Configuration Manual

MSc Research Project
Data Analytics

Suba Sri Ramesh Babu
Student ID: X21100241

School of Computing
National College of Ireland

Supervisor: Taimur Hafeez

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Suba Sri Ramesh Babu
Student ID:	X21100241
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Taimur Hafeez
Submission Due Date:	15/08/2022
Project Title:	Sentiment Analysis In Tamil Language Using Hybrid Deep Learning Approach
Word Count:	983
Page Count:	14

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:	Suba Sri Ramesh Babu
Date:	14th August 2022

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

Suba Sri Ramesh Babu
X21100241

1 Introduction

This configuration manual aims to provide a step-by-step procedure in the development and implementation of the research project which aims to classify the sentiment of the Tamil movie reviews. It provides an overview of the various hardware and software requirements that are involved in setting up the working environment to run the code smoothly. It also explains about the programming language used and libraries used in pre-processing the Tamil text. This manual also provides an overview of the various experiments performed in this research and results with evaluation metrics.

2 System Configurations

2.1 Hardware Configuration

The configration of the Hardware to build the research:

- **Device Name:** MacBook Pro
- **Operating System:** MacOSBigSurOS
- **Processor:** 2.3GHz Dual-Core Intel Core i5
- **RAM:** 16GB
- **Number of Core:** 2
- **Graphic Type:** intel iris Plus Graphics 640 1536 MB

2.2 Software Configuration

- **IDE:** Google Colabatory (Cloud Based Jupyter Notebook)
- **Programming Language:** MacOSBigSurOS
- **Web Browser:** Google Chrome
- **Documentation:** Overleaf

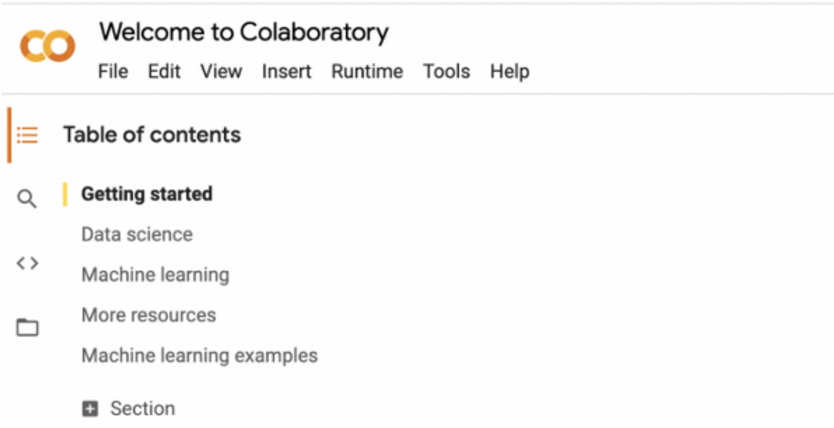

Figure 1: Getting started with Google Colaboratory.

The First step is setting up the Google Colaboratory environment to develop the code which was shown in figure 1. To access the environment, we need a Google account to sign in.

In the collab notebook, first we have mounted the drive to use the dataset and other tools from the drive.

After that all the necessary libraries listed below were imported.

- pandas
- numpy
- nltk
- seaborn
- tensorflow
- fastext
- keras
- matplotlib
- sklearn
- indicnlp

2.3 Data Source

The dataset used for this project is collected from the Kaggle repository [1] shown in figure 2. The dataset contains the Tamil movie reviews which was given in Tamil language and ratings. The raw data has noise such as punctuation, and unnecessary words to learn the meaning of the sentence by the model. So, next step involves data preparation.

[1] https://www.kaggle.com/datasets/sudalairajkumar/tamil-nlp?select=tamil_movie_reviews_train.csv

