wordcloud import WordCloud
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import emoji
from spellchecker import SpellChecker
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    precision_score,
    recall_score,
    f1_score
)
from collections import Counter
from imblearn.over_sampling import ADASYN
from imblearn.under_sampling import RandomUnderSampler
from sklearn.neighbors import kneighbors_graph, NearestNeighbors
import networkx as nx
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset, WeightedRandomSampler
from torch_geometric.data import Data, Batch
from torch_geometric.loader import DataLoader as GeometricDataLoader
from transformers import BertTokenizer, BertModel, AutoTokenizer, AutoModel, AdamW
from tqdm import tqdm


nltk.download('wordnet')
nltk.download('stopwords')
```

Figure 2: Installing and importing all the necessary libraries.

3.1 Data Collection

Then in the next step, dataset used for this research project 'Tamilmixsentiment' is fetched from 'Hugging Face'. This dataset contains code-mixed Tamil-English (Tanglish) as text and sentiment labels (Chakravarthi, 2020). The train and test data that were directly fetched from the Hugging Face is shown in the figure 3.

```
[4] ######################################## LOADING DATASET FROM HUGGINGFACE ########################################

[9] tam_eng_code_full = load_dataset('community-datasets/tamilmixsentiment') # Data from hugging is loaded as a dictionary
    print(tam_eng_code_full)

    TEC_train = tam_eng_code_full['train'] # spliting dictionary based on Train
    TEC_test = tam_eng_code_full['test'] # spliting dictionary based on Test

    print(type(TEC_train))  # To identify the datatype
    print(TEC_train)

    DatasetDict({
        train: Dataset({
            features: ['text', 'label'],
            num_rows: 11335
        })
        validation: Dataset({
            features: ['text', 'label'],
            num_rows: 1260
        })
        test: Dataset({
            features: ['text', 'label'],
            num_rows: 3149
        })
    })
    <class 'datasets.arrow_dataset.Dataset'>
    Dataset({
        features: ['text', 'label'],
        num_rows: 11335
    })
```

Figure 3: Fetching data as DatasetDict from Hugging Face.

In this step as Figure 4 shows, the train and test data being converted into a pandas data frame.

```
[ ] ######################################## CONVERTING TO PANDA DATAFRAME ########################################

[10] TEC_train_df = TEC_train.to_pandas()
     TEC_test_df = TEC_test.to_pandas()

     print(type(TEC_train_df))
     print(TEC_train_df)

     <class 'pandas.core.frame.DataFrame'>
                                                    text  label
     0                    Trailer late ah parthavanga like podunga      0
     1                  Move pathutu vanthu trailer pakurvnga yaru      0
     2                    Puthupetai dhanush  ah yarellam pathinga      0
     3            Dhanush oda character ,puthu sa erukay , mass ta      0
     4            vera level ippa pesungada mokka nu thalaivaaaaaa      0
     ...                                                       ...    ...
     11330              Yuvan shankar Raja anna fan's like here...      0
     11331       A masterpiece best revenge film I've ever scene      0
     11332                   Enna pa thala ya kamiya than katringa      0
     11333                                 R A A S H I K H A N N A      3
     11334  Trailer la nalla thaan iruku ana sound thaan k...      3

     [11335 rows x 2 columns]
```

Figure 4: Loading data as pandas data frame.

3.2 Exploring Data

Figure 5 to figure 7, shows the data exploration and visualization steps where the data is further explored and understood. This data has symbols, numbers and punctuations which have to be cleaned. Hence, feature engineering and handling class imbalance were done to prepare the data for model training.

```
[ ] ######################################## EXPLORATORY DATA ANALYSIS ########################################

# Head for all DF
print("TRAIN DataFrame:")
print(TEC_train_df.head())

print("\nTEST DataFrame:")
print(TEC_test_df.head())
```

```
TRAIN DataFrame:
                                                text  label
0          Trailer late ah parthavanga like podunga      0
1        Move pathutu vanthu trailer pakurvnga yaru      0
2          Puthupetai dhanush  ah yarellam pathinga      0
3  Dhanush oda character ,puthu sa erukay , mass ta      0
4  vera level ippa pesungada mokka nu thalaivaaaaaa      0

TEST DataFrame:
                                                text  label
0         Yarayellam FDFS ppga ippove ready agitinga      0
1  Ennada viswasam mersal sarkar madhri time la l...      0
2  yuvan vera level ya .... valuable script. SK i...      0
3  70 vayasulayum thanoda rasigargala sandhosapad...      2
4      all the best anna...Telugu makkal selvan fans      0
```

Figure 5: Loading data as pandas data frame

The class distribution of the train and test data are visualized in figure 6 and the frequently occurring words are visualized as Word Cloud in figure 7.

