PROJECT REPORT

TOXIC COMMENT CLASSIFICATION COMPARISON BETWEEN LSTM, BILSTM, GRU, AND BIGRU

JONATHAN KEVIN GIUSTINO
20.K1.0009

Faculty of Computer Science
Soegijapranata Catholic University
2024

APPROVAL AND RATIFICATION PAGE

Judul Tugas Akhir	:	TOXIC COMMENT CLASSIFICATION COMPARISON BETWEEN LSTM, BILSTM, GRU, AND BIGRU
Diajukan oleh	:	JONATHAN KEVIN GIUSTINO
NIM	:	20.K1.0009
Tanggal disetujui	:	08 Januari 2024
Telah setujui oleh		
Pembimbing	:	Yonathan Purbo Santosa S.Kom., M.Sc.
Penguji 1	:	Rosita Herawati S.T., M.I.T.
Penguji 2	:	Hironimus Leong S.Kom., M.Kom.
Penguji 3	:	R. Setiawan Aji Nugroho S.T., MCompIT., Ph.D
Penguji 4	:	Y.b. Dwi Setianto S.T., M.Cs.
Penguji 5	:	Dr. Yulianto Tejo Putranto S.T., M.T.
Penguji 6	:	Yonathan Purbo Santosa S.Kom., M.Sc.
Ketua Program Studi	:	Rosita Herawati S.T., M.I.T.
Dekan	:	Prof. Dr. F. Ridwan Sanjaya S.E., S.Kom., MS.IEC.

Halaman ini merupakan halaman yang sah dan dapat diverifikasi melalui alamat di bawah ini.

sintak.unika.ac.id/skripsi/verifikasi/?id=20.K1.0009

DECLARATION OF AUTHORSHIP

I, the undersigned:

Name : Jonathan Kevin Giustino

ID : 20.K1.0009

declare that this work, titled " TOXIC COMMENT CLASSIFICATION COMPARISON BETWEEN LSTM, BILSTM, GRU, AND BIGRU", and the work presented in it is my own. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at Soegijapranata Catholic University
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
3. Where I have consulted the published work of others, this is always clearly attributed.
4. Where I have quoted from the work of others, the source is always given.
5. Except for such quotations, this work is entirely my own work.
6. I have acknowledged all main sources of help.
7. Where the work is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Semarang, 8 January 2024

Jonathan Kevin Giustino

20.K1.0009

HALAMAN PERNYATAAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Yang bertanda tangan dibawah ini:

Nama : Jonathan Kevin Giustino

Program Studi : Teknik Informatika

Fakultas : Ilmu Komputer

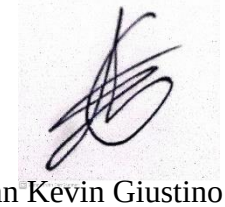
Jenis Karya : Skripsi

Menyetujui untuk memberikan kepada Universitas Katolik Soegijapranata Semarang Hak Bebas Royalti Nonekslusif atas karya ilmiah yang berjudul "TOXIC COMMENT CLASSIFICATION COMPARISON BETWEEN LSTM, BILSTM, GRU, AND BIGRU ". Dengan Hak Bebas Royalti Nonekslusif ini Universitas Katolik Soegijapranata berhak menyimpan, mengalihkan media/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan mempublikasikan tugas akhir ini selama tetap mencantumkan nama saya sebagai penulis / pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Semarang, 8 Januari 2024

Yang menyatakan



Jonathan Kevin Giustino

20.K1.0008

ACKNOWLEDGMENT

I have personally received a great number of supports, advice, and assistance throughout this document writing process. I would like to thank my supervisors Yonathan Purbo Santosa S.Kom., M.Sc. for helping me formulating this topic. I would also like to sincerely thank the other lecturers whom definitely left an indelible mark on my academic journey as well, though unnamed, they haven't missed my attention and am really grateful for their mentorship and guidance during my studies. I would also like to thank my friend for mental support to finish this document.

Dear precious family and friends thank you so much for the love, support and the priceless advice that has been my backbone throughout my stay in Soegijapranata Catholic University. You have been unwavering in your faith in me and always willing to provide a refuge for solace away from the demands of my thesis. You have been inspiring, and this achievement is as much yours as it is mine.

ABSTRACT

One of the biggest problems in modern internet age is toxicity. In this study, our aim is to draw an effective method to classify toxicity in text comment in this case specifically on Wikipedia comment. Old Regular Rule based model attacks, Machine Learning Methods suffer from rules based approaches and thus are incapable of accurate detection of toxicity while maintaining precision at the same time. To overcome this limitation, recurrent neural networks (RNNs), wherein, long short-term memory (LSTM) networks, gated recurrent unit(GRU) have been proposed. In this paper, author compares the LSTM, BiLSTM, GRU, and BiGRU for multi label classification for which is the best model to use towards jigsaw toxicity challenge dataset to classify toxicity. This study will be finding out which of the model is the best for classification and the difference between different type of pre-processing. We'll go ahead to use League of Legends tribunal datasets from kaggle as our base. The results obtained were that, the highest accuracy on the test was attained by BiLSTM model without cleaning with on 87.208% accuracy, 55.205% Precision, 68.540% Recall, and 60.623% F1-Score the result also shows while preprocessing on cleaning improve the resultant metrics by a marginal amount for regular LSTM, GRU, and BiGRU it doesn't always improve the result.

Keyword: Toxicity, LSTM, GRU, BiLSTM, BiGRU

TABLE OF CONTENTS

LIST OF FIGURE

LIST OF TABLE

CHAPTER 1
INTRODUCTION

1.1. Background

Impulse toxicity and abusive language of late are rising, as well as harmful behaviors that have negative consequences to people or the whole community. The forms of exhibiting toxicity may stem from hateful speech, cyberbullying, as well as harassment. Cyberbullying is a disruptive behavior defined as the intentional maliciousness towards others through computers, mobile smartphones, as well as other electronic gadgets in which others can be identified. Its victims have been proven to suffer sadness, stress, poor work and school performance and in severe cases, suicide ideation [1]. Guaranteeing the environment for online interaction remains welcome and safe calls for the ability to identify and properly eliminate toxic language.

Previous works have explored several approaches to toxicity detection that include rule-based models, machine or deep learning models. However, these frameworks intersect at mostly low accuracy, poor scalability, and being unable to extrapolate vital language expressions [2]. The elementary and rule-based systems cannot capture subtle meanings while machine/deep learning models require a large amount of pre-labeled datasets which is not always readily available especially depending on domain or languages used as well. With multiplayer game chat, or some social media sites, it is clearly demonstrated derivation of Spelling correction, language identification, and other common approaches in Natural Language Processing (NLP) were either irrelevant or underperformed that need to be provided. [2].

Recently, new studies show that RNNs and mostly LSTM networks can extract information from long range and complex sources at an increased efficiency. These models of deep learning are very effective in the manipulation of sequence data as inputs of text and even dependencies spanning over a long period between the elements of the data [3]. Word embedding technique can establishing semantic relations amongst words as the mainstay of natural language processing. It can be coupled with other forms of recurrent neural networks such as LSTM to analyze both the temporal features and semantic associations between data or sequence of text [4].

.

In this study, the author will use LSTM, Bidirectional LSTM, GRU, and Bidirectional GRU to classify toxicity and aid in its detecting and classifying toxicity in online socialization compared to previous rule-based models and other learning mechanisms. The classification result are in the form of Toxic, Severe Toxic, Obscene, Threat, Insult, and Identity Hate

1.2. Problem Formulation

1. Can the LSTM, Bidirectional LSTM, GRU, and BiGRU model successfully classify type of toxicity on the dataset?
2. How does different preprocessing method affect the performance of the model?
3. Which is the best model for multi label classification?

1.3. Scope

This project uses the jigsaw-toxic-comment dataset from kaggle. The train data containing 159571 unique comments will be split into 80% for training and 20% for validation. For testing we use test data provided that amounts to 6090 data.

1.4. Objective

This thesis aim to assess the LSTM and BiLSTM model's efficiency on our datasets and compare its performance with other RNN models such as GRU and BiGRU, and classifying what kind of toxic behavior is it.

CHAPTER 2
LITERATURE STUDY

Murnion, et al. [1] article goes in depth about data collected from online multiplayer game World of Tanks from Wargaming API. After that data is then processed and used for sentiment analysis with Microsoft azure cognitive service and twin word sentiment analysis, test results were surprisingly disappointing because of two possible reasons. The first and most plausible explanation is that the services may not have been utilized to their fullest extent because of a lack of knowledge with these technologies. The second reason is because in-game conversation is quite different from normal communication in terms of language structure and substance, with more brief code words being used than full-on sentences such as “gg” for good game, “wp” for well played or “typical lemming train” where bad strategy is applied by a team resulting in a low chance of winning the match.

Similar with Murnion's [1] result, research from past Marten, et al. [2] on the bases of Dota 2 text chat dataset from in game replays of 12923 matches, from matches in February 2012. The language used in this dataset is excessively elliptical, punctuated incorrectly, and hardly adheres to grammatical rules. As a result, common NLP techniques including language identification, spelling correction, and part-of-speech recognition were either ineffective or underwhelming. To counteract, Author used special tokenization based on simple white- space splitting. For this study the usage of capitalization in text chat to indicate emotional reaction, such as yelling, the case of the letters is left unaltered. Utilizing n-grams for distinguishing between toxic content and regular profanity. After that the author explains the correlation of winning/losing to the toxicity using SVM and TF-IDF and concludes the winning teams use less toxicity at the late stages of the match, The losing team, compared to the winning team at this point, shows a considerably stronger tendency to criticize their teammates performance once it becomes clear that they will win.

Dubey, et al. [3] focuses on why LSTM are effective for clasifying toxic comment detection dataset. LSTM provides adequate and appropriate results on testing data. LSTM networks are capable of retrieving significant data from intricate sources with excellent efficacy. These deep learning models are proficient at handling text-based sequence inputs and capturing temporal

dependencies within the data The developed model provides the proportion of toxic severity of the supplied sentence in addition to classifying a specific sentence as toxic or non-toxic. Overall, 94.49% for both accuracy and precision plus 92.79% recall, and of test data on the trained model were achieved using said model.

Esposito , et al [5] paper explain in detail many method to select the perfect threshold for the model that is trained on imbalanced data are prone to overpredict the majority class. For binary data, the classification threshold is set by default to 0.5 which in most case is not ideal for imbalanced data. The author found that their method can improve model performance in where the model is trained using imbalanced data

Ibrahim , et al [6] improved upon Georgakopoulos CNN-based algorithm that can only determine whether a comment is harmful. They ignore the issue of identifying the particular types of toxicity included in the statement. To address this issue in the dataset, a model using CNNs, bidirectional LSTMs, and bidirectional GRUs has been suggested. The ensemble model works better than previous techniques, according to evaluation findings, with F1 scores of 0.828 for toxic/non-toxic categorization and 0.872 for toxicity categories prediction, respectively.

Anand and Eswar [4] reach the similar consensus as Ibrahim [6] on their research based on public dataset available at Github. The dataset comprises five tags—toxic, threat, obscene, insult, and racism—for each Bengali social media remark, with LSTM and ANN performing best, in terms of training and testing, loss and accuracy. Following LSTM and ANN in terms of performance are CNN, GloVe & LSTM, and LSTM and ANN.

Using RNN, Mossie and Wang [7] their earlier study has been expanded and adjusted (Mossie & Wang, 2018). Their primary focus is on low-resource languages like Amharic, which may be used to identify the group that is most at risk. Instead, than concentrating on websites or blogs that already have a clear objective, the author selected Facebook pages that are susceptible to suspected hate speech. Word2Vec embedding and RNN-GRU demonstrated the best

performance from their findings of hate speech identification, with an AUC of 97.85% and an accuracy of 92.56%.

Obadimu, et al. [8] utilizes the Perspective API, a CNN-based classification tool, to rank the perceived toxicity of a particular remark on the Youtube comment dataset. This tool was built by Google's Project Jigsaw and Counter Abuse Technology teams. To assess whether a statement may be regarded as "toxic" to a conversation, this model employs a Convolutional Neural Network (CNN) trained with word-vector inputs. The author then used Latent Dirichlet Allocation (LDA) topic modeling to further analyze the semantic patterns within the collection of unstructured text bodies from these channels. This method made it possible to identify the collection of ethereal subjects that are present in the comments. After that, execute the topic modeling using the gensim Python module. Each comment underwent preprocessing, tokenization, and vectorization as if it were a separate document. The tokens' bigrams and trigrams were calculated using the WordNetLemmatizer, which also extracted word roots from the text.

Abassi et al [9] describe specific problem with many of these toxicity classification models is their sensitivity to frequently targeted identity groups, such as LGBT, Jews, Christian, Muslim , Blacks, and Whites. These statements are only considered harmful when interpreted in the correct context, like racial slurs. The resultant RNN model outperforms the other models on Kaggle datasets using Multilabel classification as a basis for their model. The RNN model's precision, recall, and F1-score are 96.84%, 96.72%, and 96.77%, respectively, while its accuracy is 94.49%.

Yazgili and Baykara [10] gather methods and dataset from many sources prior with Ibrahim [6] paper as a few examples. Author explained the type of toxicity that can occur and must be detected in the algorithm, few examples include flaming, harassment, and denigration. Yazgilli concluded that for toxicity detection, our main priority is not positive results of studies above mentioned, even though it is pleasing to look at. One suggestion for obtaining more effective results, is to analyze the words according to the purpose of use in the sentence, rather than the

word-based analysis in text analysis. In other words, it is the analysis of the general judgments of the sentences.

Mohamad [11] research is focused on exploring the effect of preprocessing when applied to state of the art model. Author demonstrated that the training of model with minimal transformation can still produce relatively decent model. Applying even basic transformations for preprocessing could in some cases lead to worse performance and should be applied with great concern.