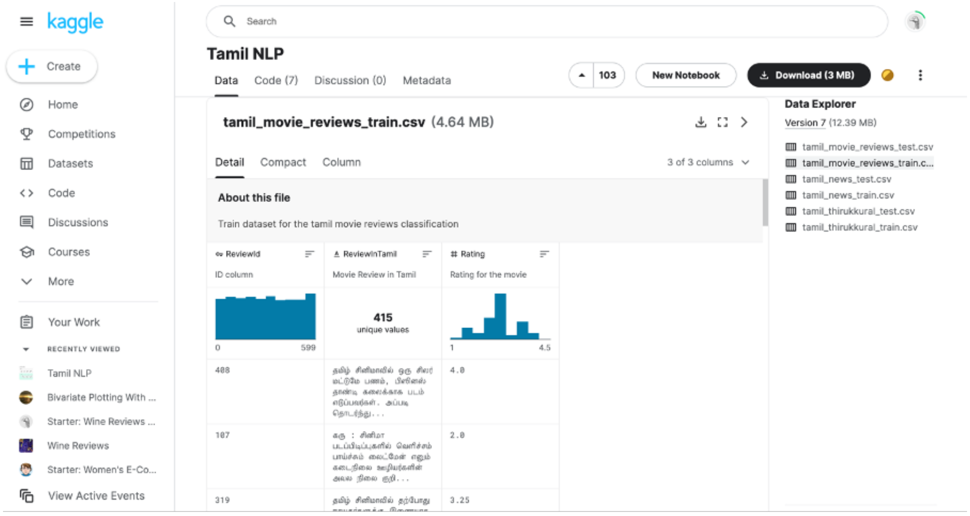

Figure 2: Dataset from Kaggle.

3 Implementation

The following section aims to provide an overview of the various steps involved in the implementation of the research work.

These include the data preparation, feature extraction and the proposed models implementation.

The figure 3 shows the necessary libraries imported for this research.

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import nltk
    import seaborn as sns
    import tensorflow as tf
    import re
    from nltk.tokenize import word_tokenize
    from nltk.corpus import stopwords
    from indicnlp import common
    from indicnlp import loader
    from indicnlp.morph import unsupervised_morph
    import fasttext
    import fasttext.util
    from tensorflow.keras.optimizers import Adam, SGD
    from tensorflow.keras.callbacks import TensorBoard
    from keras import backend as K
    from keras.preprocessing.text import Tokenizer,sequence
    from keras.preprocessing.sequence import pad_sequences
    from keras.models import Model
    from keras.models import Sequential
    from keras.layers.embeddings import Embedding
    from keras import optimizers
    from keras.layers import SpatialDropout1D
    from keras.layers import TimeDistributed,Conv1D,Dense,Embedding,Input,Dropout,LSTM,Bidirectional,MaxPooling1D,Flatten,concatenate,GRU
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_auc_score,roc_curve,precision_recall_curve
```

Figure 3: Imported Libraries.

3.1 Data Preparation

The datasets available in Kaggle contains train and test data. The datasets were uploaded from the google drive.

The figure 4 and figure 5 shows the train data and test data loading into pandas dataframe.

Train Data:

```
[ ] data = pd.read_csv('./drive/MyDrive/tamil_movie_reviews_train.csv')
    data.head()
```

Figure 4: Train Data.

Test Data:

```
[ ] test = pd.read_csv('./drive/MyDrive/tamil_movie_reviews_test.csv')
    test.head()
```

Figure 5: Test Data.

Before pre-processing we are merging train and test data which was shown in the figure 6.

Merging Data for Data cleansing process.

```
[ ] data = data.append(test)
    df = data
    df.head()
    df.shape

    (601, 3)
```

Figure 6: Merging Train and Test Data.

Sample data after merging process complete shown in figure 7.

```
df.head()
```

	ReviewId	ReviewInTamil	Rating
0	408	தமிழ் சினிமாவில் ஒரு சிலர் மட்டுமே பணம், பிஸின...	4.00
1	107	கரு : சினிமா படப்பிடிப்புகளில் வெளிச்சம் பாய்ச...	2.00
2	319	தமிழ் சினிமாவில் தற்போது நாயகர்களுக்கு இணையாக ...	3.25
3	484	உலக அளவில் அனைத்து தரப்பினரையும் தன் நடிப்பால்...	2.25
4	204	கரு : வில்லனின் கையாள் , வில்லன் செய்த நம்பிக்...	3.00

Figure 7: Sample of Dataset.

The raw data collected from the internet which was provided by different peoples might contains more noise and unwanted symbols. The following figure 8 shows the removal of punctuations and tokenisation each sentence into words.