```
handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=palette[i], markersize=10)
           for i in range(len(label_mapping))]
plt.legend(handles, label_mapping.values(), title='Labels', loc='upper right')
plt.show()
```

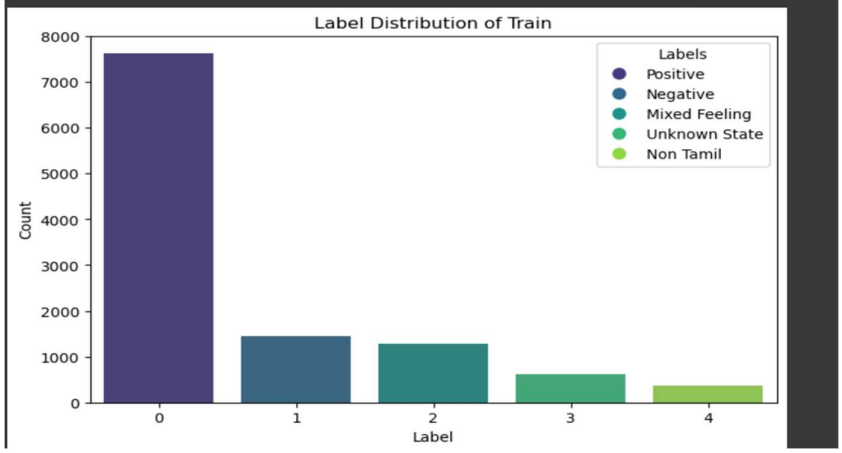


Figure 6: Class Distribution of the train and test data.

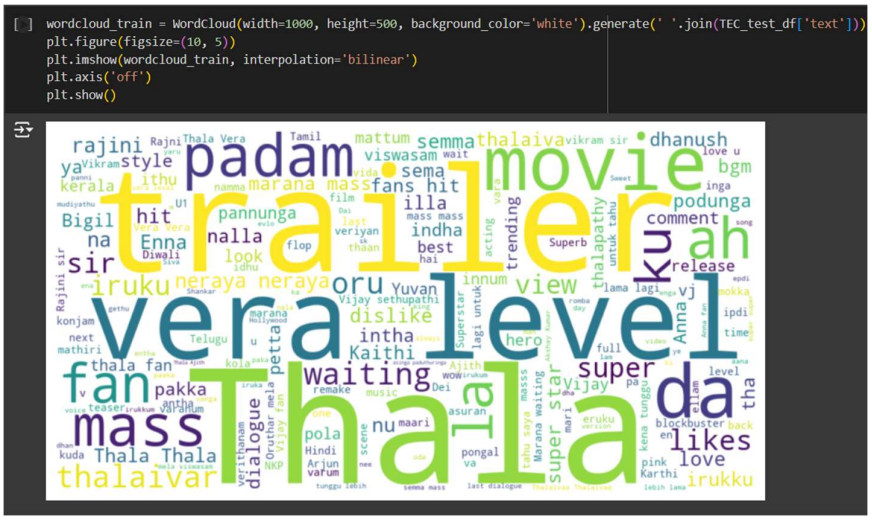


Figure 7: Word Cloud of frequent words.

3.3 Data Preprocessing

Next the data is cleaned and pre-processed as shown from figure 8 to figure 12. In order to maintain uniformity, both train and test data were converted into lowercase and the special characters, punctuation, numbers and stop words were removed (Dowlagar, 2021). The special characters, punctuation and numbers were removed from the data using regex as shown in figure 8 and figure 9.

```
# Removing special characters and punctuation
def remove_special_characters(text):
    return re.sub(r'[^a-zA-Z0-9\s]', '', text)

TEC_train_df['text'] = TEC_train_df['text'].apply(remove_special_characters)
TEC_test_df['text'] = TEC_test_df['text'].apply(remove_special_characters)
## to verify wether the special char are removed
print("TRAIN DataFrame:")
print(TEC_train_df.head())
```

```
TRAIN DataFrame:
                                          text  label  contains_tamil  \
0         Trailer late ah parthavanga like podunga      0           False
1       Move pathutu vanthu trailer pakurvnga yaru      0           False
2         Puthupetai dhanush  ah yarellam pathinga      0           False
3   Dhanush oda character puthu sa erukay  mass ta      0           False
4  vera level ippa pesungada mokka nu thalaivaaaaaa      0           False
```

Figure 8: Removing special characters and punctuation

```
[ ] # Removing numbers from text
    def remove_numbers(text):
        return re.sub(r'\d+', '', text)

    TEC_train_df['text'] = TEC_train_df['text'].apply(remove_numbers)
    TEC_test_df['text'] = TEC_test_df['text'].apply(remove_numbers)
    ## to verify wether the numbers are removed
    print("Test DataFrame:")
    print(TEC_test_df.head())
```

```
Test DataFrame:
                                                text  label  conta
0         yarayellam fdfs ppga ippove ready agitinga      0
1  ennada viswasam mersal sarkar madhri time la l...      0
2  yuvan vera level ya  valuable script sk in action      0
3   vayasulayum thanoda rasigargala sandhosapadut...      2
4         all the best annatelugu makkal selvan fans      0
```

Figure 9: Removing Numbers

After normalizing the text, it was then converted into lowercase using the str function as shown in the figure 10. This step standardized the text and makes the data consistent.

```
[ ] # Converting all text to lowercase
    TEC_train_df['text'] = TEC_train_df['text'].str.lower()
    TEC_test_df['text'] = TEC_test_df['text'].str.lower()
    # to verify wether the char are changed to lower
    print("TRAIN DataFrame:")
    print(TEC_train_df.head())
```

```
TRAIN DataFrame:
                                                text  label
0          trailer late ah parthavanga like podunga      0
1        move pathutu vanthu trailer pakurvnga yaru      0
2          puthupetai dhanush  ah yarellam pathinga      0
3    dhanush oda character puthu sa erukay  mass ta      0
4  vera level ippa pesungada mokka nu thalaivaaaaaa      0
```

Figure 10: Converting to lowercase

After conversion to lowercase, figure 11 shows the removal of stop words.