CHAPTER 3
RESEARCH METHODOLOGY

3.1. Research Methodology

The research will follow flowchart, displayed on Figure 3.1. First, we prepare and preprocess our dataset. We need to make sure our dataset can be processed by both our GRU and LSTM. These results will be evaluated and analyzed, in search for conclusive result and creating a comprehensive report.

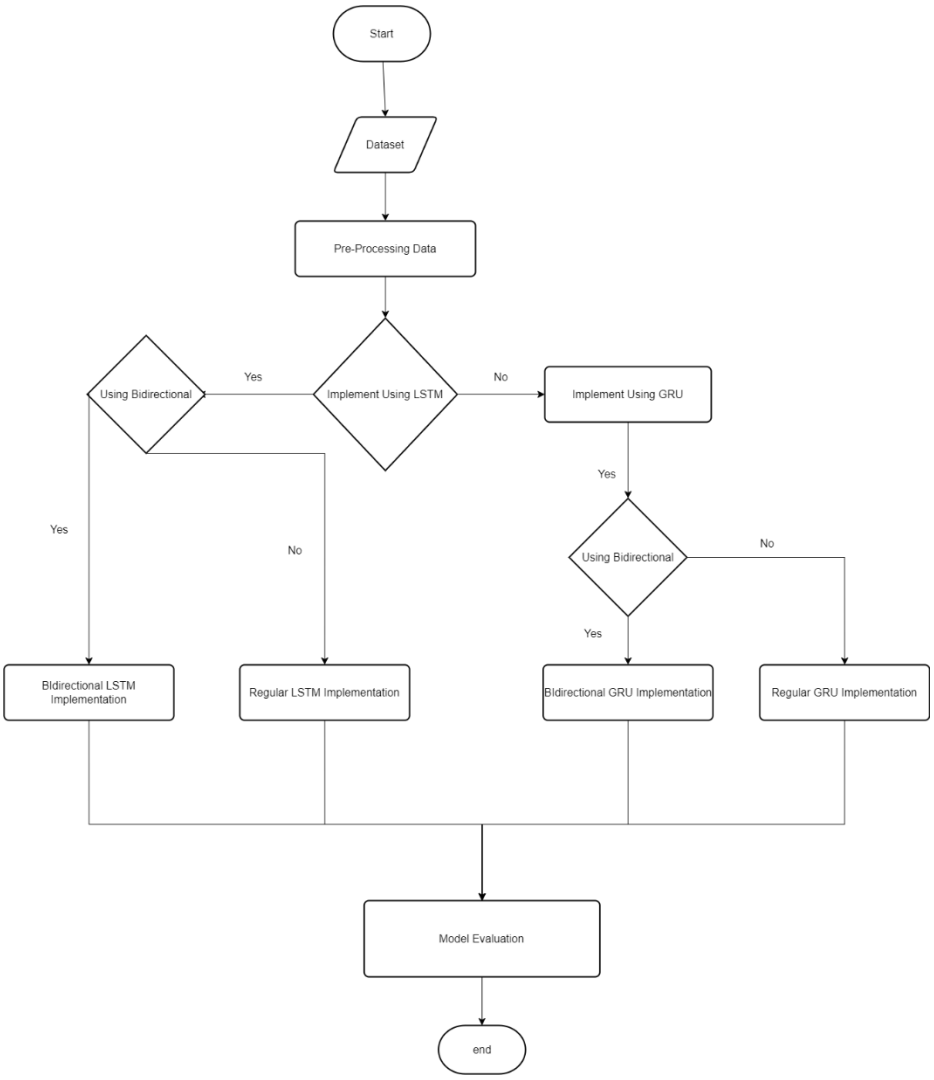


Figure 3.1 Research Methodology

3.2. Dataset Collection

The study will use jigsaw-toxic-comment-classification-challenge dataset available at kaggle [12]. This dataset consists of text chat of League of Legends reported cases before it went down. This dataset consist of 9370 data that include shown in Figure 3.2 :

- **train.csv** - the training set, contains comments with their assigned binary labels
- **test.csv** - the test set which will be used for our testing
- **sample_submission.csv** - a sample submission file to submit in kaggle competition
- **test_labels.csv** – real labels for the test data

Dataset contains 6 toxic behavior labeled by human raters. The types of toxicity are:

- toxic
- severe_toxic
- obscene
- threat
- insult
- identity_hate

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
id							
0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
0005c987bdfc9d4b	Hey... what is it..\n@ | talk .\nWhat is it.....	1	0	0	0	0	0
0007e25b2121310b	Bye! \n\nDon't look, come or think of comming ...	1	0	0	0	0	0
001810bf8c45bf5f	You are gay or antisemmitian? \n\nArchangel WH...	1	0	1	0	1	1
00190820581d90ce	FUCK YOUR FILTHY MOTHER IN THE ASS, DRY!	1	0	1	0	1	0

Figure 3.2 Example of train data

3.3. Data Pre-processing

We arranged pre-processing tasks with combination of Murnion et al method [1] and Marten [2] method. We use 3 different types of preprocessing on our train data that will be each implemented on our model are:

- **Cleaned** : Removed non aplhanumeric, unnecessary character, and stopword removal
- **Cleaned without using stopword removal :** Removed non aplhanumeric and unnecessary character

- **Not Cleaned :** Plain training dataset

After being pre-processed the train data will be splitted into 80% used for training data and 20% used for validation and optimizing purpose. Testing will use test.csv provided on the dataset

3.4. Model Development

Our multi-label classifier model will be implemented using deep learning frameworks, TensorFlow. Taking into account such crucial components as input dataset format, total layers augmented implementations, strong activation implementations and overall concordant harmonic reverberation is paramount. There will be 4 model created for this research. Model with LSTM, model created with Bidirectional LSTM, model created with GRU and model created with Bidirectional GRU. All model well be tuned by their hyper parameter, We'll tune out the epoch size, batch size, layer size, and threshold to find out the best result.

3.4.1. LSTM Algorithm

Long Short-Term Memory (LSTM) is one of RNN (Recurrent Neural Network) model designed to capture long-term dependencies in sequential data. LSTM have memory cells and three gates (input, forget, and output), this method is able to address vanishing gradient problem on regular RNN. LSTMs usually used on natural language processing task due to the ability to retain and selectively use information over extended sequences.

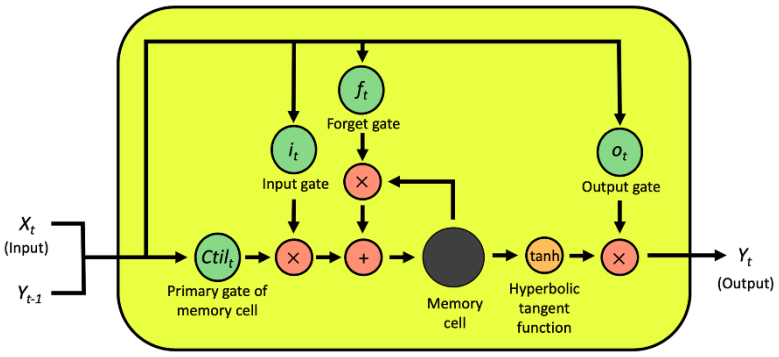


Figure 3.3 Summary of LSTM model

3.4.2. Bidirectional LSTM Algorithm

Bidirectional Long Short-Term Memory (BiLSTM) is a version of LSTM designed to capture information from two direction, past and future in sequential data hence the bidirectional term. Unlike regular LSTM, in BiLSTM the input flows in both directions and can utilize information from both sides. At each step of the sequence, information flows backward in time from both the past and future through their gated hidden states at every time step, thereby making the model more context-aware and thus considerably have better performance for regular LSTM in most case scenario with the tradeoff being slower time to train.

3.4.3. GRU Algorithm

Gated Recurrent Unit (GRU) is a type of RNN with similar architecture to LSTM and also designed to mitigate vanishing gradient problem in sequential data. Instead of LSTM three main gate, GRU simplifies the structure by only having two gates: reset gate and update gate, resulting in lighter. While being usually used for similar purpose which is natural language processing, GRU is computationally less intensive and more fitting for the tasks with lower data count compared with LSTMs.

3.4.4. Bidirectional GRU Algorithm

Bidirectional Gated Recurrent Unit (BiGRU) is a development of GRU which both collects information in two directions – the past and the future – with respect to the current time step in the input sequence. Just like BiLSTM, BiGRU allows capturing dependencies between backward and forward ones on each other. thereby making the model more context-aware and thus considerably have better performance compared with regular GRU with the tradeoff being slower time to train.

3.5. Model Evaluation and Analysis

In this research, the analysis will evaluate and compare the performance of LSTM, BiLSTM, GRU, and BiGRU. The author will record the result of accuracy, precision, recall and f1-score of each model and the effect of preprocessing applied.

1. **Accuracy** : Percentage of correct predictions for the test data. calculated by dividing the number of correct predictions by the number of total predictions.

2. **Precision** : Percentage of **positive true predictions** compared to overall positive prediction results.
3. **Recall** : Percentage of **positive true predictions** compared to overall positive true results.
4. **F1-score** : F-score is weighted comparison of average precision and recall percentage

After all results have been recorded comparing the result between each model, the author will be made to be able to see the comparative value between the four algorithm models. By comparing all algorithm models, we can understand which model is better and more efficient in classifying toxicity.

CHAPTER 4

IMPLEMENTATION

4.1. Experiment Setup

To complete this project, author used a laptop with Intel i5-9300H CPU and 16 GB of available RAM. The author uses Python as a programming language. To run the program, author uses Kaggle notebook with NVIDIA P100 accelerator to accelerate learning process and uses Python programming language.

4.2. Pre-processing

```
1. import pandas as pd
2. import re
3. import matplotlib.pyplot as plt
4. from tensorflow.keras import utils
5. from tensorflow.keras.models import Sequential
6. from tensorflow.keras.layers import Dense, Embedding, MaxPooling1D,
   Dropout, GRU, LSTM, Bidirectional, SpatialDropout1D, TextVectorization
7. from tensorflow.keras.callbacks import ModelCheckpoint
8. from tensorflow.keras.models import load_model
9. from sklearn.model_selection import train_test_split
10.    from sklearn.metrics import classification_report
11.    from sklearn.metrics import f1_score, precision_score, recall_score,
   accuracy_score
```

First, we need to import some important prerequisite to run the code, Line 1 is used to import pandas library that will be used to read the CSV file that contains text comments and labels used for training and testing. Line 2 is used to import re used to replace character at preprocessing. Line 3 is matplotlib used for plotting image to show the result through the training. Line 4-9 is used to import necessary the proposed model of LSTM, BiLSTM, GRU, and BiGRU. Line 9 is used to split the train data. Line 10 and 11 is used for evaluation purpose later.

4.2.1. Preprocessing Train Data

```
12.    import nltk
13.    from nltk.corpus import stopwords
14.    from nltk.tokenize import word_tokenize
15.    nltk.download('stopwords')
16.
17.    def remove_english_stopwords(text):
18.       stop_words = set(stopwords.words('english'))
19.       word_tokens = word_tokenize(text)
```

```
20.       filtered_text = [word for word in word_tokens if not word in
   stop_words]
21.       return ' '.join(filtered_text)
22.
23.    def remove_unnecessary_char(text):
24.        text = re.sub('\n',' ',text) # Remove every '\n'
25.        text                                                          =
   re.sub('((www\.[^\s]+)|(https?://[^\s]+)|(http?://[^\s]+))',' ',text)  #
   Remove every URL
26.        text = re.sub('  +', ' ', text) # Remove extra spaces
27.        return text
28.
29.    def remove_nonaplhanumeric(text):
30.        text = re.sub('[^0-9a-zA-Z]+', ' ', text)
31.        return text
32.
33.    def preprocess(text):
34.        text = remove_nonaplhanumeric(text)
35.        text = remove_unnecessary_char(text)
36.        text = remove_english_stopwords(text)
37.        return text
```

In this project the author used three level of preprocessing cleaned with stopword, cleaned without stopwords, and not cleaned. Line 12 -14 is used for importing nltk used for preprocessing the dataset. Line 17 -21 is used to remove English stopwords, Line 23-27 is used to remove unnecessary character that include /n, URL links, and extra space. Line 29-31 is used to remove English stopwords. Line 33-37 is used for implement preprocessing, three level of preprocessing explained again here:

- **Cleaned** : Removed non aplhanumeric, unnecessary character, and stopword removal
- **Cleaned without using stopword removal :** Removed non aplhanumeric and unnecessary character
- **Not Cleaned :** Plain training dataset

```
38.    train_data     =     pd.read_csv('/kaggle/input/jigsaw-toxic-comment-
   classification-challenge/train.csv',index_col='id')
39.    train_data['comment_text']                                        =
   train_data['comment_text'].apply(preprocess)
40.    train_data['comment_length']                                      =
   train_data['comment_text'].apply(lambda row: len(row))
41.    train_data.head()
```

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate	comment_length
id								
0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0	44
0005c987bdfc9d4b	Hey talk What exclusive group WP TALIBANS good...	1	0	0	0	0	0	208
0007e25b2121310b	Bye Don look come think comming back Tosser	1	0	0	0	0	0	43
001810bf8c45bf5f	You gay antisemmitian Archangel WHite Tiger Me...	1	0	1	0	1	1	425
00190820581d90ce	FUCK YOUR FILTHY MOTHER IN THE ASS DRY	1	0	1	0	1	0	38

Figure 4.1 Example of train data after adding comment length

After creating function to preprocess our data we import train data using code from line 38 and preprocessing the data using code line 39, we add 1 more row filled with the length of the comment. After adding comment length, we can check the length of toxic comments to use in our model using code used By using code on line 42-49 , we can search the maximum comment length used for embedding in our model. Different preprocessing method yield different result.

```
42.    toxic_data = train_data[train_data['toxic'] == 1]
43.    print(
44.        'max', toxic_data['comment_length'].max(),
45.        'min', toxic_data['comment_length'].min(),
46.        'mean', toxic_data['comment_length'].mean(),
47.        'median', toxic_data['comment_length'].median(),
48.        '75%', toxic_data['comment_length'].quantile(0.75),
49.    )
```

And the resulting data from the code from line 42-49 is shown below.