```
[14] def punctuation_remove(text_data):
         # Appending non punctuated words
         punctuation ="".join([t for t in text_data if t not in string.punctuation])
         return punctuation
```

Figure 8: Removing Puncutations and Word Tokenisation.

Output after removing punctuations and tokenisation was shown in the figure 9.

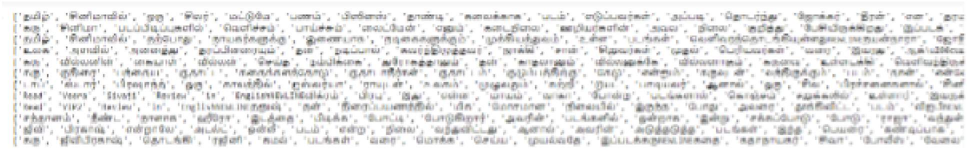

Figure 9: Output after Removing Punctuation and Word Tokenisation.

Performing morphological analysis for formalized linguistics structure. Morphological analysis was done using the indic NLP library which was imported at starting of the code. Using IndicNLP resources (Fernando and Wijayasiriwardhane; 2020)each tokenised word was morphologically analysed and split into separate tokens which explains in figure 10.

```
INDIC_NLP_RESOURCES = r"./drive/MyDrive/indic_nlp_resources-master"

# Initialize the Indic NLP library
loader.load()

# Morphological Analyser
analyzer = unsupervised_morph.UnsupervisedMorphAnalyzer('ta')
```

```
[ ] final = []
    for t in token_list:
      new = []
      for a in t:
        ma= analyzer.morph_analyze_document(a.split(' '))
        #print(ma)
        new.append(ma)
      fl= flatten_list(new)
      final.append(fl)
```

Figure 10: Morphological Analysis.

Sample output after morphological analysis we got is shown in figure 11.

Figure 11: Output of Morphological Analysis.

The list of stop words were provided by TamilNLP resources which contains 125 Tamil stopwords and was downloaded for github. The figure 12 shows the function created to load the stopword in Tamil and the data after stopwords got removed was stored in text variable.

```python
def __remove_stopwords(query):
    parsed = []

    #Opening File to load the stopwords in tamil
    with open('./drive/MyDrive/TamilStopWords.txt', encoding="utf8") as file:
        contents = file.read()

    for word in query:
        if word not in contents:
            parsed.append(word)

    return parsed
```

```python
text= []
for a in final:
  stopwords_removed = __remove_stopwords(a)
  stopwords_removed = ' '.join(stopwords_removed)
  text.append(stopwords_removed)
```

Figure 12: Stopword Removal.

In the next step to make the input in same size and shape the process of padding was carried out. The figure 13 illustrate the process of padding.

```python
[ ]  d = pad_sequences(X, maxlen=max_length)
     print('Shape of data tensor:', d.shape)

     Shape of data tensor: (601, 1150)

     print(d)

     [[     0      0      0 ...     39     13    325]
      [     0      0      0 ...   4952    152     51]
      [     0      0      0 ...   2678    367   2134]
      ...
      [     0      0      0 ...    424   1043   1043]
      [     0      0      0 ...    137     15  24798]
      [     0      0      0 ...   1602   1392    502]]
```

Figure 13: Padding.

3.2 Data Transformation

The blocks in figure 14 explains the way the rating column get transformed into binary labelled column as 0 and 1 which is negative and positive respectively.

```
[ ] y = np.zeros_like(rating)
    y[rating>3] = 1
```

```
[ ] print(y, len(y))
```

```
[1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0.
 1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0.
 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 0.
 1. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0.
 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1.
 0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 1. 0.
 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1.
 0. 0. 0. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1.
 0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1.
 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.
 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0.
 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1.
 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0.
 0.] 601
```

Figure 14: Converting Rating into Binary classification.

3.3 Feature Extraction

The next step is feature extraction. This process involves transforming each word into vectors. For this fastText Word Embedding (Senevirathne et al.; 2020) can be used. In this research work the fasttext model which was pre-trained on Tamil was used. Figure 15shows the importing of fasttext and figure 16 shows embedding matrix created using fastext.

```
[ ] import fasttext
    import fasttext.util
```

```
[ ] ft = fasttext.load_model('./drive/MyDrive/cc.ta.300.bin')
```

Figure 15: Importing fasttext.

```
[ ] unique_words = len(word_index)
    total_words = unique_words + 1
    skipped_words =0
    embedding_dim = 300
    embedding_matrix = np.zeros((total_words, embedding_dim))
    for word, index in tokenizer.word_index.items():
      try:
        embedding_vector = ft[word]
      except:
        skipped_words = skipped_words+1
        pass
      if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
    print("Embedding Matrix shape : ",embedding_matrix.shape)

    Embedding Matrix shape :  (24826, 300)
```

Figure 16: Building Embedding Matrix.

The figure 17 shows the building of embedding layer.