```
[ ] # Removing stopwords from text
    stop_words = set(stopwords.words('english'))

    def remove_stopwords(text):
        return ' '.join([word for word in text.split() if word not in stop_words])

    TEC_train_df['text'] = TEC_train_df['text'].apply(remove_stopwords)
    TEC_test_df['text'] = TEC_test_df['text'].apply(remove_stopwords)
    ## to verify wether the stopwords are removed
    print("Test DataFrame:")
    print(TEC_test_df.head())
```

```
Test DataFrame:
                                            text  label  contains_tamil
0         yarayellam fdfs ppga ippove ready agitinga      0           False
1  ennada viswasam mersal sarkar madhri time la l...      0           False
2      yuvan vera level ya valuable script sk action      0           False
3  vayasulayum thanoda rasigargala sandhosapaduth...      2           False
4               best annatelugu makkal selvan fans      0           False
```

Figure 11: Removing stop words

Finally, the text is lemmatized as shown in the figure 12.

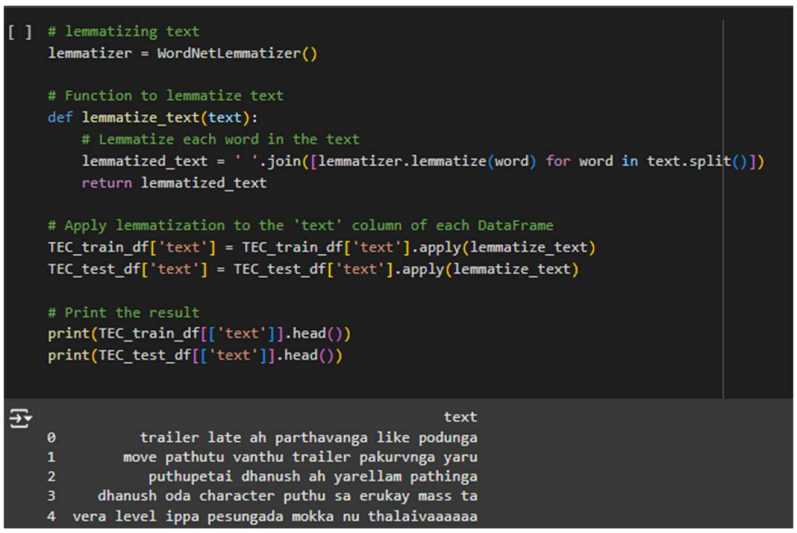


Figure 12: Lemmatizing text

3.4 Feature Engineering

After the preprocessing step, the 'text' column containing code-mixed Tamil-English data is feature engineered. 'Text' column is a raw data where it consists of a mix of words with English and Tanglish (Romanized Tamil words). Due to this, the model will struggle to convert these words into proper embedding (Banerjee, 2020). Hence, in order for the model to properly embed the words and classify the sentiment, the 'text' was back transliterated into its original script, such as English text into English script and Tamil 'text' into Tamil script.

This is done using the following steps such as tokenizing words, Back Transliteration, Combining the English words and the Back Transliteration for better word embedding (Dowlagar, 2021). Initially the 'text' column is tokenized based on the whitespace and moved into a new column name 'Wordset', with a list of words from the text column shown in the figure 13.

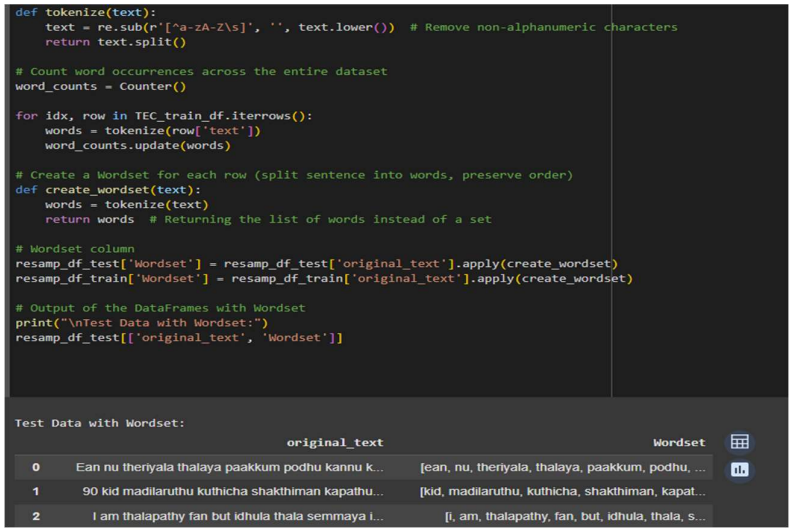


Figure 13: Tokenization of text based on white-space