1. Cleaned data : **max 5000 min 4 mean 209.4926114816268 median 84.0 75% 181.0**
2. Cleaned without stopwords : **max 5000 min 8 mean 280.898260755852 median 118.0 75% 260.0**
3. Not cleaned : **max 5000 min 8 mean 295.24604420034 median 123.0 75% 271.0**

With this result, the author decided to use 300 as maximum length for embedding that will be later explained in Model creation.

```
50.    x_train1 = train_data['comment_text'].values
51.    y_train1 = train_data[['toxic', 'severe_toxic', 'obscene', 'threat',
   'insult', 'identity_hate']].values
52.    x_train,x_val,y_train,y_val=train_test_split(x_train1,y_train1,train
   _size=0.8,shuffle=True,random_state=42)
```

Line 52 of this code is used to split the data into 80 % training and 20% validation.

4.2.2. Preprocessing Test Data

Dataset used by author provide 2 types of test data, test.csv contains id and comments used for testing purpose and test_labels.csv. Code and picture provided few examples of test data Test data contains -1 value that is used to deter hand labelling, for our purpose we don't need the -1 value so the author removed it and combine the test data into one table using code from line 57-59. After that we split the test data into x_test and y_test, x_test will be used as the input of the model and y_test is filled with ground truth the result of the code is shown at figure 4.2 and figure 4.3.

```
53.    test_data = pd.read_csv('/kaggle/input/jigsaw-toxic-comment-
   classification-challenge/test.csv',index_col='id')
54.    test_labels_data = pd.read_csv('/kaggle/input/jigsaw-toxic-comment-
   classification-challenge/test_labels.csv',index_col='id')
55.    test_labels.head()
56.    test_labels_data.head()
```

id	comment_text
00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...
0000247867823ef7	== From RfC == \n\n The title is fine as it is...
00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...
00017563c3f7919a	:If you have a look back at the source, the in...
00017695ad8997eb	I don't anonymously edit articles at all.

id	toxic	severe_toxic	obscene	threat	insult	identity_hate
00001cee341fdb12	-1	-1	-1	-1	-1	-1
0000247867823ef7	-1	-1	-1	-1	-1	-1
00013b17ad220c46	-1	-1	-1	-1	-1	-1
00017563c3f7919a	-1	-1	-1	-1	-1	-1
00017695ad8997eb	-1	-1	-1	-1	-1	-1

Figure 4.2 Example of test.csv and test_labels.csv

```
57.    test_data = test_data.join(test_labels_data)
58.    test_data = test_data[test_data['toxic'] != -1]
59.    test_data.head()
```

id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0001ea8717f6de06	Thank you for understanding. I think very high...	0	0	0	0	0	0
000247e83dcc1211	:Dear god this site is horrible.	0	0	0	0	0	0
0002f87b16116a7f	":::: Somebody will invariably try to add Relig...	0	0	0	0	0	0
0003e1cccfd5a40a	" \n\n It says it right there that it IS a typ...	0	0	0	0	0	0
00059ace3e3e9a53	" \n\n == Before adding a new product to the l...	0	0	0	0	0	0

Figure 4.3 Test_data after combining two table

```
60.    x_test = test_data['comment_text'].values
61.    y_test = test_data[['toxic', 'severe_toxic', 'obscene', 'threat',
   'insult', 'identity_hate']].values
```

4.3. Creating The Model

The author creates 4 different model LSTM, BiLSTM, GRU, and BiGRU for multilabel classification purpose, all model will be treated the same at preprocessing, threshold searching and evaluation. before doing that there is part of the code that is used by all the model shown below. The author used max comment length of 300 and max vocabulary size to 10000 . encoder is this is first layer of network that convert coming messages into list of numbers. TextVectorization is a layer which maps text features to integer sequences. encoder adapt to essentially building vocabulary based on the comments on training data. This is shown at code line 62-65

```
62.    max_comment_len = 300
63.    num_words = 10000
64.    encoder = TextVectorization(max_tokens=num_words)
65.    encoder.adapt(train_data['comment_text'].values)
```

4.3.1. LSTM Model

First LSTM model used for this project is coded on line 66-78. LSTM model above contains encoder shown at line 67 to convert message to list of number, embedding layer shown at line 68-73 turns positive integers (indexes) into dense vectors of fixed size. Dense vectors this is modern kind of representing words as vector in a fixed size to capture semantic relationship with our input dim the same as vocabulary length, output dim of 128, mask_zero set to true so the model should treat the value 0 as a special "padding" value, and input length of 300.

```
66.    model_lstm = Sequential([
67.        encoder,
68.        Embedding(
69.            input_dim=len(encoder.get_vocabulary()),
70.            output_dim=128,
71.            mask_zero=True,
72.            input_length=max_comment_len,
73.        ),
74.        SpatialDropout1D(0.5),
75.        LSTM(40, return_sequences=True),
76.        LSTM(40),
77.        Dense(6, activation='sigmoid'),
78.    ])
```

SpatialDropout is used to reduce overfitting by dropping some neuron during training. For the LSTM layer the author used 2 layers of LSTM with 40 units. The first LSTM layer captures sequential information, and the second one refines this information for downstream processing, and finally Dense output layer with 6 units corresponds to six label on the dataset ('toxic',

'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate') and sigmoid activation for multi-label classification task

```
79.    model_lstm.compile(optimizer='adam', loss='binary_crossentropy',
   metrics=['accuracy', 'AUC'])
80.
81.    model_lstm_save_path = '/kaggle/working/toxic_model_lstm'
82.    checkpoint_callback_lstm = ModelCheckpoint(
83.        model_lstm_save_path,
84.        monitor='val_accuracy',
85.        save_best_only=True,
86.        verbose=1,
87.        save_format='tf',
88.    )
```

After that we compile the model using line 79 of the code, using adam optimizer and binary_crossentropy loss function. The author added save path for the model to save the model with the best val_accuracy on line 81-88.

```
89.    history_lstm = model_lstm.fit(
90.        x_train,
91.        y_train,
92.        validation_data=(x_val,y_val),
93.        epochs=5,
94.        batch_size=512,
95.        callbacks=[checkpoint_callback_lstm],
96.    )
97.
98.    plt.plot(history_lstm.history['accuracy'],
99.             label='Training accuracy')
100.   plt.plot(history_lstm.history['val_accuracy'],
101.            label='Validation accuracy')
102.   plt.xlabel('Epochs')
103.   plt.ylabel('Accuracy')
104.   plt.legend()
105.   plt.show()
106.   plt.plot(history_lstm.history['loss'],
107.            label='Training loss')
108.   plt.plot(history_lstm.history['val_loss'],
109.            label='Validation loss')
110.   plt.xlabel('Epochs')
111.   plt.ylabel('Loss')
112.   plt.legend()
113.   plt.show()
114.
115.   saved_model = load_model(model_lstm_save_path)
116.   saved_model.evaluate(x_val, y_val, verbose=1)
```

After that we fit the model with batch size 512 and epoch of 5 (line 89-96), and by using code from line 98-113 we can plot the graph for learning process. Using code from line 115-116 we load the best model and evaluate it using validation data.

With line 115-116 we can evaluate early how our model will perform by evaluating it on validation data.

4.3.2. BiLSTM Model

For BiLSTM model the author used the same encoder and embedding technique on a new Kaggle notebook. The difference can be seen in line 143-144, the model still has 40 units of LSTM model which are now Bidirectional.

```
117.  model_bidirectional = Sequential([
118.      encoder,
119.      Embedding(
120.          input_dim=len(encoder.get_vocabulary()),
121.          output_dim=128,
122.          mask_zero=True,
123.          input_length=max_comment_len,
124.      ),
125.      SpatialDropout1D(0.5),
126.      Bidirectional(LSTM(40, return_sequences=True)),
127.      Bidirectional(LSTM(40)),
128.      Dense(6, activation='sigmoid'),
129.  ])
```

And just like regular LSTM model, this BiLSTM model have two layer captures sequential information, and the second one refines this information for downstream processing, and finally Dense output layer with 6 units corresponds to six label on the dataset ('toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate') and sigmoid activation for the same goal of multi-label classification task

```
130.  model_bidirectional.compile(optimizer='adam',
   loss='binary_crossentropy', metrics=['accuracy', 'AUC'])
131.
132.  model_bidir_save_path = '/kaggle/working/toxic_model_bidir'
133.  checkpoint_callback_lstm = ModelCheckpoint(
134.      model_bidir_save_path,
135.      monitor='val_accuracy',
136.      save_best_only=True,
137.      verbose=1,
138.      save_format='tf',
139.  )
```

BiLSTM model is compiled with adam optimizer and using binary_crossentropy loss function, on line 132-139 we have save path and checkpoint to save the model with the best val_accuracy

```
140.  history_bidir = model_bidirectional.fit(
```

```
141.      x_train,
142.      y_train,
143.      validation_data=(x_val,y_val),
144.      epochs=5,
145.      batch_size=512,
146.      callbacks=[checkpoint_callback_bidir],
147.  )
148.
149.  plt.plot(history_bidir.history['accuracy'],
150.           label='Training accuracy')
151.  plt.plot(history_bidir.history['val_accuracy'],
152.           label='Validation accuracy')
153.  plt.xlabel('Epochs')
154.  plt.ylabel('Accuracy')
155.  plt.legend()
156.  plt.show()
157.  plt.plot(history_bidir.history['loss'],
158.           label='Training loss')
159.  plt.plot(history_bidir.history['val_loss'],
160.           label='Validation loss')
161.  plt.xlabel('Epochs')
162.  plt.ylabel('Loss')
163.  plt.legend()
164.  plt.show()
165.
166.  saved_model = load_model(model_lstm_save_path)
167.  saved_model.evaluate(x_val, y_val, verbose=1)
```

After that we fit the model with batch size of 512 and epoch of 5. and by using code from line 170-181 to show training graph and early evaluation using validation data.

4.3.3. GRU Model

For the GRU model we still use encoder and embedding technique on a new Kaggle notebook. Main difference can be seen in line 177 and line 179, now the model uses GRU layer instead of LSTMs.

```
168.  model_gru = Sequential([
169.      encoder,
170.      Embedding(
171.          input_dim=len(encoder.get_vocabulary()),
172.          output_dim=128,
173.          mask_zero=True,
174.          input_length=max_comment_len,
175.      ),
176.      SpatialDropout1D(0.5),
177.      GRU(20, return_sequences=True),
178.      BatchNormalization(),
179.      GRU(20),
180.      Dense(6, activation='sigmoid'),
181.  ])
```

In line 169-175 we have encoder and embedding layer just like prior model, the model also uses spatial dropout shown at line 176 for reducing overfitting. The difference can be found in line 177-179. Here the author uses GRU layer with 20 unit that return full sequence, then the data goes to batch normalization layer and after that goes into another GRU layer with 20 unit that returns only the final hidden state before going into dense output layer output layer with 6 units corresponds to six label on the dataset ('toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate') and sigmoid activation for multi-label classification task.

```
182.  model_gru_save_path = '/kaggle/working/toxic_model_gru'
183.  checkpoint_callback_gru = ModelCheckpoint(
184.      model_gru_save_path,
185.      monitor='val_accuracy',
186.      save_best_only=True,
187.      verbose=1,
188.      save_format='tf',
189.  )
190.
191.  model_gru.compile(optimizer='adam', loss='binary_crossentropy',
     metrics=['accuracy', 'AUC'])
192.
193.  history_gru = model_gru.fit(
194.      x_train,
195.      y_train,
196.      epochs=5,
197.      batch_size=512,
198.      validation_data=(x_val,y_val),
199.      callbacks=[checkpoint_callback_gru],
200.  )
```

The GRU model then compiled and fitted with adam optimizer, binary_crossentropy loss function and trained with 5 epoch and 512 batch size. And the author also uses model save path at line 196 to save the model with the highest val_accuracy value.

```
201.  plt.plot(history_gru.history['accuracy'],
202.           label='Training accuracy')
203.  plt.plot(history_gru.history['val_accuracy'],
204.           label='Validation accuracy')
205.  plt.xlabel('Epochs')
206.  plt.ylabel('Accuracy')
207.  plt.legend()
208.  plt.show()
209.  plt.plot(history_gru.history['loss'],
210.           label='Training loss')
211.  plt.plot(history_gru.history['val_loss'],
212.           label='Validation loss')
213.  plt.xlabel('Epochs')
214.  plt.ylabel('Loss')
215.  plt.legend()
216.  plt.show()
217.  saved_model = load_model(model_gru_save_path)
```

```
saved_model.evaluate(x_val, y_val, verbose=1)
```

after training, we use matplotlib show training graph and loss graph of our model using code from line 201-218.

