HATE SPEECH DETECTION USING MACHINE LEARNING

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Updesh Kumar **(AP21110010860)**

Ayushman Singh **(AP21110010870)**

Under the Guidance of

(Elakkiya E)

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

Nov, 2023

Certificate

Date: 4-Dec-23

This is to certify that the work present in this Project entitled "**HATE SPEECH DETECTION USING MACHINE LEARNING**" has been carried out by **Updesh Kumar and Ayushman Singh** under my supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Dr. Elakkiya E

Assistant Professor,

Department of Computer Science Engineering, SRM University AP

Acknowledgements

We as a team would like to show our sincere gratitude to Dr. Elakkiya E for giving us a wonderful case study that is based on research and for their assistance in helping me finish my project, which was titled "Hate Speech Detection Using Machine Learning"

We would especially like to thank our supervisor for giving us valuable comments and advice that were quite beneficial to us as we finished the job. We will always be appreciative of you in this regard. We are thankful to the university for being hugely supportive during the course of the project. It gave us an opportunity to participate and learn about various machine learning models and training datasets using them.

Finally, We appreciate the assistance of our department & teaching and non-teaching staff in completing our project work. We would like to acknowledge that this case study review is completed entirely by our team on hate speech detection using machine learning.

Thank you all for making this research project a truly rewarding and unforgettable experience.

Sincerely,

Updesh Kumar

Ayushman Singh

Hate Speech Detection Using Machine Learning

SRM University Andhra Pradesh

Table of Contents

Abstract

This research report is dedicated to the precise and thorough examination of hate speech within the context of textual content, excluding audio and video mediums. In an era where written communication prevails across various digital platforms, the identification and management of hate speech in text have become paramount. This study adopts state-of-the-art deep learning techniques, specifically Long Short-Term Memory networks, to develop a robust hate speech detection model.

The research process entails the collection and curation of a representative Twitter dataset, carefully annotated to distinguish between hate speech and non-hate speech instances. Subsequently, advanced natural language processing (NLP) techniques and LSTM-based models are employed to extract meaningful features from the textual data, aiming to discern patterns indicative of hate speech.

Moreover, recognizing the nuances of hate speech influenced by context and cultural variations, the research integrates ensemble-based models, which combine multiple classifiers to enhance the accuracy and robustness of hate speech detection. Additionally, semantic word embeddings are captured using the Word2Vec method, further refining the detection system's capabilities.

The outcomes of this study offer a substantial contribution to the development of effective tools for combatting hate speech in digital environments, specifically within written communication channels. By automating the identification and flagging of hateful content, this research endeavours to foster responsible digital citizenship and safeguard individuals from the harmful consequences of hate speech. The insights gained from this report hold significant promise for enhancing content moderation strategies and promoting a more inclusive and secure online sphere, particularly in the realm of text-based communication.

Abbreviations

LSTM	Long Sort Term Memory
LSTM-SNP	Long Sort Term Memory Spiking Neural P
Inpt	Input
Outt	output
BiLSTM	Bidirectional LSTM
CNNs	Convolutional Neural Networks
RNNs	Recurrent Neural Networks

List of Tables

List of Figures

1. Introduction

Word is highly dependent on the verbal or written language to communicate among each other. Almost every work in day-to-day life can be communicated verbally but for official communication or for purpose authenticity, we prefer documented communication (basically written). The world is run by some sort of rule and regulation considering emotions too. Every country has a set of rules known as constitution which guarantee people fundamental rights and raise their voice if their rights are violated.

More specifically, today anyone can deliver their thoughts which can be offensive to others. Delivering uncensored content to people may lead to violence and instability in the society. Therefore, there is a need for such a system that can ensure the suitability of some speech and identify if concerned person need to detained for some offensive or violence provoking speeches. Today, it become very important analyze and classify the public speeches to ensure stability and peace.

The task of determining if communication—such as text, audio, and so on—contains hatred or incites violence against an individual or a group of individuals is known as hate speech detection. Prejudice towards 'protected qualities', such as age, gender, sexual orientation, or race, is typically the basis for this.

1.1 Background

Over the years, research has suggested a number of methods for detecting hate speech; most recently, deep learning methods have been used. It is commonly established that deep learning classification models function and generalise successfully when trained on a vast, varied, and superior data set. Due to the lack of such a resource, earlier comparable efforts created datasets that were manually annotated and have tight size limitations.

This section outlines a number of well-known publicly labelled datasets that have been utilised by scholars to test and train hate speech detectors. To train the various hate classification algorithms featured in our trials, we take advantage of a large-scale corpus of created hate and non-hate sequences that extend these datasets.

This section also provides a general review of the methods currently used for detecting hate speech, along with a thorough explanation of the techniques we use to classify hate speech in our work.

We then employ this methodology to propose efficient and compact character-based hate detectors, and we analyse relevant research on character-based convolution neural networks.