```
[ ] embedding_layer = Embedding(total_words, embedding_dim, weights=[embedding_matrix], input_length=max_length, trainable= False)
```

Figure 17: Embedding Layer.

After the data pre-processing steps the data was splitted into train and test data with the ratio of 80:20 that explained in figure 18.

```
[ ] train_features, test_features, train_labels, test_labels = train_test_split(d, Y, test_size=.20)
```

Figure 18: Splitting data into Train and Test.

3.4 Model Building

3.4.1 CNN-LSTM

BUILDING CNN-LSTM MODEL

```
[ ] model_1 = Sequential()
    model_1.add(embedding_layer)
    model_1.add(SpatialDropout1D(0.2))
    model_1.add(Conv1D(filters = 32, kernel_size = 1, activation='relu', padding='same'))
    model_1.add(MaxPooling1D(pool_size=2, strides= None, padding='valid'))
    model_1.add(Conv1D(filters = 64, kernel_size = 1, activation='relu', padding='same'))
    model_1.add(MaxPooling1D(pool_size=2, strides= None, padding='valid'))
    #model_3.add(Activation('relu'))

    model_1.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
    model_1.add(Dense(2, activation='softmax'))
    model_1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 19: Building CNN-LSTM.

We also included early stopping shown in figure 20to prevent the model from overfitting.

```
[ ] callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
```

Figure 20: Early Stopping.

After CNN-LSTM model was built we need to train the model on the data on training data for that we have tuned the hyperparameters and introduce early stopping aswell that was shown in the figure 21.

```
history_1 = model_1.fit(train_features, train_labels, epochs=20, batch_size=32,validation_split=0.2, callbacks= [callback])

Epoch 1/20
12/12 [==============================] - 19s 1s/step - loss: 0.6085 - accuracy: 0.7214 - val_loss: 0.5767 - val_accuracy: 0.7500
Epoch 2/20
12/12 [==============================] - 13s 1s/step - loss: 0.5845 - accuracy: 0.7448 - val_loss: 0.5861 - val_accuracy: 0.7500
Epoch 3/20
12/12 [==============================] - 14s 1s/step - loss: 0.5697 - accuracy: 0.7448 - val_loss: 0.5713 - val_accuracy: 0.7500
Epoch 4/20
12/12 [==============================] - 16s 1s/step - loss: 0.5715 - accuracy: 0.7448 - val_loss: 0.5685 - val_accuracy: 0.7500
Epoch 5/20
12/12 [==============================] - 8s 640ms/step - loss: 0.5630 - accuracy: 0.7448 - val_loss: 0.5720 - val_accuracy: 0.7500
Epoch 6/20
12/12 [==============================] - 8s 714ms/step - loss: 0.5591 - accuracy: 0.7448 - val_loss: 0.5648 - val_accuracy: 0.7500
Epoch 7/20
12/12 [==============================] - 8s 633ms/step - loss: 0.5528 - accuracy: 0.7448 - val_loss: 0.5639 - val_accuracy: 0.7500
Epoch 8/20
12/12 [==============================] - 8s 638ms/step - loss: 0.5455 - accuracy: 0.7448 - val_loss: 0.5609 - val_accuracy: 0.7500
Epoch 9/20
12/12 [==============================] - 8s 712ms/step - loss: 0.5330 - accuracy: 0.7448 - val_loss: 0.5584 - val_accuracy: 0.7500
Epoch 10/20
12/12 [==============================] - 8s 641ms/step - loss: 0.5187 - accuracy: 0.7474 - val_loss: 0.5597 - val_accuracy: 0.7500
Epoch 11/20
12/12 [==============================] - 8s 638ms/step - loss: 0.5133 - accuracy: 0.7552 - val_loss: 0.5662 - val_accuracy: 0.7500
Epoch 12/20
12/12 [==============================] - 8s 718ms/step - loss: 0.4617 - accuracy: 0.7708 - val_loss: 0.5396 - val_accuracy: 0.7708
Epoch 13/20
12/12 [==============================] - 9s 694ms/step - loss: 0.4148 - accuracy: 0.7995 - val_loss: 0.5673 - val_accuracy: 0.6667
Epoch 14/20
12/12 [==============================] - 8s 644ms/step - loss: 0.3949 - accuracy: 0.8359 - val_loss: 0.5331 - val_accuracy: 0.6979
Epoch 15/20
12/12 [==============================] - 8s 630ms/step - loss: 0.3504 - accuracy: 0.8411 - val_loss: 0.5429 - val_accuracy: 0.7396
Epoch 16/20
12/12 [==============================] - 7s 615ms/step - loss: 0.3159 - accuracy: 0.8776 - val_loss: 0.5863 - val_accuracy: 0.7708
Epoch 17/20
12/12 [==============================] - 9s 722ms/step - loss: 0.3019 - accuracy: 0.8776 - val_loss: 0.6977 - val_accuracy: 0.7708
Epoch 18/20
12/12 [==============================] - 8s 648ms/step - loss: 0.2984 - accuracy: 0.8958 - val_loss: 0.5283 - val_accuracy: 0.7396
Epoch 19/20
12/12 [==============================] - 8s 641ms/step - loss: 0.2290 - accuracy: 0.9089 - val_loss: 0.5616 - val_accuracy: 0.7396
Epoch 20/20
12/12 [==============================] - 8s 645ms/step - loss: 0.1825 - accuracy: 0.9323 - val_loss: 0.6074 - val_accuracy: 0.7083
```