4.3.4. Bidirectional GRU Model

Here is the code used for the BiGRU model using the same encoder and embedding technique on a new Kaggle notebook. Structurally similar with regular GRU model with the main difference shown at line 228 and 230 that show GRU model is now bidirectional

```
model_bi_gru = Sequential([
    encoder,
    Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=128,
        mask_zero=True,
        input_length=max_comment_len,
    ),
    SpatialDropout1D(0.5),
    Bidirectional(GRU(20, return_sequences=True)),
    BatchNormalization(),
    Bidirectional(GRU(20)),
    Dense(6, activation='sigmoid'),
])
```

In line 220-226 the model still uses the same encoder and embedding layer just like prior GRU model, the model also uses spatial dropout for reducing overfitting. The difference can be found in line 228-230. Here the Bidirectional GRU layer with 20 unit that return full sequence, then the data goes to batch normalization layer and after that goes into another Bidirectional GRU layer with 20 unit that returns only the final hidden state before going into dense output layer that amounts to 6 units corresponds to six label on the dataset ('toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate') and sigmoid activation for multi-label classification task

```
model_bi_gru_save_path = '/kaggle/working/toxic_model_bi_gru'
checkpoint_callback_gru = ModelCheckpoint(
    model_bi_gru_save_path,
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1,
    save_format='tf',
)

model_bi_gru.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy', 'AUC'])

history_bi_gru = model_bi_gru.fit(
```

```
245.        x_train,
246.        y_train,
247.        epochs=5,
248.        batch_size=512,
249.        validation_data=(x_val,y_val),
250.        callbacks=[checkpoint_callback_gru],
251.    )
```

The BiGRU model then compiled and fitted with adam optimizer, binary_crossentropy loss function and trained with 5 epoch and 512 batch size. And the author also uses model save path at line 233 to save the model with the highest val_accuracy value.

```
252.    plt.plot(history_bi_gru.history['accuracy'],
253.             label='Training accuracy')
254.    plt.plot(history_bi_gru.history['val_accuracy'],
255.             label='Validation accuracy')
256.    plt.xlabel('Epochs')
257.    plt.ylabel('Accuracy')
258.    plt.legend()
259.    plt.show()
260.    plt.plot(history_bi_gru.history['loss'],
261.             label='Training loss')
262.    plt.plot(history_bi_gru.history['val_loss'],
263.             label='Validation loss')
264.    plt.xlabel('Epochs')
265.    plt.ylabel('Loss')
266.    plt.legend()
267.    plt.show()
268.    saved_model = load_model(model_bi_gru_save_path)
269.    saved_model.evaluate(x_val, y_val, verbose=1)
```

after training, we use matplotlib show training graph and loss graph of this bidirectional model shown at line 266-281. Model is also shown early evaluation using code shown in line 282-289

4.3.5. Finding Optimal Threshold

At this point, all model can be used at real data and with the resulting output in between 0 and 1. With the example code from line 270-272 and will yield result seen below,

```
270.    test_labels =
        saved_model.predict(test_data[test_data['toxic']==1][:5]['comment_text'
        ].values)
271.    for labels in test_labels:
272.        print([ round(lbl, 2) for lbl in labels])
```

```
1/1 [==============================] - 3s 3s/step
[0.45, 0.01, 0.09, 0.0, 0.06, 0.02]
[0.96, 0.13, 0.85, 0.04, 0.68, 0.11]
[0.97, 0.42, 0.93, 0.11, 0.83, 0.28]
[0.23, 0.0, 0.02, 0.0, 0.03, 0.0]
```

```
[0.95, 0.07, 0.79, 0.02, 0.6, 0.08]
```

However, the author planned to make true or false prediction for each label instead of sigmoid number between 0 and 1 so we need to find threshold [5], to find threshold author used validation dataset and threshold from 0,1 to 0,9 with 0,1 increment and will be used for all model. With the code shown below

```
273. from sklearn.metrics import classification_report, f1_score,
   precision_score, recall_score, accuracy_score
274.
275. thresholds = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
276.
277. predicts_val = saved_model.predict(x_val, verbose=1)
278.
279. for threshold in thresholds:
280.   y_pred_val = (predicts_val > threshold)
281.   clr = classification_report(y_val, y_pred_val)
282.   accuracy = accuracy_score(y_val, y_pred_val)
283.   f1 = f1_score(y_val, y_pred_val, average='weighted')
284.   precision = precision_score(y_val, y_pred_val, average='weighted')
285.   recall = recall_score(y_val, y_pred_val, average='weighted')
286.   print(f"Threshold: {threshold}\nClassification Report:\n----------
   ------------\n{clr}\nF1 Score: {f1}\nPrecision: {precision}\nRecall:
   {recall}\n\nAccuracy: {accuracy}\n")
```

On line 273 we import library necessary for our evaluation including precision, recall, and f1 score. On line 275-286 we use looping to find the threshold using validation data. After finding the optimal threshold for each of our model we can find the threshold to use on or test dataset

```
287. thresholds = [] # selected based on validation threshold
288.
289. predicts = saved_model.predict(x_test, verbose=1)
290.
291. y_pred = (predicts > threshold)
292.
293. f1 = f1_score(y_test, y_pred, average='weighted')
294. accuracy = accuracy_score(y_test, y_pred)
295. clr = classification_report(y_test, y_pred)
296. precision = precision_score(y_test, y_pred, average='weighted')
297. recall = recall_score(y_test, y_pred, average='weighted')
298. print(f"Threshold: {threshold}\nClassification Report:\n----------
   ----------\n{clr}\nF1 Score: {f1}\nPrecision: {precision}\nRecall:
   {recall}\n\nAccuracy: {accuracy}\n")
```

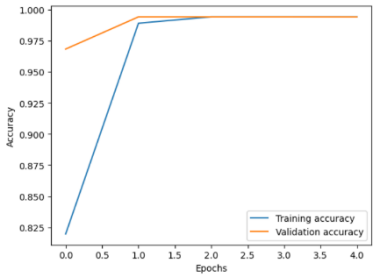
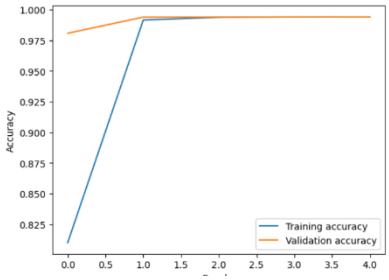
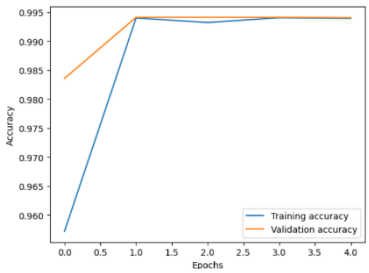
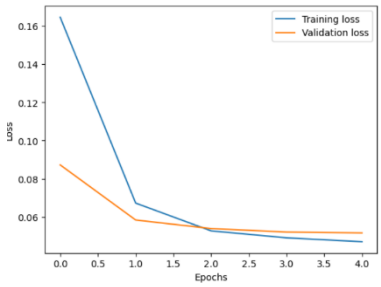
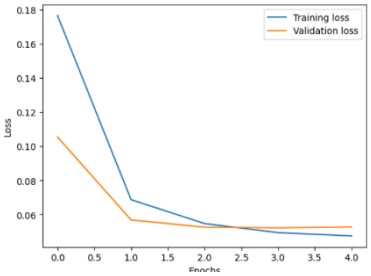
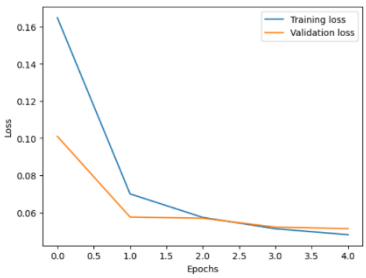
After finding the optimal threshold of each of the model, we insert it into line 287 and finding precision, recall, and f1 score of the predicted result compared to real test value.

4.4. Result

All the results that have been acquired is shown below from training graph, early evaluation, and selected threshold of each of our model. From each of the model the author will show training loss graph, early evaluation, and threshold finding using validation data.

4.4.1. LSTM Model

Table 4.1. Regular LSTM training graph and early evaluation

	Cleaned	Cleaned without S. removal	Not Cleaned
Training Graph			
Loss Graph			
Early evaluation	`loss: 0.0873` `accuracy: 0.9976`	`loss: 0.0758` `accuracy: 0.9959`	`loss: 0.0841` `accuracy: 0.9968`

Based on this early evaluation on regular LSTM, model with clean preprocessing method provide the best accuracy with 0.9976 but not by much compared to the other two, while the model with the least loss is the model with Cleaned without S. removal with loss value of 0,0758

Table 4.2. Finding Regular LSTM threshold on each of Pre-processing type

Regular LSTM														
LSTM cleaned + stopword					LSTM cleaned without stopword					LSTM not cleaned				
Th	Accuracy	Precision	Recall	F1- Score	Th	Accuracy	Precision	Recall	F1- Score	Th	Accuracy	Precision	Recall	F1- Score
0,1	87,58%	54,87%	84,80%	65,98%	0,1	86,83%	56,64%	85,98%	67,97%	0,1	86,46%	53,90%	84,75%	65,47%
0,2	89,80%	62,23%	78,92%	69,17%	0,2	90,24%	66,27%	78,65%	71,89%	0,2	89,72%	63,05%	77,58%	69,29%
0,3	90,98%	67,56%	72,94%	69,83%	0,3	91,41%	70,77%	72,84%	71,74%	0,3	91,00%	68,13%	70,09%	68,79%
0,4	91,40%	70,94%	66,50%	68,18%	0,4	91,73%	74,33%	67,27%	70,10%	0,4	91,51%	70,49%	64,28%	67,20%
0,5	91,46%	77,66%	59,41%	65,56%	0,5	91,77%	75,29%	61,85%	67,81%	0,5	91,61%	73,48%	65,84%	65,84%
0,6	91,35%	76,36%	52,45%	61,99%	0,6	91,66%	78,00%	56,56%	65,32%	0,6	91,59%	76,11%	54,88%	63,76%
0,7	91,17%	79,67%	43,96%	56,17%	0,7	91,33%	80,87%	48,39%	59,91%	0,7	91,38%	79,01%	47,89%	59,59%
0,8	90,40%	83,55%	29,23%	41,24%	0,8	90,68%	83,94%	38,05%	50,17%	0,8	90,35%	63,04%	29,60%	40,14%
0,9	89,87%	42,53%	7,14%	12,23%	0,9	90,18%	64,79%	24,96%	35,36%	0,9	90,05%	41,87%	15,55%	22,68%
Precision Median		70,94%	Threshold Chosen 0.4		Precision Median		74,33%	Threshold Chosen 0.4		Precision Median		68,13%	Threshold Chosen 0.3	