1.2 Motivation for Advanced Machine Learning Techniques:

To get over the shortcomings of traditional methods, researchers are using state-of-the-art machine learning algorithms to identify hate speech. Massive volumes of social media data can be scanned by machine learning algorithms to find trends that might point to hate speech.

To ensure that the data is appropriately tokenized, encoded, and formatted to meet the requirements of LSTM, a good text representation takes into account the text's key features and semantic meaning. The model can learn contextual word and sentence representations as a result, making it easier to detect hate speech. The models can learn from this information more effectively and produce accurate predictions if the text data is meaningfully encoded, formatted, and represented. The incentives for using machine learning, such as the requirement for scalability, effectiveness, and objective detection, are examined in this subtopic. It highlights the potential of cutting-edge tools to gather semantic understanding and contextual information, which are essential for correctly recognising hate speech detection.

1.3 LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTM) methods are made to efficiently model and analyze sequential data, like time series or text data. The ability of LSTM networks to retain and use information over extended time periods, in contrast to standard feedforward neural networks, makes them particularly well-suited for duties including dependencies and the context preservation. The input data is typically converted into numbers depictions, such as word embeddings for written information or numerical properties for time series data, identical to other neural network architectures. The memory cell, that stores and processes data over time, is an essential part of LSTM networks. An input gate, forget gate, and output gate are the three fundamental components of a memory cell. The input gate selects which information from the input stream and the memory cell's previous state should be kept there.

It determines how much of each input element to retain by calculating an activation value that ranges from 0 to 1 for that element. The forget gate determines which information from the previous memory cell state is lost. It calculates an activation value for each component, ranging from zero to one, to determine how much of the previous memory cell state to forget. The input gate, forget gate, and current input are

used by the memory cell to update its state. It modifies the cell state by fusing the newly acquired data from the input gate with the data that was saved from the previous memory cell state. What data should be output from the current memory cell state is decided by the output gate. It determines how much of the information in the current memory cell state to reveal by computing an activation value for each element, ranging from 0 to 1. The result and the revised memory cell status from the previous phase are the inputs to the present step as the network using LSTM handles the sequential information one element at a time. The network can track relationships and background information over time thanks to this recurrence. LSTM networks are trained employing labeled data and an optimisation algorithm like Adam, just like other neural network models. By modifying the weights and biases of the network via back propagation, the goal is to minimize a loss function.

LSTM networks are commonly used in various applications, like language modelling, machine translation, sentiment analysis, and speech recognition, where understanding sequential data or making accurate predictions depend on modelling long-term dependencies.

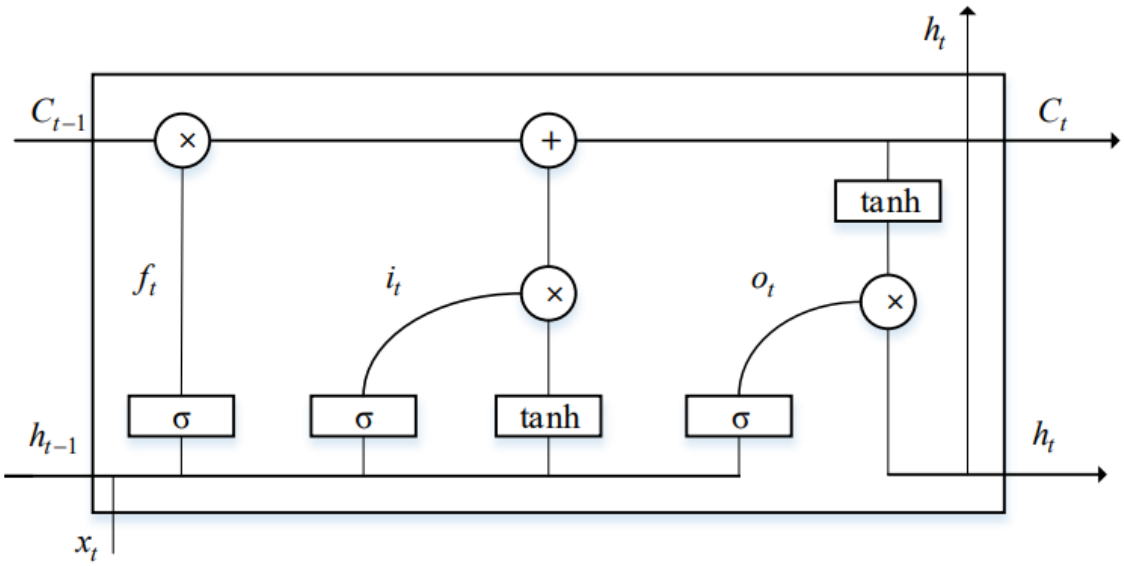

Figure 1. long-short Term Memory Model

The LSTM structure can be seen in Figure 3. The present moment's history data will be stored in the memory cell (Zt). The input-receiving gate (Inpt) regulates how much the input vector changes the data's current state in the memory cell. The forget gate (Fort) regulates how much the data in the current memory cell is influenced by previous knowledge from the past. Information output from a particular memory cell is managed by the output gate (Outt). Y = [y1, y2, y3 ……yT] is the input word vector matrix and xt is the dimension vector, The LSTM equations are:

$$Inp_t = \sigma(K_i [M_{t-1}, x_t] + q_i) \quad \text{.....(1)}$$

$$Out_t = \sigma(K_o [M_{t-1}, x_t] + q_o) \quad \text{.....(2)}$$

$$For_t = \sigma(K_f [M_{t-1}, x_t] + q_f) \quad \text{.....(3)}$$

$$Z_t = for_t * Z_{t-1} + Inp_t * tanM(K_c * [M_{t-1}, x_t] + q_c)$$

$$M_t = Out_t * tanM(Z_t) \quad \text{.....(4)}$$

Where M_t is the final outcome of the LSTM unit, σ(.) is the sigmoid activation, tanM(.) is a hyperbolic tangent function, K_i, K_o, and K_f are the weight matrices of the forget gate, output gate and input gate, respectively, and q_f, q_o, and q_i are the bias terms of the three control gates respectively. In conclusion, LSTM functions as an intermediary to gather data through memory cells and filter data according to the three gating units in order to get the results of hidden layers at the moment.

1.4 Bidirectional LSTM model

The choice of BiLSTM stems from its ability to understand the connections within sequences by examining data in both forward and backward directions. In simpler terms, it looks at the words that come before and after a particular word in a sentence to better grasp its meaning.

The BiLSTM architecture involves two unidirectional LSTMs that process the sequence in both directions. Imagine having two separate LSTM networks: one reads the sequence of tokens as is, while the other reads it in reverse order. Each of these LSTM networks provides a probability vector as output, and the final result is a combination of these two sets of probabilities.

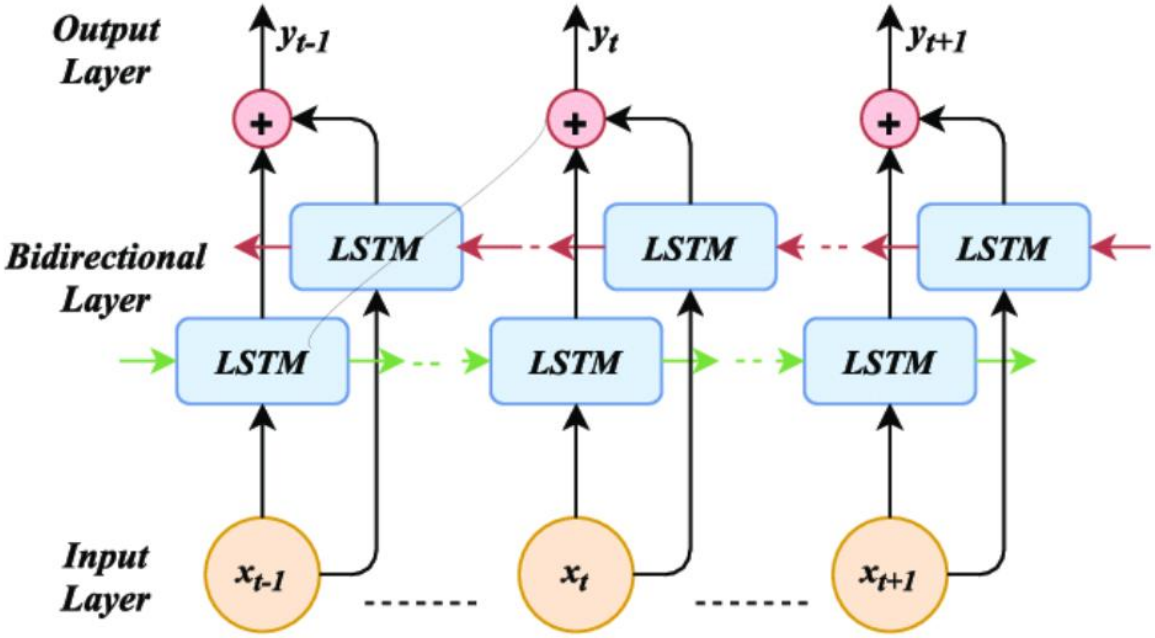

Figure 2: BiLSTM model

1.5 Bidirectional LSTM-SNP model

The BiLSTM-SNP model is a bidirectional LSTM-SNP model that captures two-way semantic dependence more effectively by taking into account both forward and backward information. BiLSTM-SNP is able to concatenate the hidden states of the two LSTM-SNPs as the representation of each position while simultaneously utilising forward and backward channels. The LSTM-SNPs for the forward and backward directions are expressed as follows:

$$\vec{u}_t, \vec{h}_t = \text{LSTM-SNP}\left(\vec{u}_{t-1}, \vec{h}_{t-1}, W\right)$$
$$\overleftarrow{u}_t, \overleftarrow{h}_t = \text{LSTM-SNP}\left(\overleftarrow{u}_{t+1}, \overleftarrow{h}_{t+1}, W\right)$$

To extract contextual semantic information about aspect words, BiLSTM-SNP is employed.