The tokenized words are then back transliterated. This process converts the text back to the original script which is Tamil in this case. The transliterate tool 'XlitEngine' from the library 'ai4bharat-transliteration' (Dowlagar, 2021) is used for this. This model converts the tokenized words from the 'Wordnet' to Tamil scripts and stores it in a new column named 'Back_trans' as shown in the figure 14.

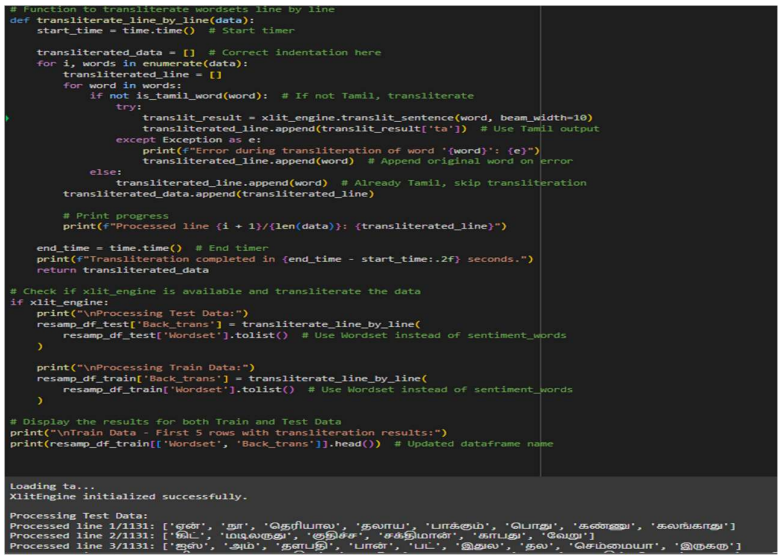


Figure 14: Back-Transliteration of the Wordset

Based on the words dictionary from 'nltk' library, the proper English words are identified and a new column named 'combined words' is created. This has the English and Tamil words in respective scripts from the 'Back_trans' column shown in the figure 15.

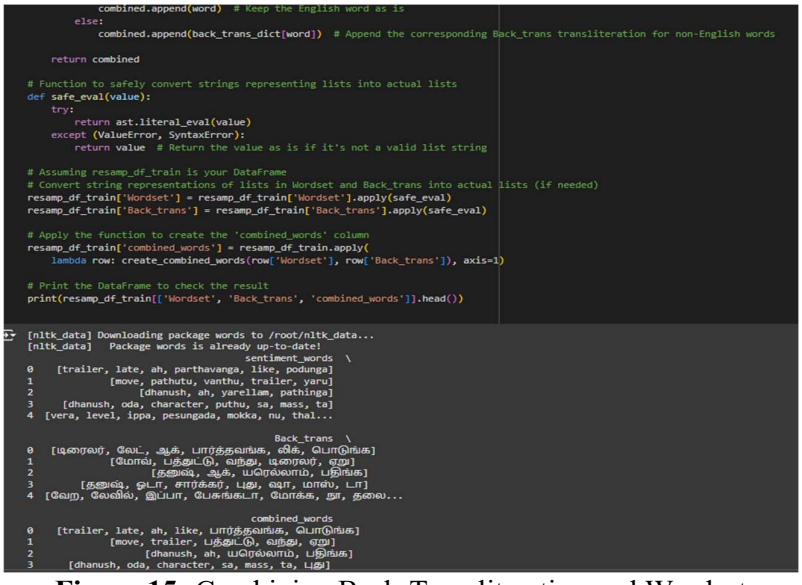


Figure 15: Combining Back-Transliteration and Wordset

The final column 'combined words' has text data with both Tamil and English in the proper script shown in the figure 16.

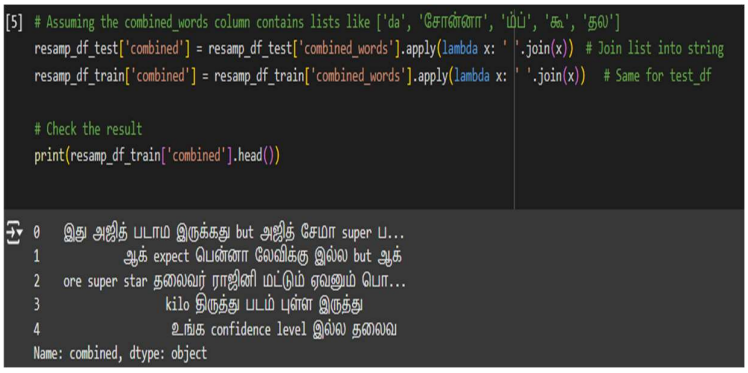


Figure 16: Combined column with Tamil and English words

3.5 Handling Class Imbalance

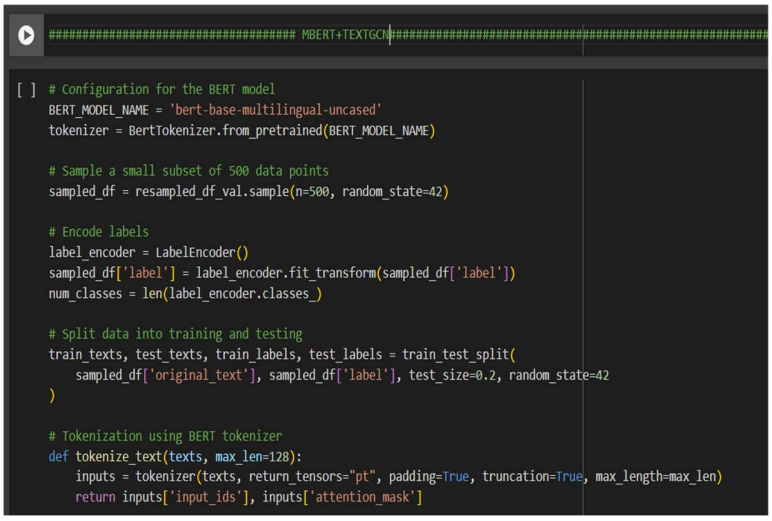
Given the dataset is highly imbalanced, so the data is being resampled by using the random under-sampling technique as it showed a better outcome after experimenting with multiple resampling techniques. As suggested by (Banerjee, 2020; Chakravarthi, 2020), the 'unknown_state' and 'non-tamil' labels contribute significantly less of total data and also introduces noise into data, so those two labels were ignored. The majority three classes are being resampled using random under sampling mentioned by (Gokhale, 2022) as shown in the figure 17.

Figure 17: Resampling using Under sampling

3.6 Model Building and Training

3.6.1 MBERT+TEXTGCN (BASE MODEL)

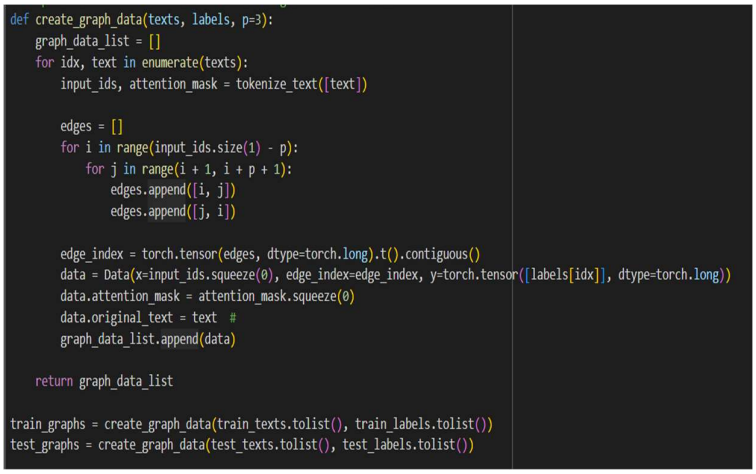
The mBERT model is used as the base model for all experimenting. The feature extraction process is performed by the multilingual BERT tokenizer to create word embeddings with padding (Dowlagar, 2021). The label is encoded using label encoder function as shown in the figure 18.