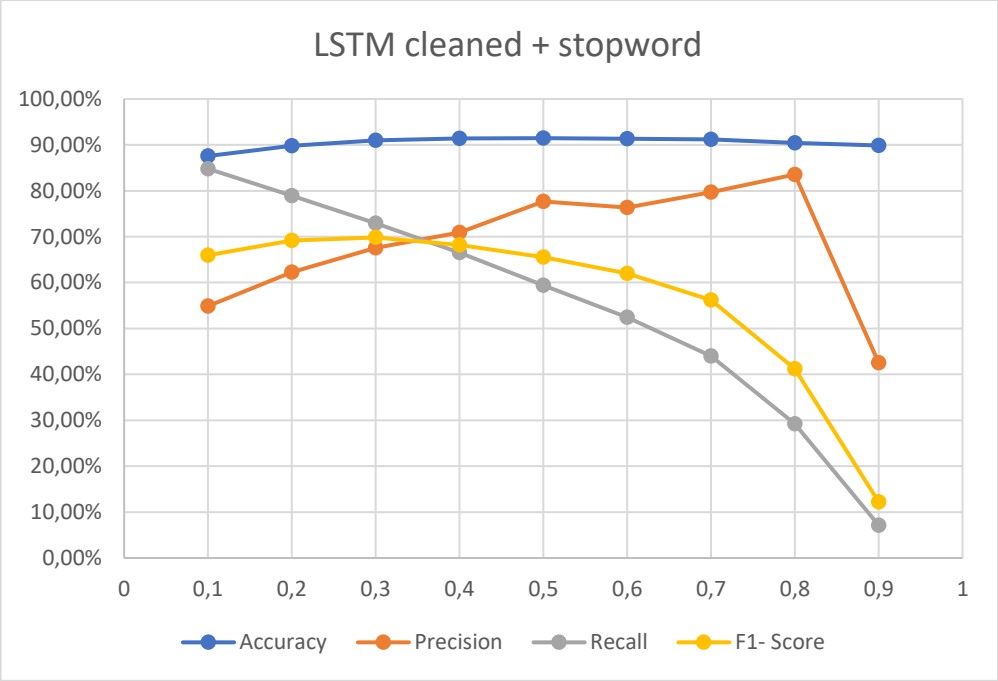


Figure 4.4 Threshold Graph of LSTM Model Cleaned + Stopword

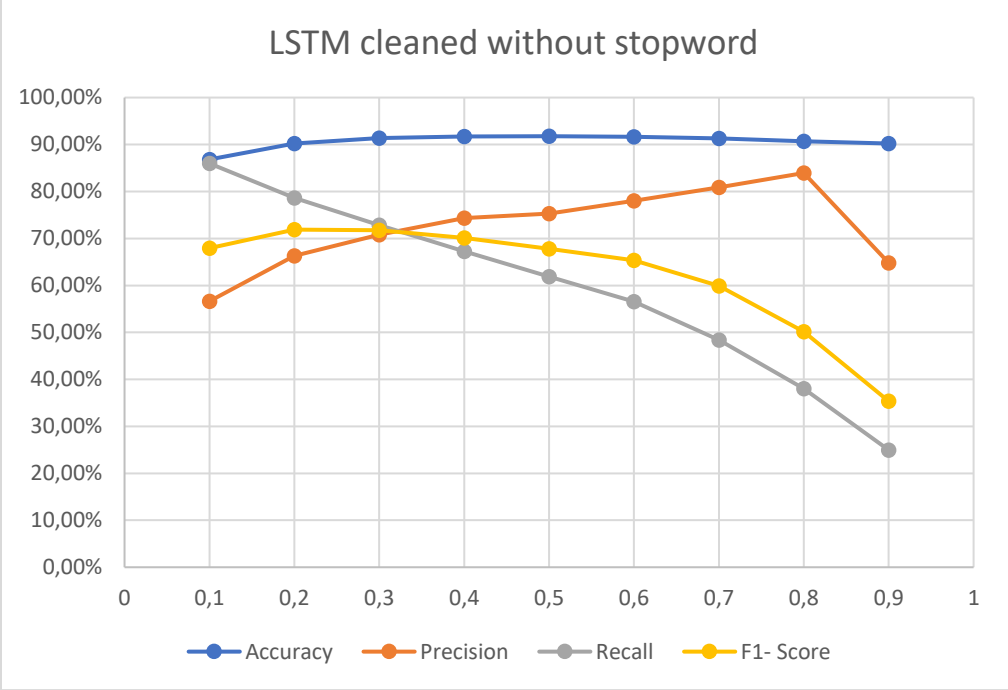


Figure 4.5 Threshold Graph of LSTM Model Cleaned without Stopword

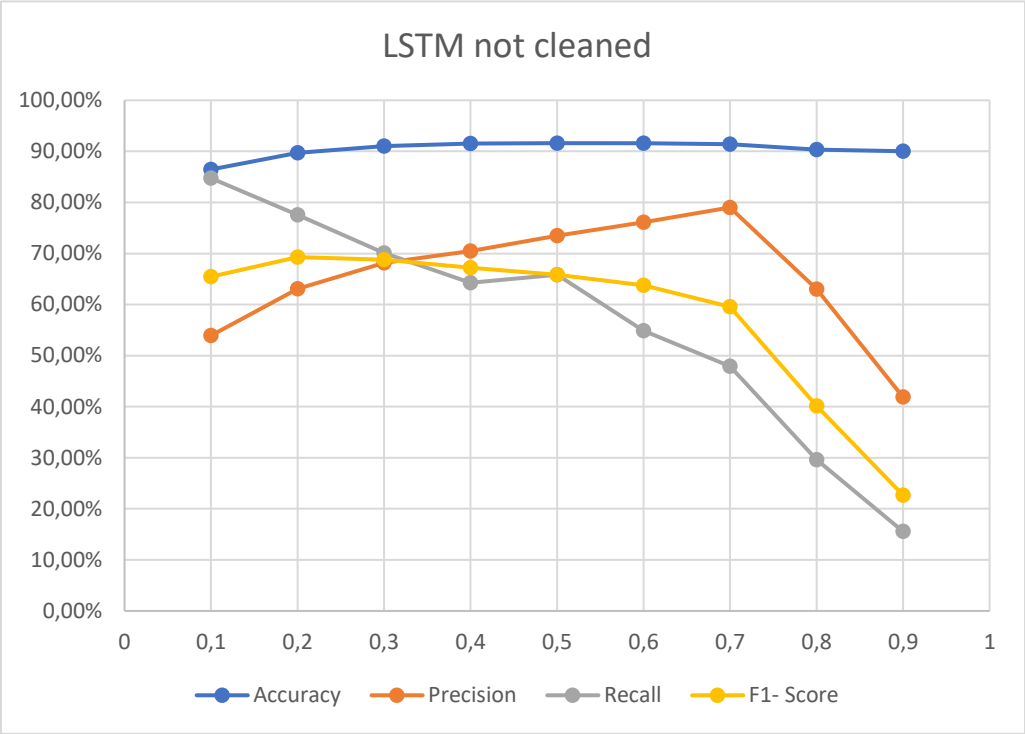
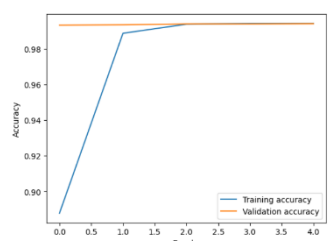
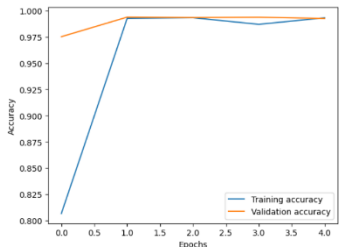
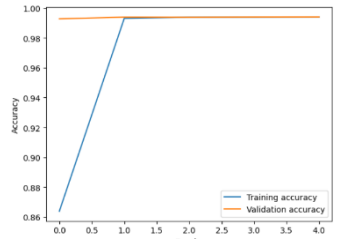
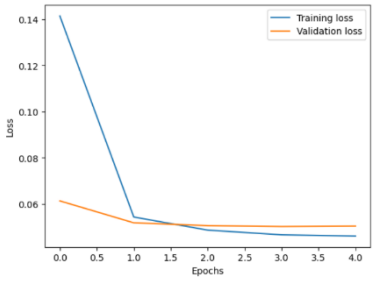
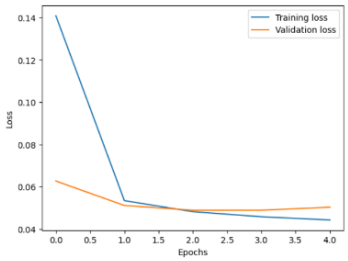
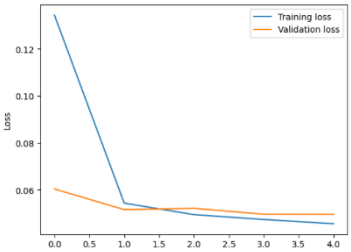


Figure 4.6 Threshold Graph of LSTM Model notcleaned

With the data shown on table 4.2 we can graphed the data that is shown on figure 4.4, figure 4.5, and figure 4.6 and we can evaluate our findings. On all of LSTM graph we see the accuracy value is constantly high between all threshold while the precision value is steadily increasing before dropping off, f1-score value is also increasing slightly with the increase of threshold before maximum point and after that keep decreasing, and recall is constantly decreasing with the increase of threshold .Using median value of precision to find the optimal threshold we found that our threshold is on the point where the precision value is still high with the f1-score value is second best of all threshold. This means by using threshold found we have a balanced model with a slight bias to precision. Based on evaluation result, author will uses the threshold of 0,4 in LSTM cleaned + stopword removal and LSTM cleaned without stopword while using 0.3 in LSTM without cleaning.

4.4.2. BiLSTM Models

Table 4.3. Bidirectional LSTM training graph and early evaluation

	Cleaned	Cleaned without S. removal	Not Cleaned
Training Graph			
Loss Graph			
Early evaluation	`loss: 0.0820` `accuracy: 0.9975`	`loss: 0.0770` `accuracy: 0.9930`	`loss: 0.0771` `accuracy: 0.9687`

BiLSTM model compared with regular LSTM before has shown great accuracy and loss on evaluation data. Compared to regular LSTM, BiLSTM shown more prone overfitting tendency compared to a regular (non-bidirectional) model this can be caused by BiLSTM model. Usually, to compensate overfit on the model we can use early stopping, author uses method that save the model that have the highest value of val_accuracy on training epoch so if the on next epoch isn't improving / stagnating the new epoch isn't saved to mitigate some of overfitting problem. After that our result shows we have model cleaned with stopword removal have the best accuracy with 0.9975 while the model with the least loss is model cleaned without stopword removal.

Table 4.4. Finding BiLSTM threshold on each of Pre-processing type

Bidirectional LSTM														
Bidir LSTM cleaned + stopword					Bidir LSTM cleaned without stopword					Bidir LSTM not cleaned				
Th	Accuracy	Precision	Recall	F1- Score	Th	Accuracy	Precision	Recall	F1- Score	Th	Accuracy	Precision	Recall	F1- Score
0,1	87,65%	56,95%	86,14%	68,24%	0,1	82,93%	46,85%	91,09%	61,67%	0,1	87,31%	55,67%	87,39%	67,61%
0,2	90,19%	65,61%	79,33%	71,64%	0,2	88,46%	59,05%	84,65%	69,49%	0,2	89,96%	64,21%	80,94%	71,35%
0,3	91,14%	71,66%	73,81%	72,43%	0,3	90,29%	64,56%	79,22%	71,13%	0,3	91,10%	69,06%	75,77%	72,19%
0,4	91,92%	75,19%	68,89%	71,90%	0,4	91,23%	69,77%	73,86%	71,44%	0,4	91,66%	73,15%	71,08%	72,06%
0,5	91,90%	78,28%	62,93%	69,36%	0,5	91,66%	74,05%	67,68%	69,59%	0,5	91,97%	76,94%	66,19%	70,91%
0,6	91,84%	82,50%	57,67%	66,58%	0,6	91,82%	75,49%	62,20%	68,17%	0,6	91,92%	80,42%	60,28%	67,86%
0,7	91,55%	81,09%	51,59%	62,78%	0,7	91,81%	79,01%	56,23%	65,60%	0,7	91,75%	80,15%	54,47%	64,69%
0,8	91,09%	83,95%	43,11%	56,06%	0,8	91,31%	82,64%	46,93%	59,10%	0,8	91,45%	83,15%	46,96%	59,54%
0,9	90,26%	86,79%	29,06%	40,06%	0,9	90,27%	81,43%	27,00%	37,35%	0,9	90,53%	86,55%	32,87%	45,87%
Precision Median		78,28%	Threshold Chosen 0.5		Precision Median		74,05%	Threshold Chosen 0.5		Precision Median		76,94%	Threshold Chosen 0.5	