Figure 21: Training CNN-LSTM.

The figure 22 shows the evaluation of Train and test data and its accuracy.

```
[ ] train_scores_1 = model_1.evaluate(train_features, train_labels, verbose=0)
    print("Train %s: %.2f%%" % (model_3.metrics_names[1], train_scores_1[1]*100))

    Train accuracy: 92.29%

[ ] test_scores_1 = model_1.evaluate(test_features, test_labels, verbose=0)
    print("Test %s: %.2f%%" % (model_3.metrics_names[1], test_scores_1[1]*100))

    Test accuracy: 78.51%
```

Figure 22: Evaluation and Train and Test Accuracy for CNN-LSTM.

we have created a function to get the classification report and confusion matrix which was defined in the figure 23.

```
[ ] def modell_evaluate():
        # predict class with test set
        y_pred_test =  np.argmax(model_3.predict(test_features), axis=1)
        print('Accuracy:\t{:0.1f}%'.format(accuracy_score(np.argmax(test_labels,axis=1),y_pred_test)*100))

        #classification report
        print('\n')
        print(classification_report(np.argmax(test_labels,axis=1), y_pred_test))

        #confusion matrix
        confmat = confusion_matrix(np.argmax(test_labels,axis=1), y_pred_test)

        fig, ax = plt.subplots(figsize=(4, 4))
        ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
        for i in range(confmat.shape[0]):
            for j in range(confmat.shape[1]):
                ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center')
        plt.xlabel('Predicted label')
        plt.ylabel('True label')
        plt.tight_layout()
```

Figure 23: Creating function to summary of the model.

The figure 24 shows the classification Report and confusion matrix of CNN-LSTM.

```
[ ] model1_evaluate()

    Accuracy:       78.5%

                  precision    recall  f1-score   support

               0       0.83      0.90      0.87        93
               1       0.55      0.39      0.46        28

        accuracy                           0.79       121
       macro avg       0.69      0.65      0.66       121
    weighted avg       0.77      0.79      0.77       121
```

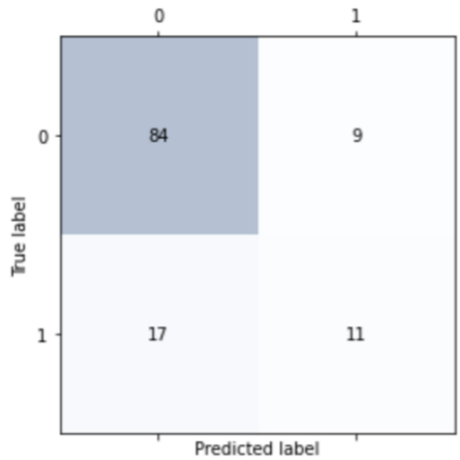

Figure 24: Classification Report and Confusion Matrix of CNN-LSTM.