```
################################## MBERT+TEXTGCN#########################################################################

[ ]  # Configuration for the BERT model
     BERT_MODEL_NAME = 'bert-base-multilingual-uncased'
     tokenizer = BertTokenizer.from_pretrained(BERT_MODEL_NAME)

     # Sample a small subset of 500 data points
     sampled_df = resampled_df_val.sample(n=500, random_state=42)

     # Encode labels
     label_encoder = LabelEncoder()
     sampled_df['label'] = label_encoder.fit_transform(sampled_df['label'])
     num_classes = len(label_encoder.classes_)

     # Split data into training and testing
     train_texts, test_texts, train_labels, test_labels = train_test_split(
         sampled_df['original_text'], sampled_df['label'], test_size=0.2, random_state=42
     )

     # Tokenization using BERT tokenizer
     def tokenize_text(texts, max_len=128):
         inputs = tokenizer(texts, return_tensors="pt", padding=True, truncation=True, max_length=max_len)
         return inputs['input_ids'], inputs['attention_mask']
```

Figure 18: Feature Extraction and Label encoding for mBERT

The graph data is created with the word embedding token, generated by the mBERT Tokenizer. Figure 19 shows the token as the node of the graph data and the edges are the relationships between them.



```
def create_graph_data(texts, labels, p=3):
    graph_data_list = []
    for idx, text in enumerate(texts):
        input_ids, attention_mask = tokenize_text([text])

        edges = []
        for i in range(input_ids.size(1) - p):
            for j in range(i + 1, i + p + 1):
                edges.append([i, j])
                edges.append([j, i])

        edge_index = torch.tensor(edges, dtype=torch.long).t().contiguous()
        data = Data(x=input_ids.squeeze(0), edge_index=edge_index, y=torch.tensor([labels[idx]], dtype=torch.long))
        data.attention_mask = attention_mask.squeeze(0)
        data.original_text = text  #
        graph_data_list.append(data)

    return graph_data_list

train_graphs = create_graph_data(train_texts.tolist(), train_labels.tolist())
test_graphs = create_graph_data(test_texts.tolist(), test_labels.tolist())
```

Figure 19: Creating graph data with mBERT Tokens

As shown in the figure 20, mBERT pretrained model tokenizes the word into embeddings. Based on this embedding, graph data is generated which then passed into three layers of GCN to process the inter-word relationship. The first two layers identify first and second order relationships. The final fully connected SoftMax layer captures and predicts the sentiment (Dowlagar, 2021).