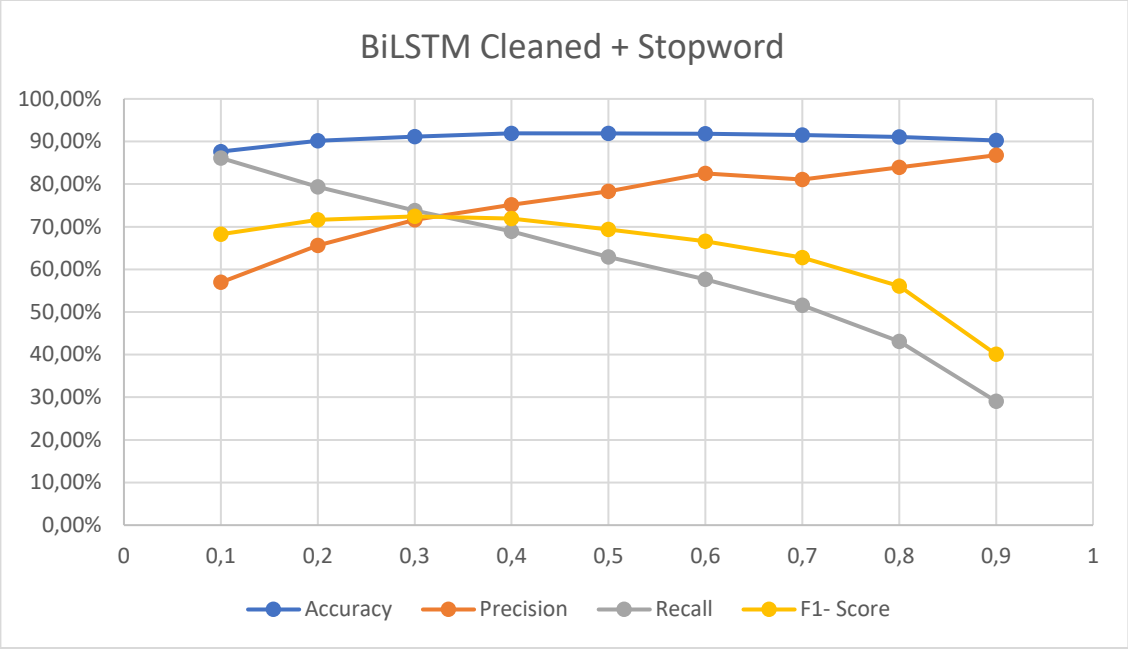


Figure 4.7 Threshold Graph of BiLSTM cleaned with stopword removal

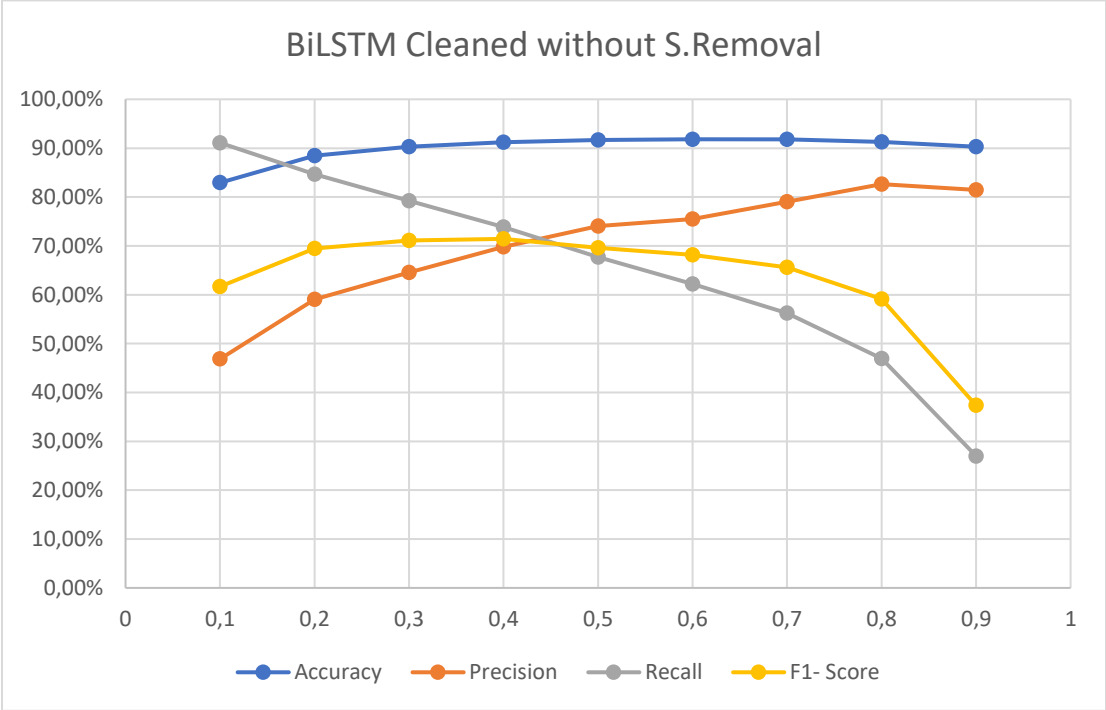


Figure 4.8 Threshold Graph of BiLSTM cleaned without stopword removal

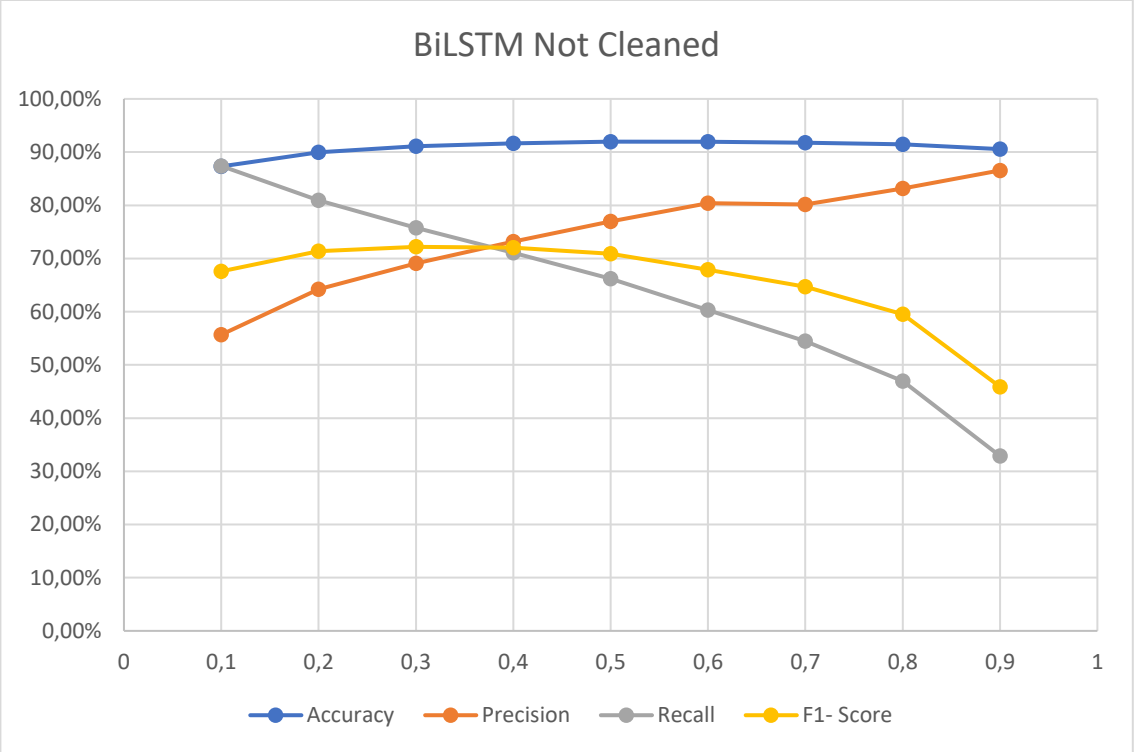
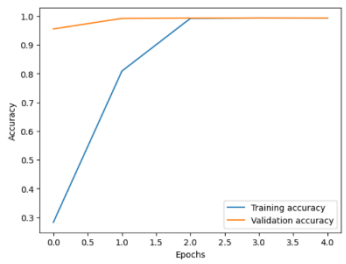
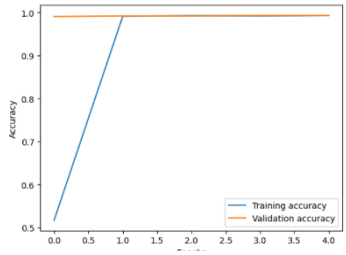
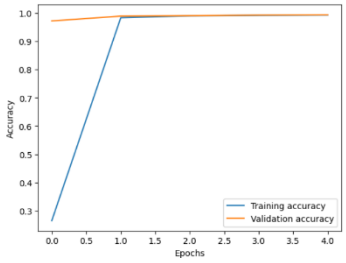
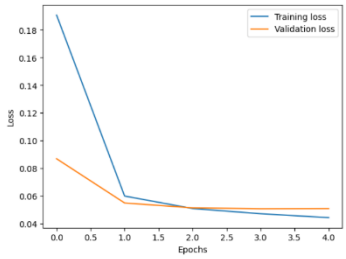
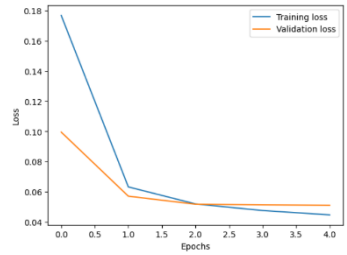
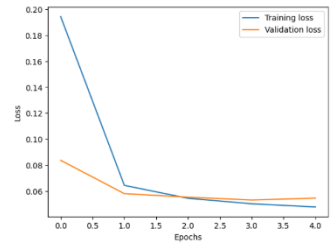


Figure 4.9 Graph of BiLSTM not cleaned

After collecting the data for finding our threshold shown at table 4.4 and making the graph of the table at figure 4.7, figure 4.8, and figure 4.9 to show how our threshold performs. Similar to regular LSTM graph, On BiLSTM graph we can see the accuracy value is constantly high between all threshold while the precision value is steadily increasing before dropping off, f1-score value is also increasing slightly with the increase of threshold before maximum point and after that keep decreasing, and recall is constantly decreasing with the increase of threshold. Using median value of precision to find the optimal threshold we found that our threshold is on the point where the precision value is high and f1-score value is second best in all threshold, in this case author will use the threshold of 0,5 across all BiLSTM model

4.4.3. GRU Model

Table 4.5. GRU Model training graph & early evaluation

	Cleaned	Cleaned without S. removal	Not Cleaned
Training Graph			
Loss Graph			
Early evaluation	`loss: 0.1136` `accuracy: 0.9966`	`loss: 0.0870` `accuracy: 0.9958`	`loss: 0.0948` `accuracy: 0.9960`

Based on the training graph of these GRU models on table 4.5, we can see on all of them training accuracy is quickly improving while the loss function is keep decreasing until around epoch number 2 and after that is slightly improving this is the case of overfitting. Based on early evaluation the accuracy value of these models are comparable to the previous two model but shown slightly higher loss compared to LSTM and BiLSTM model.

Table 4.6. Finding GRU threshold on each of Pre-processing type

Regular GRU model														
GRU cleaned + stopword					GRU cleaned no stopword					GRU not cleaned				
Thr	Accuracy	Precision	Recall	F1- Score	Thr	Accuracy	Precision	Recall	F1- Score	Thr	Accuracy	Precision	Recall	F1- Score
0,1	84,41%	49,89%	89,10%	63,60%	0,1	83,28%	48,69%	90,05%	62,90%	0,1	82,44%	44,74%	91,12%	59,64%
0,2	88,12%	59,30%	82,68%	68,81%	0,2	87,76%	59,20%	84,20%	68,85%	0,2	86,34%	53,47%	85,68%	65,62%
0,3	90,08%	66,36%	77,65%	71,06%	0,3	89,70%	65,30%	78,75%	70,45%	0,3	88,70%	60,52%	80,89%	68,35%
0,4	91,24%	71,21%	72,62%	71,45%	0,4	90,84%	70,42%	74,10%	70,90%	0,4	90,12%	65,37%	76,94%	70,06%
0,5	91,78%	76,89%	67,25%	70,42%	0,5	91,37%	73,34%	69,15%	69,99%	0,5	91,06%	69,15%	72,14%	69,92%
0,6	91,83%	79,87%	61,58%	68,03%	0,6	91,72%	74,27%	64,81%	69,07%	0,6	91,50%	70,70%	67,53%	69,07%
0,7	91,76%	81,80%	55,41%	65,07%	0,7	91,82%	78,10%	59,37%	67,07%	0,7	91,69%	74,64%	62,19%	67,82%
0,8	91,35%	84,58%	46,42%	59,05%	0,8	91,51%	81,70%	50,89%	61,80%	0,8	91,59%	79,03%	54,14%	64,09%
0,9	90,51%	86,74%	31,07%	44,00%	0,9	90,60%	85,49%	34,68%	47,20%	0,9	90,87%	84,36%	37,31%	50,66%
Precision Median		76,89%	Threshold Chosen 0.5		Precision Median		73,34%	Threshold Chosen 0.5		Precision Median		69,15%	Threshold Chosen 0.5	