3.4.2 CNN-BiLSTM

Building CNN-BiLSTM MODEL in the figure 25.

```
model_2 = Sequential()
model_2.add(embedding_layer)
model_2.add(SpatialDropout1D(0.2))
model_2.add(Conv1D(filters = 32, kernel_size = 1, activation='relu', padding='same'))
#model.add(BatchNormalization())
model_2.add(MaxPooling1D(pool_size=2, strides= None, padding='valid'))
model_2.add(Conv1D(filters = 64, kernel_size = 1, activation='relu', padding='same'))
model_2.add(MaxPooling1D(pool_size=2, strides= None, padding='valid'))
model_2.add(Activation('relu'))

model_2.add(Bidirectional(LSTM(150, return_sequences=True)))
model_2.add(Dropout(0.3))
model_2.add(Bidirectional(LSTM(96)))
model_2.add(Dropout(0.2))
#model_2.add(Dense(64,activation='sigmoid'))
model_2.add(Dense(32,activation='relu'))
#model_2.add(Flatten())
model_2.add(Dense(2,activation='sigmoid'))
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 25: Building CNN-BiLSTM.

Training the CNN-BiLSTM on training data in figure 26.

```
hist_2 = model_2.fit(train_features, train_labels, epochs=20, batch_size=32, validation_split=0.2, callbacks= [callback])

Epoch 1/20
12/12 [==============================] - 28s 2s/step - loss: 0.6085 - accuracy: 0.7135 - val_loss: 0.5785 - val_accuracy: 0.7500
Epoch 2/20
12/12 [==============================] - 20s 2s/step - loss: 0.5724 - accuracy: 0.7448 - val_loss: 0.5841 - val_accuracy: 0.7500
Epoch 3/20
12/12 [==============================] - 22s 2s/step - loss: 0.5748 - accuracy: 0.7448 - val_loss: 0.5828 - val_accuracy: 0.7500
Epoch 4/20
12/12 [==============================] - 20s 2s/step - loss: 0.5735 - accuracy: 0.7448 - val_loss: 0.5593 - val_accuracy: 0.7500
Epoch 5/20
12/12 [==============================] - 20s 2s/step - loss: 0.5577 - accuracy: 0.7448 - val_loss: 0.5673 - val_accuracy: 0.7500
Epoch 6/20
12/12 [==============================] - 21s 2s/step - loss: 0.5562 - accuracy: 0.7448 - val_loss: 0.5570 - val_accuracy: 0.7500
Epoch 7/20
12/12 [==============================] - 21s 2s/step - loss: 0.5376 - accuracy: 0.7448 - val_loss: 0.5556 - val_accuracy: 0.7500
Epoch 8/20
12/12 [==============================] - 20s 2s/step - loss: 0.5117 - accuracy: 0.7552 - val_loss: 0.5228 - val_accuracy: 0.7604
Epoch 9/20
12/12 [==============================] - 20s 2s/step - loss: 0.4932 - accuracy: 0.7760 - val_loss: 0.5210 - val_accuracy: 0.7604
Epoch 10/20
12/12 [==============================] - 22s 2s/step - loss: 0.4688 - accuracy: 0.8021 - val_loss: 0.5126 - val_accuracy: 0.7604
Epoch 11/20
12/12 [==============================] - 20s 2s/step - loss: 0.4331 - accuracy: 0.8125 - val_loss: 0.5255 - val_accuracy: 0.7396
Epoch 12/20
12/12 [==============================] - 20s 2s/step - loss: 0.4023 - accuracy: 0.8229 - val_loss: 0.5110 - val_accuracy: 0.7396
Epoch 13/20
12/12 [==============================] - 22s 2s/step - loss: 0.3059 - accuracy: 0.8750 - val_loss: 0.5617 - val_accuracy: 0.7708
Epoch 14/20
12/12 [==============================] - 20s 2s/step - loss: 0.3315 - accuracy: 0.8620 - val_loss: 0.5415 - val_accuracy: 0.7604
Epoch 15/20
12/12 [==============================] - 21s 2s/step - loss: 0.2553 - accuracy: 0.8932 - val_loss: 0.6193 - val_accuracy: 0.8021
Epoch 16/20
12/12 [==============================] - 21s 2s/step - loss: 0.2257 - accuracy: 0.9193 - val_loss: 0.6300 - val_accuracy: 0.7500
Epoch 17/20
12/12 [==============================] - 22s 2s/step - loss: 0.2094 - accuracy: 0.9167 - val_loss: 0.6423 - val_accuracy: 0.8021
Epoch 18/20
12/12 [==============================] - 21s 2s/step - loss: 0.2323 - accuracy: 0.9062 - val_loss: 0.4905 - val_accuracy: 0.7917
Epoch 19/20
12/12 [==============================] - 29s 2s/step - loss: 0.1879 - accuracy: 0.9401 - val_loss: 0.6203 - val_accuracy: 0.7604
Epoch 20/20
12/12 [==============================] - 22s 2s/step - loss: 0.1998 - accuracy: 0.9193 - val_loss: 0.5839 - val_accuracy: 0.8333
```

Figure 26: Training CNN-BiLSTM.

After training the CNN-BiLSTM Model, the model was then evaluated on train and test data. The accuracy the model got and the evaluation of CNN-BiLSTM shown in figure 27.