Regarding aspect word, the front and back sentiment features can be obtained by the BiLSTM-SNP. The three gates of BiLSTM-SNP are controlled by the hidden states, allowing for the efficient extraction of contextual semantic information related to aspect words.

The forward and backward LSTM-SNPs in the following BiLSTM-SNP figure. The forward LSTM-SNP's hidden states are

$$\left(\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_i\right)$$

, and the hidden states for backward LSTM-SNP are

$$\left(\overleftarrow{h}_n, \overleftarrow{h}_{n-1}, \ldots, \overleftarrow{h}_i\right)$$

The ultimate representation will be a concatenation of the hidden states of the forward and backward LSTM-SNP.

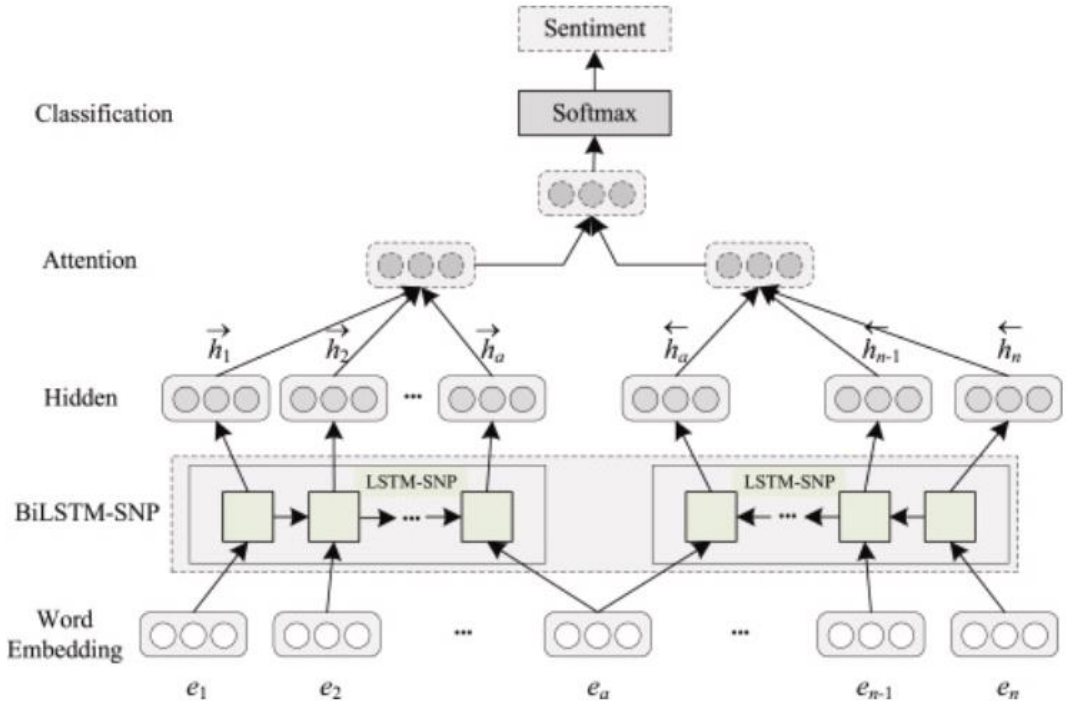

Figure 3: Bidirectional LSTM-SNP model

1.6 Dataset

This research investigates the identification of hate speech using a Twitter dataset. The text in the dataset is categorized into three groups: neither, hate speech, or offensive language. It's important to note that the dataset includes text that could be seen as racist, sexist, homophobic, or generally offensive, considering the nature of the study.

In simpler terms, the aim is to predict labels for the test dataset based on a training set of tweets and labels. A label '1' indicates the tweet is racist/sexist, while '0' indicates it is not.

2. Methodology

For the purpose of detecting hate speech, the offered code carries out a number of operations relating to data preprocessing, model training, and evaluation. Below is the outline figure of the work:

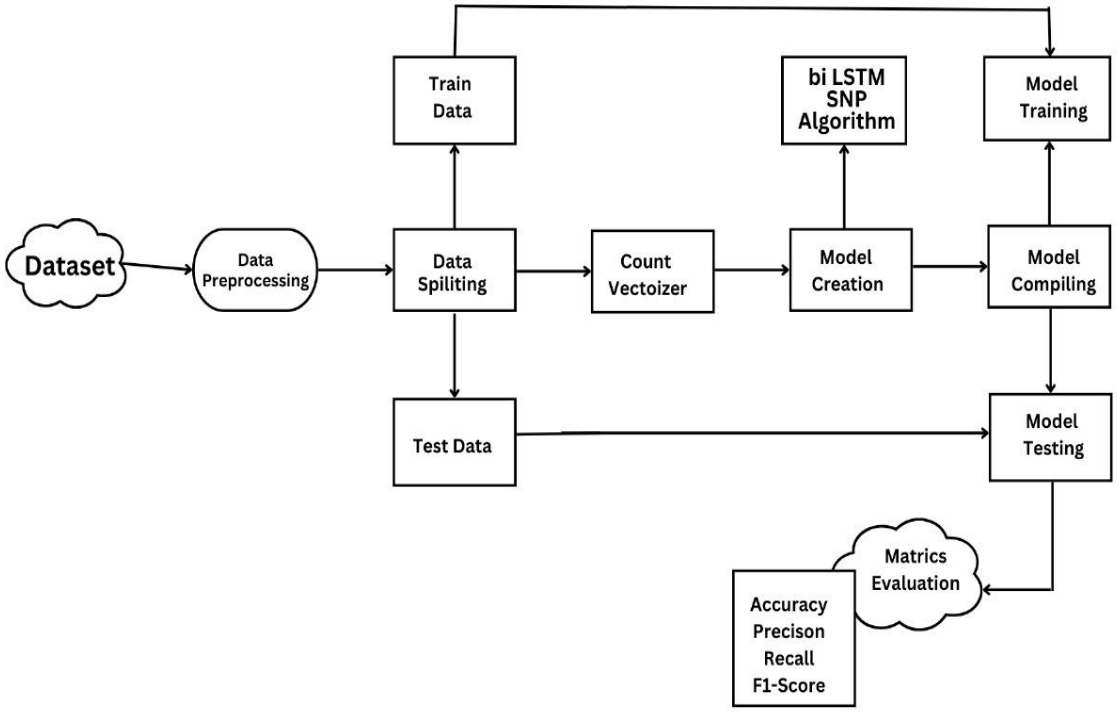

Figure 4. Block Diagram

2.1 Import Libraries:

We import a number of Python libraries in this section that are necessary for our machine learning as well as data analysis work. These libraries include the data manipulation tools NumPy and pandas, the natural language processing tools NLTK, the data visualisation tools Matplotlib and Seaborn, the machine learning tools Scikit-Learn, and the neural network model building tool Keras. These libraries provide us with the tools needed to handle data, preprocess text, construct models, and evaluate their performance.

```
import numpy as np
import pandas as pd
import os
import re
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import keras
from keras.models import Sequential
from keras.layers import LSTM, Embedding, SpatialDropout1D, Dense
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping, ModelCheckpoint
import pickle
```

Figure 6: Import libraries

2.2 Data collection:

An augmented or enriched dataset is one that was created by combining two other datasets. The process of combining columns from various databases is called data merging or joining. Based on a shared identifier or key column, rows from different datasets are merged. This process can be used when two datasets have something in common, such a common identifier or a set of features.

One can add new data or variables to the existing dataset by combining the columns of another dataset. We created our own datasets with repeated usernames in order to complete this task because there weren't any available.

In the data collection part, we load and preprocess two separate datasets. The first dataset, called 'df_twitter,' is related to Twitter sentiment analysis, while the second dataset, 'df_offensive,' deals with hate speech and offensive language. We remove unnecessary columns, transform class labels, and then merge these datasets into a single DataFrame named 'df.' This step ensures that we have a consolidated dataset for our analysis and model training.

```
df_twitter = pd.read_csv("/kaggle/input/twitter-sentiment-analysis-hatred-speech/train.csv")
df_twitter.drop('id', axis=1, inplace=True)
df_offensive = pd.read_csv("/kaggle/input/hate-speech-and-offensive-language-dataset/labeled_data.csv")
df_offensive.drop(['Unnamed: 0', 'count', 'hate_speech', 'offensive_language', 'neither'], axis=1, inplace=True)
df_offensive['class'].replace({0: 1, 2: 0}, inplace=True)
df_offensive.rename(columns={'class': 'label'}, inplace=True)
frames = [df_twitter, df_offensive]
df = pd.concat(frames)
```

Figure 6: Data Collection

2.3 Preprocessing

Preprocessing describes the set of procedures and techniques applied to input data before it is fed into a machine learning model for training or inference. Preprocessing is the process of preparing and altering data to increase the algorithm's functionality and effectiveness. Normally, preprocessing is done on both the input features (independent variables) and, if necessary, the target variable (dependent variable). Depending on the qualities of the data and the needs of the algorithm, several preprocessing procedures may be necessary.

Normalization and Tokenization: When all uppercase letters are changed to lowercase letters, this is known as Normalization. Tokenization is the division of tweets into individual tokens.

Stop-Word Removal: The stop words are not crucial phrases; they are eliminated from the tweets. Terms like a, an, of, from, etc. are frequently employed.