```python
class MBERT_TextLevelGCN(nn.Module):
    def __init__(self, bert_model_name, hidden_dim, num_classes):
        super(MBERT_TextLevelGCN, self).__init__()
        self.bert = BertModel.from_pretrained(bert_model_name)
        self.gcn1 = nn.Linear(768, hidden_dim)
        self.gcn2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc1 = nn.Linear(hidden_dim, hidden_dim // 2)
        self.fc2 = nn.Linear(hidden_dim // 2, num_classes)

        # Weight initialization
        nn.init.xavier_uniform_(self.gcn1.weight)
        nn.init.xavier_uniform_(self.gcn2.weight)
        nn.init.xavier_uniform_(self.fc1.weight)
        nn.init.xavier_uniform_(self.fc2.weight)

    def forward(self, data):
        x, edge_index, attention_mask = data.x, data.edge_index, data.attention_mask
        outputs = self.bert(input_ids=x.unsqueeze(0), attention_mask=attention_mask.unsqueeze(0))['last_hidden_state']

        x = F.relu(self.gcn1(outputs))
        x = F.relu(self.gcn2(x))

        x = torch.max(x, dim=1).values  # Use max pooling
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=-1)

# Instantiate model
model = MBERT_TextLevelGCN(bert_model_name=BERT_MODEL_NAME, hidden_dim=128, num_classes=num_classes)
```

Figure 20: mBERT + Text GCN Model.

Figure 21 shows the model is trained with cross-entropy function and optimized with adam optimizer to reduce prediction errors.

```python
def train_model(model, train_loader, optimizer, criterion, device):
    model.train()
    total_loss = 0
    for batch in train_loader:
        batch = batch.to(device)
        optimizer.zero_grad()
        output = model(batch)
        loss = criterion(output, batch.y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(train_loader)
```

Figure 21: Model training function of mBERT+TextGCN.

Figure 22 shows the evaluation is done by the test data to evaluate the metrics such as precision, recall, F1-score and the confusion matrix.

```python
def evaluate_model(model, test_loader, criterion, device):
    model.eval()
    total_loss, correct = 0, 0
    all_preds, all_labels = [], []
    classified_outcomes = []  # List to store classified outcomes

    with torch.no_grad():
        for batch in test_loader:
            batch = batch.to(device)
            output = model(batch)
            loss = criterion(output, batch.y)
            total_loss += loss.item()

            pred = output.argmax(dim=1)
            correct += pred.eq(batch.y).sum().item()
            all_preds.extend(pred.cpu().numpy())
            all_labels.extend(batch.y.cpu().numpy())

            # Store classified outcomes (text, true label, predicted label)
            for i in range(len(batch.y)):
                classified_outcomes.append({
                    'text': batch.original_text[i],  # Store the original text
                    'true_label': batch.y[i].item(),
                    'predicted_label': pred[i].item(),
                })

    accuracy = correct / len(test_loader.dataset)
    precision = precision_score(all_labels, all_preds, average='weighted', zero_division=1)
    recall = recall_score(all_labels, all_preds, average='weighted', zero_division=1)
    f1 = f1_score(all_labels, all_preds, average='weighted', zero_division=1)
    conf_matrix = confusion_matrix(all_labels, all_preds)
    class_report = classification_report(all_labels, all_preds, target_names=[str(cls) for cls in label_encoder.classes_], zero_division=1)

    return total_loss / len(test_loader), accuracy, precision, recall, f1, conf_matrix, class_report, classified_outcomes
```

Figure 22: Model Evaluation of mBERT+TextGCN.

The figure 23 shows the model training process with epoch, train loss and other metrics.

```
Epoch 5/5, Train Loss: 0.3370, Test Loss: 0.8909, Test Accuracy: 0.6600
Precision: 0.6506, Recall: 0.6600, F1-Score: 0.6373
```

Figure 23: Model Training of mBERT+TextGCN.

The figure 24 shows the classification report of the model for each class.

Classification Report:

	precision	recall	f1-score	support
0	0.44	0.19	0.27	21
1	0.80	0.67	0.73	18
2	0.54	0.68	0.60	22
3	0.61	0.83	0.70	24
4	1.00	1.00	1.00	15
accuracy			0.66	100
macro avg	0.68	0.67	0.66	100
weighted avg	0.65	0.66	0.64	100

Figure 24: Classification Report mBERT +TextGCN.

The figure 25 shows the Confusion matrix of the model classification for each class.

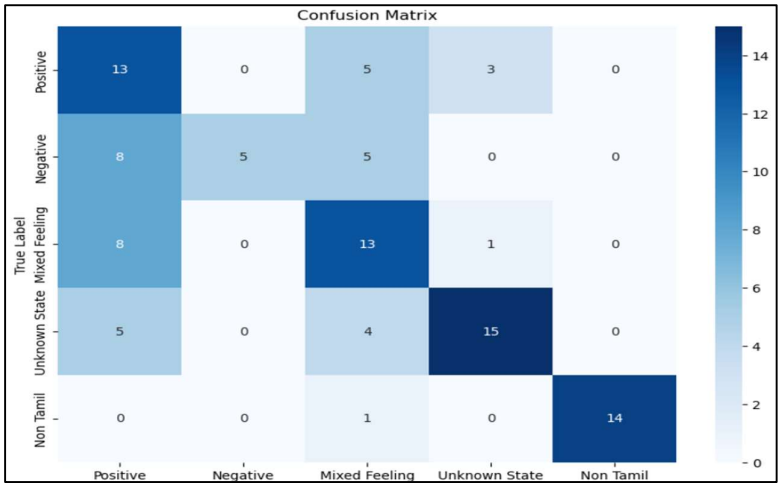


Figure 25: Confusion Matrix mBERT+TextGCN.

3.6.2 INDICBART+TEXTGCN (PROPOSED MODEL)

The proposed model IndicBART+TextGCN performs a feature extraction process by the multilingual IndicBART tokenizer. This is to create word embeddings with language tags that improves the embedding process (Dabre, 2022). The label is then encoded using label encoder function as shown in the figure 26.