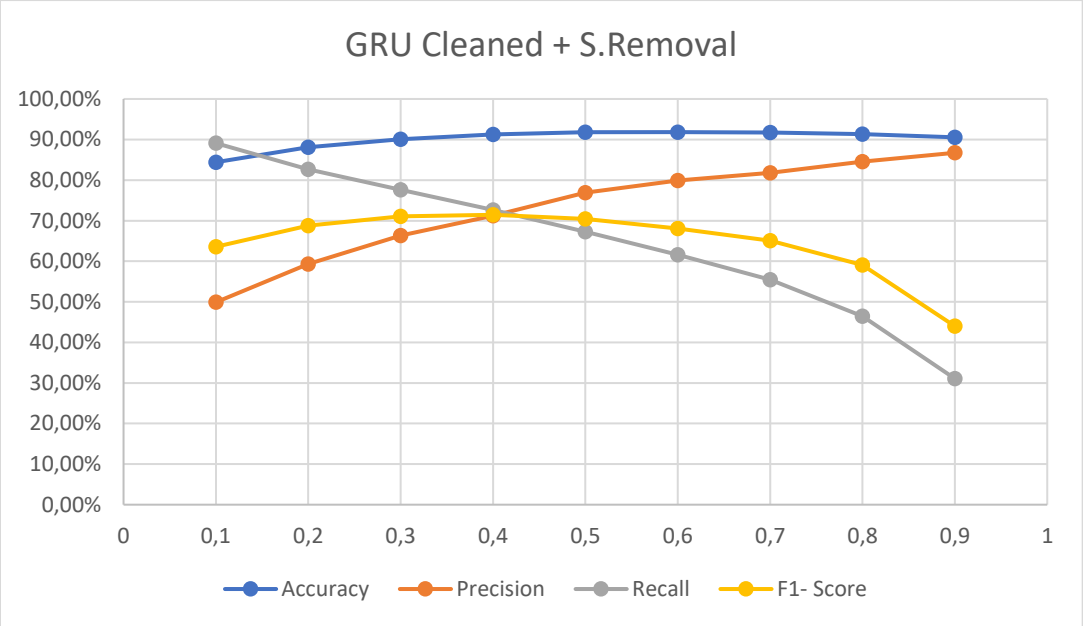


Figure 4.10 Threshold Graph of GRU Cleaned and stopword removal

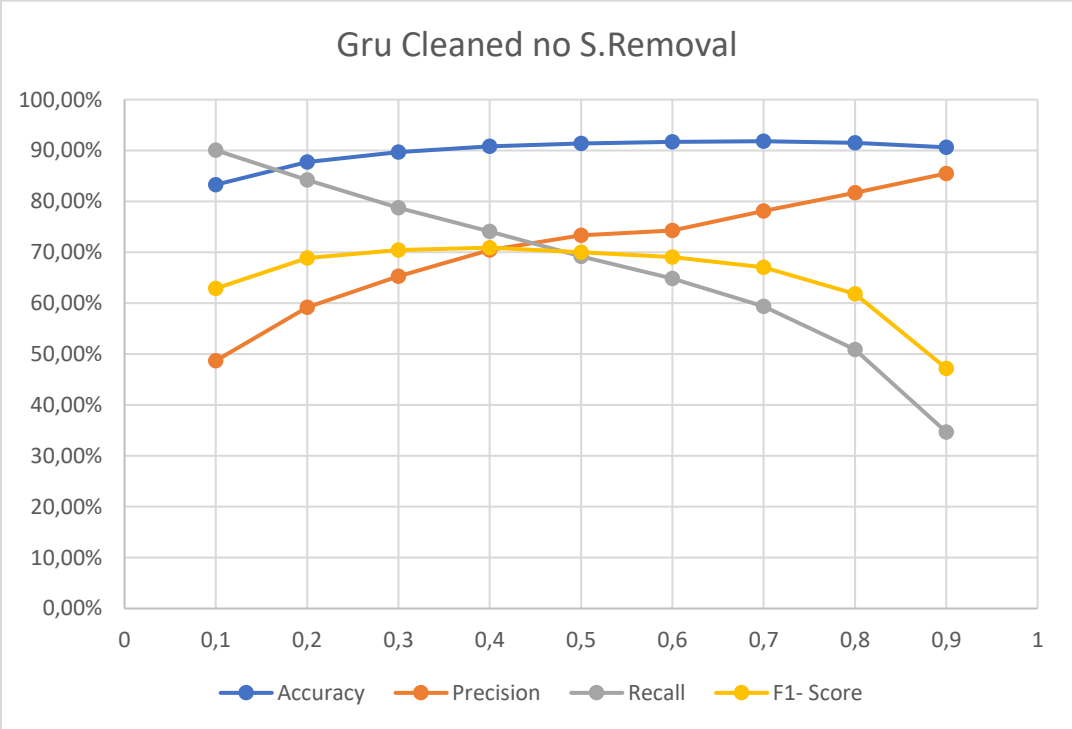


Figure 4.11 Threshold Graph of GRU cleaned without stopword removal

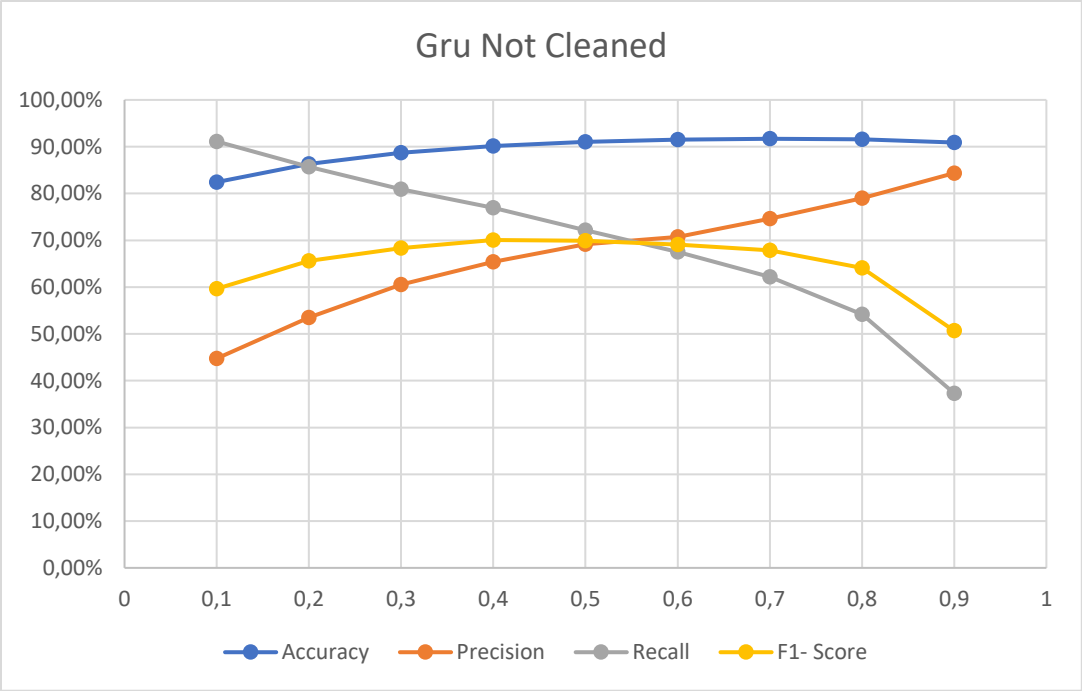
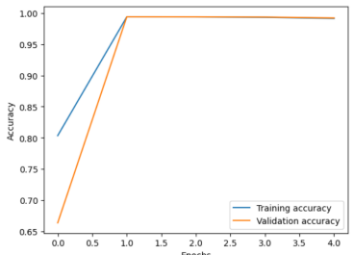
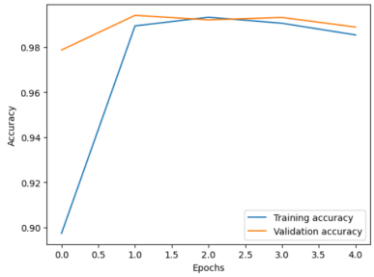
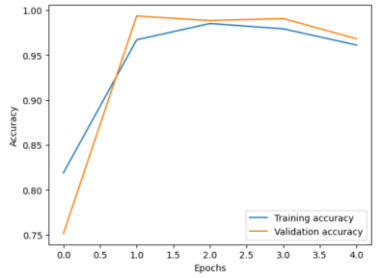
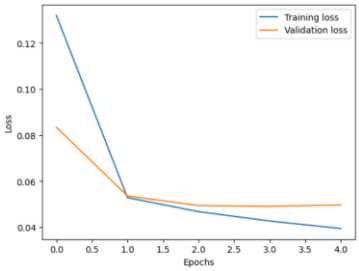
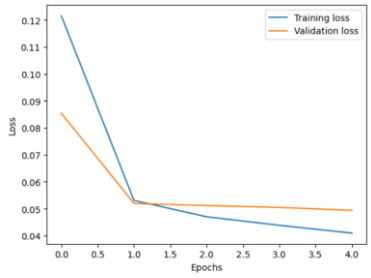
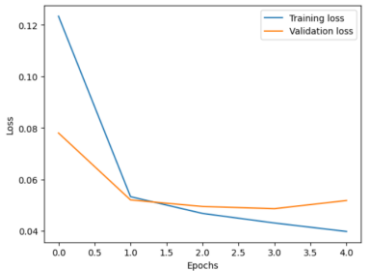


Figure 4.12 Threshold Graph of GRU without cleaning

And from the threshold graph based on table 4.5 shown at figure 4.10, figure 4.11, and figure 4.12 these GRU models have the same characteristic as prior LSTMs model with accuracy value is constantly high between all threshold while the precision and f1 score value is steadily increasing before dropping off, and recall is constantly decreasing with the increase of threshold. By using precision median, we found 0.5 as the value for the three GRU models

4.4.4. BiGRU Models

Table 4.7. BiGRU Training Graph and early evaluation

	Cleaned	Cleaned without S. removal	Not Cleaned
Training Graph			
Loss Graph			
Early evaluation	`loss: 0.1146` `accuracy: 0.9960`	`loss: 0.0760` `accuracy: 0.9726`	`loss: 0.0835` `accuracy: 0.9886`

On training graph and early evaluation shown at table 4.7 of BiGRU model, the author found no significant difference in accuracy and loss compared to regular GRU counterpart and LSTMs model in terms of accuracy and loss. But if we compare the graph, we can find the graph shape is not as flat compared to regular GRU counterpart showing some improvement before plateau showing sign off overfitting.

Table 4.8. Finding threshold BiGRU model

Bidirectional GRU model														
Bidir GRU cleaned + stopword					Bidir GRU cleaned no stopword					Bidir GRU not cleaned				
Thr	Accuracy	Precision	Recall	F1- Score	Thr	Accuracy	Precision	Recall	F1- Score	Thr	Accuracy	Precision	Recall	F1- Score
0,1	80,01%	42,33%	92,25%	57,68%	0,1	84,82%	50,07%	89,71%	63,86%	0,1	82,69%	48,45%	89,78%	62,59%
0,2	85,97%	53,99%	86,87%	66,42%	0,2	88,76%	59,81%	83,64%	69,33%	0,2	87,97%	59,08%	82,63%	68,79%
0,3	88,74%	61,49%	81,54%	69,79%	0,3	90,28%	64,40%	78,98%	70,73%	0,3	90,11%	65,50%	77,66%	70,77%
0,4	90,45%	67,58%	77,20%	71,41%	0,4	91,14%	68,97%	75,08%	71,78%	0,4	91,08%	69,86%	73,10%	71,38%
0,5	91,41%	71,55%	72,34%	71,77%	0,5	91,69%	72,71%	70,70%	71,56%	0,5	91,62%	74,18%	68,11%	70,80%
0,6	91,90%	75,95%	66,45%	70,38%	0,6	91,94%	76,37%	65,66%	69,96%	0,6	91,81%	78,22%	62,04%	68,42%
0,7	91,95%	80,63%	60,05%	67,82%	0,7	91,93%	81,03%	60,55%	67,93%	0,7	91,80%	84,09%	55,61%	65,11%
0,8	91,62%	81,58%	50,82%	62,46%	0,8	91,77%	80,79%	53,03%	63,92%	0,8	91,31%	83,36%	46,13%	57,83%
0,9	90,74%	86,10%	34,82%	47,94%	0,9	91,02%	84,78%	40,03%	53,18%	0,9	90,56%	84,43%	32,75%	43,84%
Precision Median		71,55%	Threshold Chosen 0.5		Precision Median		72,71%	Threshold Chosen 0.5		Precision Median		74,18%	Threshold Chosen 0.5	

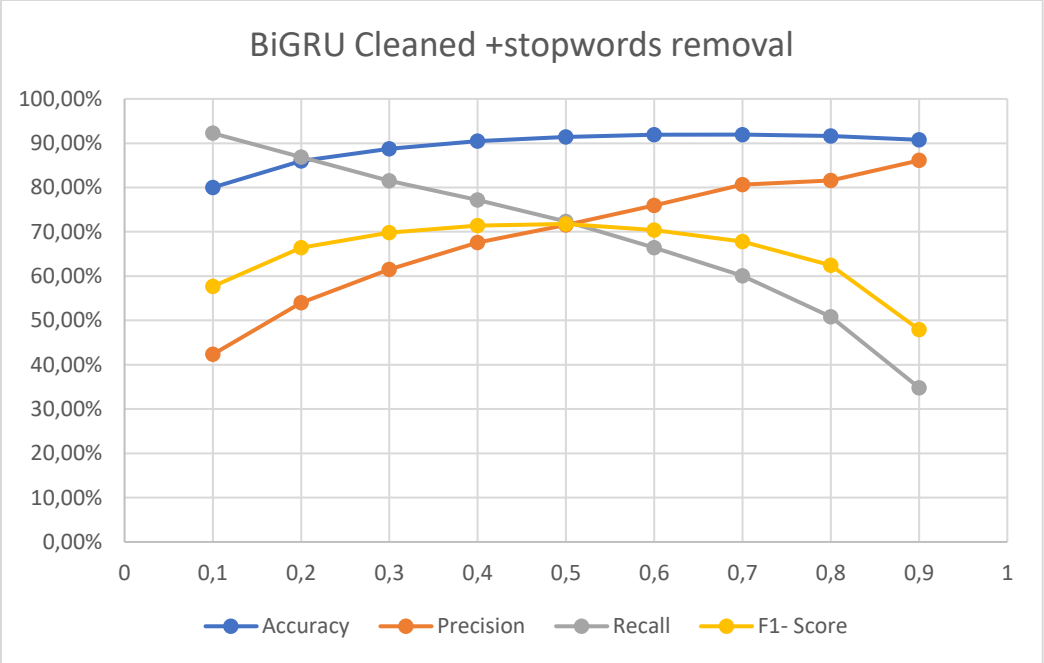


Figure 4.13 Threshold Graph of BiGRU Cleaned with stopword removal

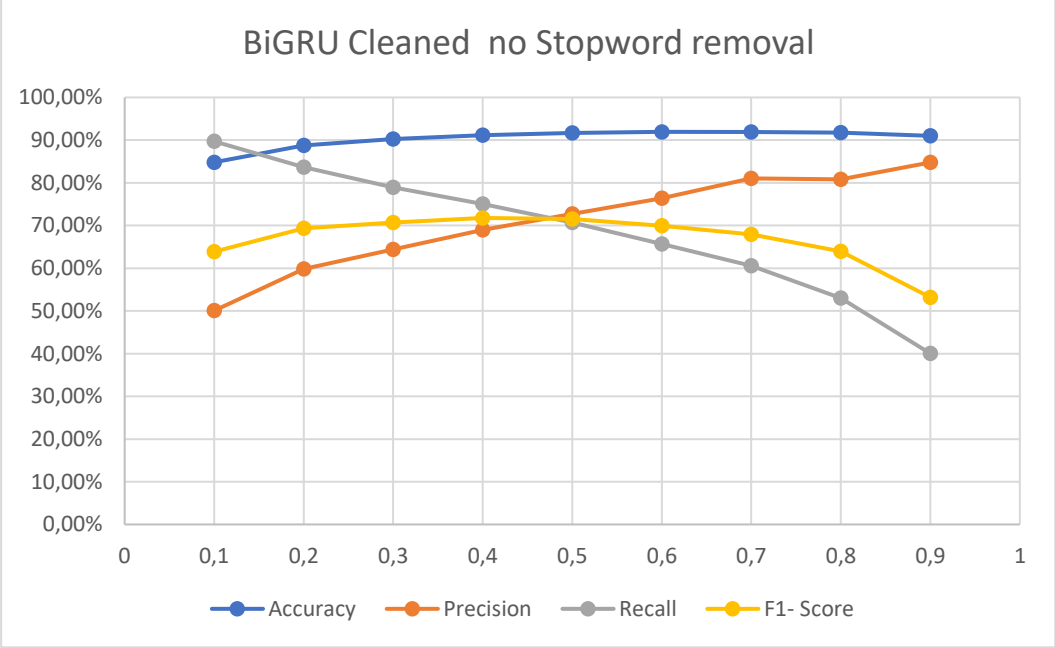


Figure 4.14 Threshold Graph of BiGRU cleaned without stopword removal

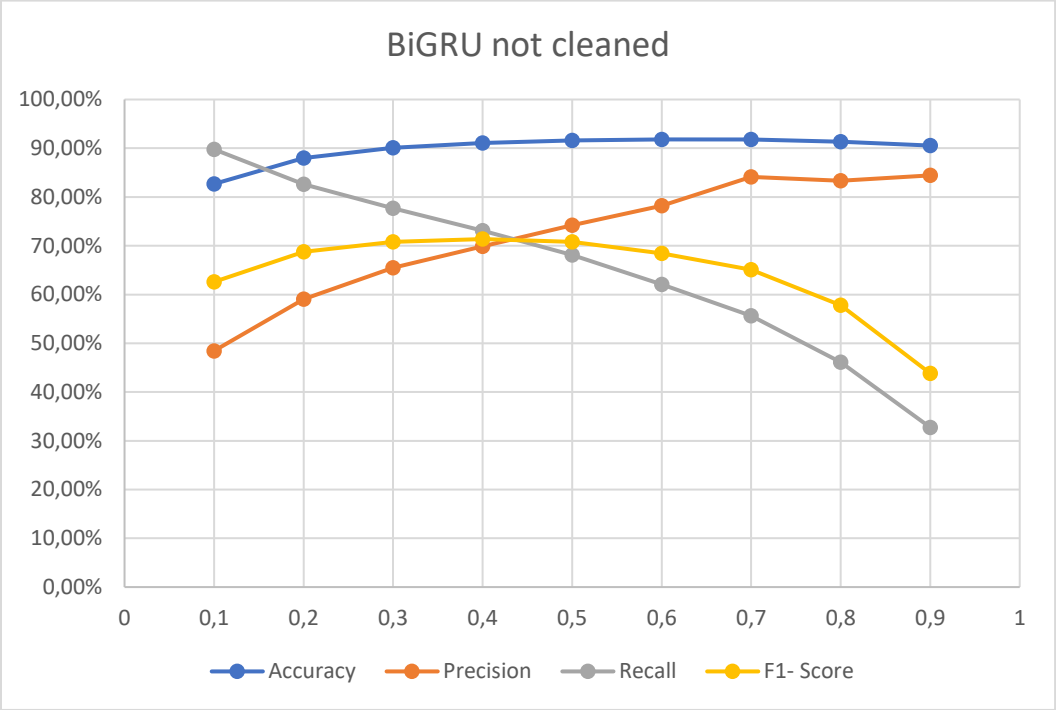


Figure 4.15 Threshold Graph of BiGRU without cleaning

And from the graph above at figure 4.13, figure 4.14, and figure 4.15, BiGRU models have the same characteristic as prior LSTMs model with accuracy value is constantly high between all threshold while the precision and f1 score value is steadily increasing before dropping off, and recall is constantly decreasing with the increase of threshold. By using precision median we found 0.5 as the value for the three BiGRU models. On both BiGRU model with cleaning we found perfect cutoff before F1-score and recall value decreasing, while the model with no cleaning the threshold is one step after peak f1-score.

4.4.5. Implementation Threshold on Test Data

We use the evaluation threshold to implement into our test data, The result found is laid on table 4.9 and the data is graphed into figure 4.8. Based on the data found BiLSTM model is the best performing model with all of three model having the best accuracy compared to regular LSTM and both of GRU Model while boating the highest f1-score value.