In this section, we perform data preprocessing to clean and prepare the text data for machine learning. We define a function called 'clean_text' that applies several text cleaning steps, including converting text to lowercase, removing URLs, HTML tags, punctuation, and stopwords, and applying stemming to reduce words to their root form. Afterward, the text data is successfully cleaned and standardised by applying this function to the 'tweet' column of DataFrame 'df'.

```
stemmer = nltk.SnowballStemmer("english")
stopword = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    text = [word for word in text.split(' ') if word not in stopword]
    text = " ".join(text)
    text = [stemmer.stem(word) for word in text.split(' ')]
    text = " ".join(text)
    return text

df['tweet'] = df['tweet'].apply(clean_text)
```

Figure 7: Preprocessing

2.4 Analyse Data:

Data analysis begins with visualizing the distribution of class labels in our combined dataset using Seaborn's countplot function. This helps us understand the balance between hate speech and non-hate speech samples in our data. Additionally, we print the shape of the DataFrame to get an overview of the number of rows and columns, providing insight into the dataset's size.

```
sns.countplot('label', data=df)
print(df.shape)
```

```
(56745, 2)
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

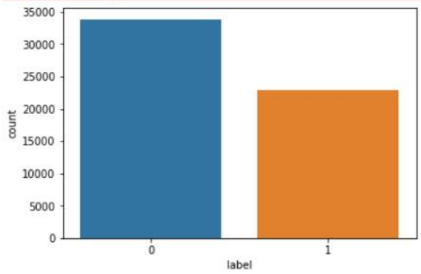

Figure 8: Analyse Data

2.5 Splitting the Data:

Data splitting is a crucial step for supervised machine learning. Here, we used Scikit-Learn's train_test_split function to divide our dataset as testing and training sets. The 'tweet' column is assigned to the 'x' variable, representing our input features, while the 'label' column is assigned to 'y,' representing our target labels. This random split ensures that we have separate data for model training and evaluation while maintaining data balance.

```
x = df['tweet']
y = df['label']
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=42)
```

Figure 9: Splitting Data

2.6 Adding Custom SNP layer

The Simple Neural Processor custom layer is designed to introduce a linear transformation within a neural network. In its initialization (_init_) method, the layer is configured with a parameter, output_dim, representing the desired dimensionality of the layer's output. The build method, executed when the layer is first used, establishes a trainable weight matrix, termed kernel. This matrix has dimensions determined by the input shape and output_dim, with the weights initialized uniformly. During the forward pass, the layer's call method executes the primary computation by calculating the dot product between the input tensor, denoted as x, and the learned weight matrix, kernel. This operation results in an output that reflects a linear transformation of the input data. Additionally, the layer includes a get_config method, which returns a dictionary containing the configuration details of the layer. This functionality is particularly useful when saving the overall model.

```
# Custom SNP Layer
class SimpleNeuralProcessor(Layer):
    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(SimpleNeuralProcessor, self).__init__(**kwargs)

    def build(self, input_shape):
        self.kernel = self.add_weight(name='kernel', shape=(input_shape[-1], self.output_dim), initializer='uniform', trainable=True)
        super(SimpleNeuralProcessor, self).build(input_shape)

    def call(self, x):
        output = K.dot(x, self.kernel)
        return output

    def get_config(self):
        config = super(SimpleNeuralProcessor, self).get_config()
        config.update({'output_dim': self.output_dim})
        return config
```

Figure 10: Adding Custom SNP layer

2.7 Building Model and Prediction:

Bidirectional Long Short-Term Memory (BiLSTM) model construction and compilation for binary classification is done in this code segment. Tokenizing the training data involves utilizing a tokenizer that has a maximum word count (max_words) set. Next, to achieve a uniform input size, the sequences are padded to a maximum length (max_len).

The architecture of the model consists of an embedding layer, a spatial dropout layer for regularization, a custom Simple Neural Processor layer for further processing, a Bidirectional LSTM layer for contextual information capture, and a dense layer with sigmoid activation for binary classification. Accuracy, RMSprop optimizer, and binary crossentropy loss are used in the model's compilation as the evaluation metrics.

A printed version of the model's architecture summary shows the layers, output shapes, and parameters. Furthermore, to improve training efficiency and generalization, EarlyStopping and ModelCheckpoint callbacks are defined to track validation accuracy and store the optimal model weights.

```
#Model Training and evaluation

# Define EarlyStopping and ModelCheckpoint callbacks
stop = EarlyStopping(
    monitor='val_accuracy',
    mode='max',
    patience=5
)

checkpoint = ModelCheckpoint(
    filepath='./',
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True
)

# Tokenize and pad test data
test_sequences = tokenizer.texts_to_sequences(x_test)
test_sequences_matrix = sequence.pad_sequences(test_sequences, maxlen=max_len)

# Train BiLSTM model
model.fit(sequences_matrix, y_train, batch_size=128, epochs=10, validation_split=0.2, callbacks=[stop, checkpoint])

# Evaluate BiLSTM model
accr = model.evaluate(test_sequences_matrix, y_test)
lstm_prediction = model.predict(test_sequences_matrix)
```