```python
# Configuration for the IndicBART model
MODEL_NAME = "ai4bharat/IndicBART"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
indic_bart_model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME).to(torch.device('cuda' if torch

# Load datasets
ta_df = resamp_df_train
tb_df = resamp_df_test

# Combine datasets for consistency
tc_df = pd.concat([ta_df, tb_df], axis=0)
ta_df, tb_df = train_test_split(tc_df, test_size=0.2, random_state=42)

# Encode labels
label_encoder = LabelEncoder()
ta_df['label'] = label_encoder.fit_transform(ta_df['label'])
tb_df['label'] = label_encoder.transform(tb_df['label'])
num_classes = len(label_encoder.classes_)

texts_a = ta_df['combined'].tolist()
labels_a = ta_df['label'].tolist()
texts_b = tb_df['combined'].tolist()
labels_b = tb_df['label'].tolist()
```

Figure 26: Feature Extraction and Label encoding for IndicBART+TextGCN.

The embedding token generated by the IndicBART Tokenizer is used to create graph data. Figure 27 shows the node of the graph data as the token while the edges are the relationships between them.

```python
def create_graph_data(texts, labels, p=3):
    graph_data_list = []
    for idx, text in enumerate(texts):
        input_ids, attention_mask = tokenize_text([text])
        edges = []
        for i in range(input_ids.size(1) - p):
            for j in range(i + 1, i + p + 1):
                edges.append([i, j])
                edges.append([j, i])

        edge_index = torch.tensor(edges, dtype=torch.long).t().contiguous()
        data = Data(x=input_ids.squeeze(0), edge_index=edge_index, y=torch.tensor([labels[idx]], dtype=torch.long))
        data.attention_mask = attention_mask.squeeze(0)
        data.original_text = text
        graph_data_list.append(data)

    return graph_data_list
```

Figure 27: Creating graph data with IndicBART Tokens.

As shown in the Figure 28, IndicBART pretrained model tokenizes the word and the language tags with the appropriate language for better contextual understanding. Based on this, graph data is generated which then passed into three layers of GCN to process the inter-word relationship. The first two layers identify first and second order relationships. The final fully connected SoftMax layer captures and predicts the sentiment (Dowlagar, 2021).

```python
# Model Definition
class IndicBART_TextLevelGCN(nn.Module):
    def __init__(self, hidden_dim, num_classes):
        super(IndicBART_TextLevelGCN, self).__init__()
        self.indic_bart = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME)
        self.gcn1 = nn.Linear(1024, hidden_dim)
        self.gcn2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc1 = nn.Linear(hidden_dim, hidden_dim // 2)
        self.fc2 = nn.Linear(hidden_dim // 2, num_classes)

        # Weight initialization
        nn.init.xavier_uniform_(self.gcn1.weight)
        nn.init.xavier_uniform_(self.gcn2.weight)
        nn.init.xavier_uniform_(self.fc1.weight)
        nn.init.xavier_uniform_(self.fc2.weight)

    def forward(self, data):
        x, edge_index, attention_mask = data.x, data.edge_index, data.attention_mask
        encoder_outputs = self.indic_bart(input_ids=x.unsqueeze(0), attention_mask=attention_mask.unsqueeze(0)).encoder_last_hidden_state
        x = F.relu(self.gcn1(encoder_outputs))
        x = F.relu(self.gcn2(x))
        x = torch.max(x, dim=1).values
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=-1)
```

Figure 28: IndicBART Model.

Figure 29 shows the model training function which has cross-entropy and is optimized with adam optimizer to reduce prediction errors. The evaluation is done using test data to evaluate the metrics such as precision, recall, F1-score and the confusion matrix.

```python
# Training and Evaluation Functions
def train_model(model, train_loader, optimizer, criterion, device):
    model.train()
    total_loss = 0
    for batch in train_loader:
        batch = batch.to(device)
        optimizer.zero_grad()
        output = model(batch)
        loss = criterion(output, batch.y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(train_loader)

def evaluate_model(model, test_loader, criterion, device):
    model.eval()
    total_loss, correct = 0, 0
    all_preds, all_labels = [], []

    with torch.no_grad():
        for batch in test_loader:
            batch = batch.to(device)
            output = model(batch)
            loss = criterion(output, batch.y)
            total_loss += loss.item()

            pred = output.argmax(dim=1)
            correct += pred.eq(batch.y).sum().item()
            all_preds.extend(pred.cpu().numpy())
            all_labels.extend(batch.y.cpu().numpy())