```
[ ] train_scores_2 = model_2.evaluate(train_features, train_labels, verbose=0)
    print("Train %s: %.2f%%" % (model_2.metrics_names[1], train_scores_2[1]*100))

    Train accuracy: 96.25%

[ ] test_scores_2 = model_2.evaluate(test_features, test_labels, verbose=0)
    print("Test %s: %.2f%%" % (model_2.metrics_names[1], test_scores_2[1]*100))

    Test accuracy: 80.99%
```

Figure 27: Evaluation and Train and Test Accuarcy for CNN-BiLSTM.

Classification Report and Confusion Matrix shows how well the model performance is. These summary of CNN-BiLSTM given in figure 28.

```
[ ] model2_evaluate()

Accuracy:        81.0%

              precision    recall  f1-score   support

           0       0.84      0.94      0.88        93
           1       0.65      0.39      0.49        28

    accuracy                           0.81       121
   macro avg       0.74      0.66      0.69       121
weighted avg       0.79      0.81      0.79       121
```

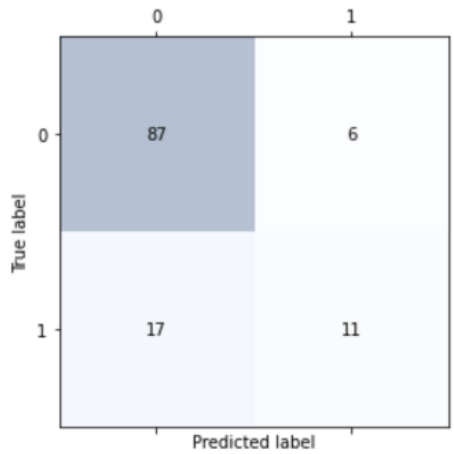

Figure 28: Classification Report and Confusion Matrix of CNN-BiLSTM.

3.5 CNN-BiGRU

Building CNN-BiGRU in the figure 29.

```
[ ] model_3 = Sequential()
    model_3.add(embedding_layer)
    model_3.add(SpatialDropout1D(0.2))
    model_3.add(Conv1D(filters = 64, kernel_size = 1, activation='relu', padding='same'))
    model_3.add(MaxPooling1D(pool_size=2, strides= None, padding='valid'))
    model_3.add(Activation('relu'))

    model_3.add(SpatialDropout1D(0.2))
    model_3.add(Bidirectional(GRU(75)))
    model_3.add(Dropout(0.2))
    model_3.add(Dense(2, activation='sigmoid'))
    model_3.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 29: Building CNN-BiGRU.