```
Epoch 1/10
266/266 [==============================] - 498s 2s/step - loss: 0.5765 - accuracy: 0.6871 - val_loss: 0.2584 - val_accuracy: 0.9232
Epoch 2/10
266/266 [==============================] - 492s 2s/step - loss: 0.2423 - accuracy: 0.9330 - val_loss: 0.2274 - val_accuracy: 0.9412
Epoch 3/10
266/266 [==============================] - 496s 2s/step - loss: 0.2027 - accuracy: 0.9453 - val_loss: 0.2084 - val_accuracy: 0.9421
Epoch 4/10
266/266 [==============================] - 493s 2s/step - loss: 0.1588 - accuracy: 0.9544 - val_loss: 0.2064 - val_accuracy: 0.9417
Epoch 5/10
266/266 [==============================] - 496s 2s/step - loss: 0.1514 - accuracy: 0.9574 - val_loss: 0.1764 - val_accuracy: 0.9453
Epoch 6/10
266/266 [==============================] - 497s 2s/step - loss: 0.1430 - accuracy: 0.9557 - val_loss: 0.1754 - val_accuracy: 0.9459
Epoch 7/10
266/266 [==============================] - 497s 2s/step - loss: 0.1299 - accuracy: 0.9615 - val_loss: 0.1996 - val_accuracy: 0.9305
Epoch 8/10
266/266 [==============================] - 499s 2s/step - loss: 0.1198 - accuracy: 0.9653 - val_loss: 0.1924 - val_accuracy: 0.9284
Epoch 9/10
266/266 [==============================] - 498s 2s/step - loss: 0.1311 - accuracy: 0.9628 - val_loss: 0.1864 - val_accuracy: 0.9328
Epoch 10/10
266/266 [==============================] - 496s 2s/step - loss: 0.1028 - accuracy: 0.9679 - val_loss: 0.1650 - val_accuracy: 0.9372
444/444 [==============================] - 39s 88ms/step - loss: 0.1796 - accuracy: 0.9345
```

Figure 11: Building Model

2.8 Evaluating the Result:

The performance of the model is assessed through key metrics such as precision, recall, F1 score, and overall accuracy, with a confusion matrix providing a comprehensive summary. Following the evaluation, the trained Bi-LSTM model and its associated text Tokenizer are preserved for future utilization. The model is stored in an HDF5 file, while the Tokenizer is serialized using the pickle module. The script also outlines the process for reloading the saved model and Tokenizer, emphasizing the potential incorporation of a custom neural processor named 'Simple Neural Processor' during the model loading phase. This thorough documentation and preservation process is crucial for ongoing analysis, deployment, or retraining of the model.

```
# Convert predictions to binary labels
res = np.array([1 if pred[0] >= 0.5 else 0 for pred in lstm_prediction], dtype=np.int32)

# Print confusion matrix for BiLSTM model
conf_matrix = confusion_matrix(y_test, res)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate precision, recall, and F1 score
precision = precision_score(y_test, res)
recall = recall_score(y_test, res)
f1 = f1_score(y_test, res)

# Print precision, recall, and F1 score
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

# Calculate and print accuracy
accuracy = (conf_matrix[0, 0] + conf_matrix[1, 1]) / np.sum(conf_matrix)
print(f"Accuracy: {accuracy}")

# Save Tokenizer and the BiLSTM model
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
model.save("hate_abusive_model_bilstm.h5")

# Load the trained BiLSTM model and tokenizer
load_model = keras.models.load_model("hate_abusive_model_bilstm.h5", custom_objects={'SimpleNeuralProcessor': SimpleNeuralProcessor})
with open('tokenizer.pickle', 'rb') as handle:
    load_tokenizer = pickle.load(handle)
```

```
Confusion Matrix:
[[7919  534]
 [ 408 5326]]
Precision: 0.9088737201365188
Recall: 0.9288454830833623
F1 Score: 0.9187510781438675
Accuracy: 0.9336011841827024
```

Figure 12: Evaluating the result

2.9 Test a New Sentence with the Loaded Model:

In this section, we demonstrate how to use the trained model to make predictions on new text data. We provide a sample sentence ('test_text') and preprocess it using the same 'clean_text' function as before. After tokenization and padding, we make a prediction using the loaded model and print the result. If the model's prediction is below 0.5, we classify it as 'No hate'; otherwise, we classify it as 'Hate and abusive.' This part illustrates the practical application of our trained model on real-world text inputs.

```
# Test a new sentence
test_text = 'I hate my country'

# Preprocess the test sentence
test_text = clean_text(test_text)
test_text = [test_text]
seq = load_tokenizer.texts_to_sequences(test_text)
padded = sequence.pad_sequences(seq, maxlen=max_len)

# Make a prediction with the loaded BiLSTM model
pred = load_model.predict(padded)

# Print the prediction
if pred[0][0] < 0.5:
    print("No hate")
else:
    print("Hate and abusive")
```

Figure 13: Test new Sentence

Output:

```
Hate and abusive
```

Figure 14: Output

3. Discussion

Comparing our analyses with machine learning algorithm.

Models	Accuracy
Proposed Model (BiLSTM SNP)	0.9336
Naive Bayes(unigram)	0.8521
SVM	0.4002
Decision Tree	0.8137
Word2vec model using LSTM	0.7815
GloVe Model using LSTM	0.829

Table 1: Analyses

The suggested model outperforms multiple benchmark models with a noteworthy accuracy of 93.36% in the research undertaken. With an accuracy of 85.21%, the Naïve Bayes (unigram) model performs admirably but lags behind the suggested model. Conversely, the SVM model exhibits a notable lag with an accuracy of 40.02%, indicating potential constraints in its predicting ability for the specified job. Comparing the Decision Tree model against the SVM, it shows its usefulness with an accuracy of 81.37%, which is rather good.

With LSTM, the Word2vec model achieves comparable accuracies of 78.15%, while the GloVe model achieves 82.9%. Although these models demonstrate the potential of using LSTM in conjunction with embedding approaches, the accuracy of the proposed model outperforms them.

4. Concluding Remarks

Our study concludes that the proposed BiLSTM SNP model performs exceptionally well, outperforming benchmark models such as Naive Bayes (85.21%), Decision Tree (81.37%), Word2vec using LSTM (78.15%), and GloVe using LSTM (82.9%). The model's accuracy of 93.36% is demonstrated. The SVM model's limitations in detecting hate speech are highlighted by the notable performance gap (40.02%) that was observed.

Our study tackles problems in hate speech detection research, such as dataset size, reliability, and linguistic diversity issues, in addition to model evaluation. Significant obstacles arise from the lack of agreement on definitions of hate speech and the complexity of the model architecture, highlighting the necessity of ongoing dataset improvement and expert participation.

We support the use of fine-grained feature set selection for both traditional and deep learning models, and draw attention to the lack of established best practices for comparing cross-dataset methods. In summary, our study not only offers a sophisticated model for detecting hate speech, but also highlights the critical issues that require the focus of the scientific community in order to advance this area of study.

In conclusion, this research work presents a robust approach to identifying hate speech and offensive language in text data. The biLSTM SNP model, trained on a combined dataset of Twitter sentiment and hate speech data, provides a valuable tool for automated content moderation and addressing online toxicity. The results obtained through this analysis can be used to enhance online platforms' content filtering and promote a safer online environment.

5. Future Work

There is a lot of scope in future to work on this. Cyberbullying detection is also possible through emoji and GIFs. Future project development may take a range of factors into account to improve the project. The model's performance may be further enhanced by additional research and optimization, including hyperparameter tuning. Using advanced deep learning architectures such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), the model is able to identify intricate patterns and connections in text data. The model might be improved by the addition of strategies like transformer models or attention processes.

References

1. Sekolah.mu, 2021. Deep Learning Techniques for Text Classification. Towards Data science.

2. Fatima Alkomah (2022). A Literature Review of Textual Hate Speech Detection Methods and Datasets. https://www.mdpi.com/2078-2489/13/6/273

3. Ali Toosi , 2015. Twitter Sentiment Analyses. https://www.kaggle.com/datasets/arkhoshghalb/twitter-sentiment-analysis-hatred-speech

4. Andrii Samoshyn,2019. Hate speech and offensive language dataset https://www.kaggle.com/datasets/mrmorj/hate-speech-and-offensive-language-dataset

5. Areei Al-Hassan ,2019.Detection of hate speech in social networks: A survey on multilingual corpus.6th International Conference on Computer Science and Information Technology.

6. Yanping Huang,Oian Liu,Hong Peng, Jun Wang ,Quin Yang(2023).Sentiment classification using bidirectional LSTM-SNP model and attention mechanism. Expeert System with Spplications. https://www.sciencedirect.com/science/article/pii/S0957417423002312?ref=pdf_download&fr=RR-2&rr=82c7fd970bc631e1#sec2

7. Mohamed Arbane, Rachid Benlarmi , Youcef Brik, Ayman Diyab Alahmr ,2023. Social media-based Covid-19 sentiment classification model using Bi-LSTM.Expert Systems with applications. https://www.sciencedirect.com/science/article/pii/S0957417422017353?ref=pdf_download&fr=RR-2&rr=82c7feee681731e1

8. Md Saraar Jahan, Mourad Ousallah (2021). A systematic review of Hate Speech automatic detection using Natural Language Processing.Neurocomputing https://www.sciencedirect.com/science/article/pii/S0925231223003557

9. Pitsilis, G.K., Ramampiaro, H., Langseth, H,2018. Effective hate-speech detection in Twitter data using recurrent neural networks. Appl. Intell. 4730–

4742.https://link.springer.com/article/10.1007/s10489-018-1242-y,https://arxiv.org/pdf/1801.04433.pdf

10. Malla, S., & PJA, A. (2021). COVID-19 outbreak: An ensemble pre-trained deep learning model for detecting informative tweets. Applied Soft Computing, 107, 107495. https://doi.org/10.1016/j.asoc.2021.107495

11. Ishani Priyadarshani, Sandipan Sahu Raghvendra Kumar (2023). A transfer Learning approach for detecting offensive and hate speech on social media platforms. Multimedia tools and applications. https://link.springer.com/article/10.1007/s11042-023-14481-3