    accuracy = correct / len(test_loader.dataset)
    return total_loss / len(test_loader), accuracy, all_preds, all_labels
```

Figure 29: IndicBART Model Training and Evaluation.

In order to avoid the overfitting of the model, early stop is used based on the test loss. Figure 30 shows the function used to get classification reports and visualize confusion matrix performed.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

EPOCHS = 50

early_stopping = EarlyStopping(patience=5, delta=0.01)

for epoch in range(EPOCHS):
    start_time = time.time()

    train_loss = train_model(model, train_loader, optimizer, criterion, device)
    test_loss, test_acc, _, _ = evaluate_model(model, train_loader, criterion, device)

    epoch_time = time.time() - start_time
    total_time += epoch_time  # Add current epoch time to total time

    print(f'Epoch {epoch + 1}/{EPOCHS}, Loss: {test_loss:.4f}, Accuracy: {test_acc:.4f}, Time for epoch: {epoch_time:.2f}s, Total Time: {total_time:.2f}s')

    if early_stopping(test_loss):  # Early stopping
        print(f"Early stop triggered at epoch {epoch + 1}.")
        break

# Display final epoch's detailed metrics
print("\nFinal Results (Epoch {}/{}):".format(best_epoch, EPOCHS))
print(f"Loss: {best_metrics['test_loss']:.4f}, Test Accuracy: {best_metrics['test_acc']:.4f}")
print(f"Precision: {best_metrics['precision']:.4f}, Recall: {best_metrics['recall']:.4f}, F1-Score: {best_metrics['f1']:.4f}")
print("Confusion Matrix:")
print(best_metrics['conf_matrix'])
print("Classification Report:")
print(best_metrics['class_report'])

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=best_metrics['conf_matrix'], display_labels=label_encoder.classes_)
disp.plot(cmap=plt.cm.Blues, values_format='d')
plt.title("Confusion Matrix for Final Epoch")
plt.show()
```

Figure 30: Functions for Early stopping and Classification report.

Figure 31 shows the early stop triggered at epoch 22 and the evaluation metrics such as precision, recall, F1-score, and the accuracy of the IndicBART+Text GCN model.

```
Epoch 22/50
Loss: 0.6947, Test Accuracy: 0.6839
Epoch Time: 441.19s
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.69      0.73      1660
           1       0.60      0.90      0.72      1660
           2       0.76      0.46      0.57      1660

    accuracy                           0.68      4980
   macro avg       0.71      0.68      0.67      4980
weighted avg       0.71      0.68      0.67      4980

Confusion Matrix:
[[1142  356  162]
 [  78 1500   82]
 [ 231  665  764]]
Early stopping at epoch 22.
```

Figure 31: Classification report for the IndicBART+TextGCN Model.

Figure 32 shows the confusion matrix of three classes of sentiment classified by IndicBART+Text GCN.

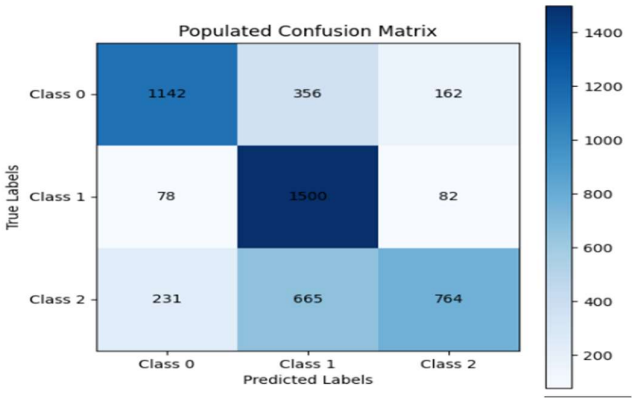


Figure 32: Confusion matrix for the IndicBART Model.

References

Banerjee, S. J. (2020). Sentiment Analysis of Code-Mixed Dravidian text using XLNet. *Proceedings of the FIRE 2020 Working Notes Track on Dravidian-CodeMix.* CEUR Workshop Proceedings. Retrieved from https://doi.org/10.48550/arXiv.2010.07773

Chakravarthi, B. M. (2020). Corpus creation for sentiment analysis in code-mixed Tamil-English text. *Proceedings of the 1st Joint SLTU and CCURL Workshop (SLTU-CCURL 2020), at the Language Resources and Evaluation Conference (LREC 2020), Marseille, France.* European Language Resources Association (ELRA). Retrieved from https://aclanthology.org/2020.sltu-1.28.pdf

Dabre, R. S. (2022). IndicBART: A pre-trained model for Indic natural language generation. *Findings of the Association for Computational Linguistics: ACL 2022.* Association for Computational Linguistics. Retrieved from https://doi.org/10.18653/v1/2022.findings-acl.145

Dowlagar, S. &. (2021). Graph Convolutional Networks with Multi-headed Attention for Code-Mixed Sentiment Analysis. *Proceedings of the First Workshop on Speech and Language Technologies for Dravidian Languages (DravidianLangTech 2021)* (pp. 59–67). Association for Computational Linguistics. Retrieved from https://aclanthology.org/2021.dravidianlangtech-1.8.pdf

Gokhale, O. P. (2022). Optimize_Prime@DravidianLangTech-ACL2022: Emotion Analysis in Tamil. *Association for Computational Linguistics.* ACL 2022 DravidianLangTech Workshop. Retrieved from https://aclanthology.org/2022.dravidianlangtech-1.35