In figure 30 training CNN-BiLSTM model on train data with hyperparameters were shown.

```
history_3 = model_3.fit(train_features, train_labels, epochs=20, batch_size=32,validation_split=0.2, callbacks= [callback])

Epoch 1/20
12/12 [==============================] - 23s 1s/step - loss: 0.6069 - accuracy: 0.7318 - val_loss: 0.5724 - val_accuracy: 0.7500
Epoch 2/20
12/12 [==============================] - 15s 1s/step - loss: 0.5884 - accuracy: 0.7448 - val_loss: 0.5679 - val_accuracy: 0.7500
Epoch 3/20
12/12 [==============================] - 14s 1s/step - loss: 0.5668 - accuracy: 0.7448 - val_loss: 0.5732 - val_accuracy: 0.7500
Epoch 4/20
12/12 [==============================] - 13s 1s/step - loss: 0.5619 - accuracy: 0.7448 - val_loss: 0.5614 - val_accuracy: 0.7500
Epoch 5/20
12/12 [==============================] - 13s 1s/step - loss: 0.5588 - accuracy: 0.7448 - val_loss: 0.5591 - val_accuracy: 0.7500
Epoch 6/20
12/12 [==============================] - 10s 835ms/step - loss: 0.5519 - accuracy: 0.7448 - val_loss: 0.5555 - val_accuracy: 0.7500
Epoch 7/20
12/12 [==============================] - 8s 705ms/step - loss: 0.5390 - accuracy: 0.7448 - val_loss: 0.5495 - val_accuracy: 0.7500
Epoch 8/20
12/12 [==============================] - 10s 834ms/step - loss: 0.5311 - accuracy: 0.7448 - val_loss: 0.5418 - val_accuracy: 0.7500
Epoch 9/20
12/12 [==============================] - 8s 698ms/step - loss: 0.5072 - accuracy: 0.7448 - val_loss: 0.5291 - val_accuracy: 0.7500
Epoch 10/20
12/12 [==============================] - 8s 705ms/step - loss: 0.4927 - accuracy: 0.7604 - val_loss: 0.5145 - val_accuracy: 0.7500
Epoch 11/20
12/12 [==============================] - 8s 694ms/step - loss: 0.4553 - accuracy: 0.7891 - val_loss: 0.4993 - val_accuracy: 0.7500
Epoch 12/20
12/12 [==============================] - 8s 703ms/step - loss: 0.4219 - accuracy: 0.7969 - val_loss: 0.4701 - val_accuracy: 0.7604
Epoch 13/20
12/12 [==============================] - 8s 711ms/step - loss: 0.3657 - accuracy: 0.8203 - val_loss: 0.4542 - val_accuracy: 0.8229
Epoch 14/20
12/12 [==============================] - 8s 697ms/step - loss: 0.3472 - accuracy: 0.8438 - val_loss: 0.4383 - val_accuracy: 0.8021
Epoch 15/20
12/12 [==============================] - 9s 724ms/step - loss: 0.2776 - accuracy: 0.8932 - val_loss: 0.4409 - val_accuracy: 0.8333
Epoch 16/20
12/12 [==============================] - 10s 857ms/step - loss: 0.2279 - accuracy: 0.9062 - val_loss: 0.4176 - val_accuracy: 0.8229
Epoch 17/20
12/12 [==============================] - 8s 703ms/step - loss: 0.1982 - accuracy: 0.9245 - val_loss: 0.4206 - val_accuracy: 0.8125
Epoch 18/20
12/12 [==============================] - 9s 790ms/step - loss: 0.1728 - accuracy: 0.9401 - val_loss: 0.4633 - val_accuracy: 0.8125
Epoch 19/20
12/12 [==============================] - 14s 1s/step - loss: 0.1698 - accuracy: 0.9453 - val_loss: 0.5807 - val_accuracy: 0.8021
Epoch 20/20
12/12 [==============================] - 8s 686ms/step - loss: 0.1435 - accuracy: 0.9479 - val_loss: 0.4625 - val_accuracy: 0.8021
```

Figure 30: Training CNN-BiLSTM.

Evaluation of CNN-BiLSTM and accuracy for predicted train and test data was provided in the figure 31.

```
[ ] train_scores_3 = model_3.evaluate(train_features, train_labels, verbose=0)
    print("Train %s: %.2f%%" % (model_4.metrics_names[1], train_scores_3[1]*100))

    Train accuracy: 95.21%

[ ] test_scores_3 = model_3.evaluate(test_features, test_labels, verbose=0)
    print("Test %s: %.2f%%" % (model_4.metrics_names[1], test_scores_3[1]*100))

    Test accuracy: 79.34%
```

Figure 31: Evaluation and Train and Test Accuracy for CNN-BiLSTM.

The figure 32 explains the Classification Report and Confusion Matrix of CNN-BiGRU.

```
[ ] train_scores_3 = model_3.evaluate(train_features, train_labels, verbose=0)
    print("Train %s: %.2f%%" % (model_4.metrics_names[1], train_scores_3[1]*100))

    Train accuracy: 95.21%

[ ] test_scores_3 = model_3.evaluate(test_features, test_labels, verbose=0)
    print("Test %s: %.2f%%" % (model_4.metrics_names[1], test_scores_3[1]*100))

    Test accuracy: 79.34%
```

Figure 32: Classification Report and Confusion Matrix of CNN-BiGRU.

References

Fernando, A. and Wijayasiriwardhane, T. K. (2020). Identifying religious extremism-based threats in srilanka using bilingual social media intelligence, *2020 International*

Research Conference on Smart Computing and Systems Engineering (SCSE), IEEE, pp. 103–110.

Senevirathne, L., Demotte, P., Karunanayake, B., Munasinghe, U. and Ranathunga, S. (2020). Sentiment analysis for sinhala language using deep learning techniques, *arXiv preprint arXiv:2011.07280* .