Table 4.9. Test Data Result

Implement Test Data					
Method	**Threshold**	**Accuracy**	**Precision**	**Recall**	**F1- Score**
LSTM cleaned + stopword	0,4	84,493%	43,459%	72,700%	54,177%
LSTM cleaned no stopword	0,4	86,005%	49,131%	70,796%	57,773%
LSTM not cleaned	0,3	85,017%	44,989%	73,934%	55,851%
Bidir LSTM cleaned + stopword	0,5	86,674%	54,814%	68,747%	60,128%
Bidir LSTM cleaned no stopword	0,5	86,083%	50,554%	70,955%	58,664%
Bidir LSTM not cleaned	0,5	87,208%	55,205%	68,540%	60,623%
GRU Cleaned + stopword	0,5	81,848%	44,922%	75,045%	54,249%
GRU Cleaned no stopword	0,5	84,917%	50,144%	72,106%	57,874%
GRU not cleaned	0,5	84,695%	45,934%	74,452%	55,970%
BiGRU Cleaned + stopword	0,5	77,927%	45,097%	77,149%	53,492%
BiGRU Cleaned no stopword	0,5	85,931%	49,748%	74,652%	59,535%
BiGRU not cleaned	0,5	85,681%	52,854%	71,231%	59,960%

Table 4.10. Test data result with the best accuracy from each model

Best Result from each type of Model				
Method	**Accuracy**	**Precision**	**Recall**	**F1- Score**
LSTM cleaned no stopword	86,005%	49,131%	70,796%	57,773%
Bidir LSTM not cleaned	87,208%	55,205%	68,540%	60,623%
GRU Cleaned no stopword	84,917%	50,144%	72,106%	57,874%
BiGRU Cleaned no stopword	85,931%	49,748%	74,652%	59,535%

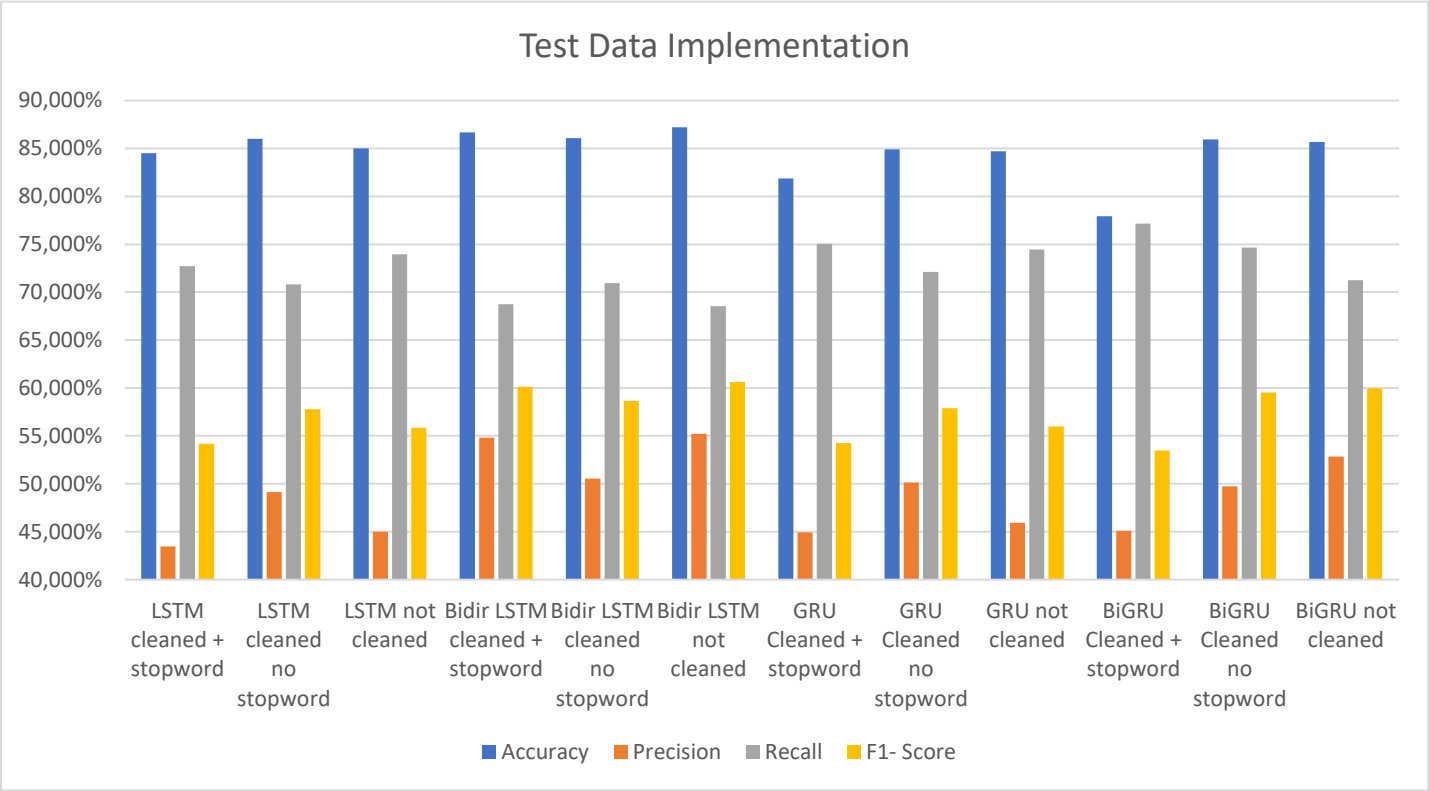


Figure 4.16 Test Data Graph

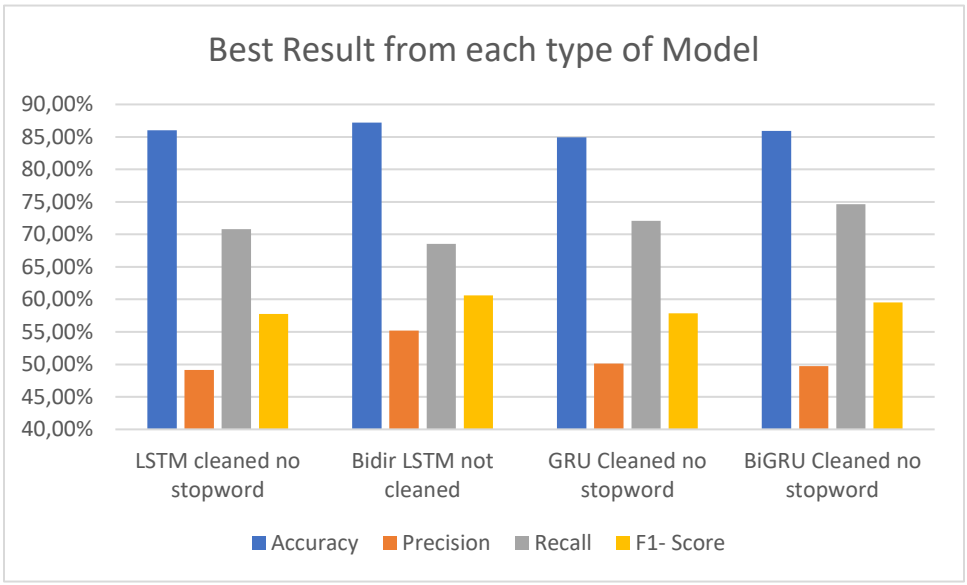


Figure 4.17 Best Result of Each Model

4.5. Discussion

The result found on figure 4.16 and table 4.9, the author found that BiLSTM model is the best performing model compared to the other three model by having the highest accuracy value, precision value and f1-score compared to regular LSTM and both of GRU Model

Table 4.11. BiLSTM Result

Implement Test Data					
Method	**Threshold**	**Accuracy**	**Precision**	**Recall**	**F1- Score**
Bidir LSTM cleaned + stopword	0,5	86,674%	54,814%	68,747%	60,128%
Bidir LSTM cleaned no stopword	0,5	86,083%	50,554%	70,955%	58,664%
Bidir LSTM not cleaned	0,5	87,208%	55,205%	68,540%	60,623%

On BiLSTM model, cleaned + stopword impact is slightly worse than non-cleaned one that is close within 0.5 between accuracy of 86,674% in cleaned + stopword and 87,208% in non-cleaned one , and on f1-score 60,623% in cleaned + stopword and 60,128 in non-cleaned. while prior model result is close cleaned without stopword yielded worse result compared to the other two in all except recall value.

Table 4.12. GRU and BiGRU Model test result

Implement Test Data					
Method	**Threshold**	**Accuracy**	**Precision**	**Recall**	**F1- Score**
GRU Cleaned + stopword	0,5	81,848%	44,922%	75,045%	54,249%
GRU Cleaned no stopword	0,5	84,917%	50,144%	72,106%	57,874%
GRU not cleaned	0,5	84,695%	45,934%	74,452%	55,970%
BiGRU Cleaned + stopword	0,5	77,927%	45,097%	77,149%	53,492%
BiGRU Cleaned no stopword	0,5	85,931%	49,748%	74,652%	59,535%
BiGRU not cleaned	0,5	85,681%	52,854%	71,231%	59,960%

Shown at table 4.11, the result of preprocessing on GRU and BiGRU model, using cleaned train data and no stopword removal provide increase in accuracy, and f1-score, and in regular GRU case is also increasing the precision value by up to 5% from 45,934% to 50,144%. The same method is reducing precision value n BiGRU model with precision from 42.while cleaned + stopword yet again having worst result in all category except for recall.

Table 4.13. LSTM Model test result

Implement Test Data					
Method	**Threshold**	**Accuracy**	**Precision**	**Recall**	**F1- Score**
LSTM cleaned + stopword	0,4	84,493%	43,459%	72,700%	54,177%
LSTM cleaned no stopword	0,4	86,005%	49,131%	70,796%	57,773%
LSTM not cleaned	0,3	85,017%	44,989%	73,934%	55,851%

On regular LSTM shown at table 4.13, cleaned without stopword removal method yielded clear improvement in all category except recall, with non-cleaned method following second on both precision and accuracy. With all of the result in mind even though cleaning data can improve some in some metrics, not cleaning the data can provide with consistent base line across all models. This might be a result of higher relevant information that might be missing as a result of cleaning process.

On all of the model they provide relatively high accuracy but model precision and F1-score value are lower than desired, this is indicating that all models are making a lot of false positive prediction that could be problematic case scenario on toxicity classification meaning non toxic user can be flagged for toxicity. This is most likely this case of the dataset being heavily imbalanced. To mitigate this, we can implement better classification threshold or do a better job at feature engineering our result can most likely be improved upon.

CHAPTER 5
CONCLUSION

In this research, the author compares LSTM, BiLSTM, GRU, and BiGRU on toxicity classification. From the result it can be concluded that all of the models can be used for toxicity classification. but on the test data BiLSTM model outperform the other model. The best resulting BiLSTM is the one that is not applied with any cleaning with 87,208% Accuracy, 55,205% Precision, 68,540% Recall, and 60,623% F1-Score followed by BiLSTM model with cleaning and stop word removal with the result 86,674% Accuracy, 54,814% Precision, 68,747% Recall, and 60,128% F1-score.

Preprocessing with cleaning the data without removing the stopwords improved results metric in accuracy, precision, and f1-score on LSTM, GRU, BiGRU. This improvement while noticeable, is still relatively close to base result without any kind of cleaning. The author believes it is more impactful to the results, by modifying the model structure itself to suit the dataset better will result in better performance metrics by avoiding underfitting/overfitting rather than focusing on different type of preprocessing.

All models have relatively high accuracy but sacrifice in precision value and F1-score value, this is indicating that all models is making a lot of false positive prediction that could be problematic case scenario on toxicity classification meaning nontoxic user can be flagged for toxicity. This is most likely this case of the dataset being heavily imbalanced. By implementing better classification threshold or do a better job at feature engineering especially specifying the case for multi label classification result metrics can most likely be improved upon.

For the suggestion, author believes that model based on this paper can be developed more by adding different type of preprocessing, training and implementing the model on dataset with different languages, and modifying model structure further to suit the dataset. Author also recommends to trying out many kinds of pre trained embedding instead of using custom embedding such as Word2Vec, BERT, GloVe, and other to further find the impact of each type of embedding in toxicity classification.

REFERENCES

[1] S. Murnion, W. J. Buchanan, A. Smales, and G. Russell, "Machine learning and semantic analysis of in-game chat for cyberbullying," *Computers & Security*, vol. 76, pp. 197–213, Jul. 2018, doi: 10.1016/j.cose.2018.02.016.

[2] M. Martens, S. Shen, A. Iosup, and F. Kuipers, "Toxicity detection in multiplayer online games," in *2015 International Workshop on Network and Systems Support for Games (NetGames)*, Zagreb: IEEE, Dec. 2015, pp. 1–6. doi: 10.1109/NetGames.2015.7382991.

[3] K. Dubey, R. Nair, Mohd. U. Khan, and Prof. S. Shaikh, "Toxic Comment Detection using LSTM," in *2020 Third International Conference on Advances in Electronics, Computers and Communications (ICAECC)*, Bengaluru, India: IEEE, Dec. 2020, pp. 1–8. doi: 10.1109/ICAECC50550.2020.9339521.

[4] M. Anand and R. Eswari, "Classification of Abusive Comments in Social Media using Deep Learning," in *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India: IEEE, Mar. 2019, pp. 974–977. doi: 10.1109/ICCMC.2019.8819734.

[5] C. Esposito, G. A. Landrum, N. Schneider, N. Stiefl, and S. Riniker, "GHOST: Adjusting the Decision Threshold to Handle Imbalanced Data in Machine Learning," *J. Chem. Inf. Model.*, vol. 61, no. 6, pp. 2623–2640, Jun. 2021, doi: 10.1021/acs.jcim.1c00160.

[6] M. Ibrahim, M. Torki, and N. El-Makky, "Imbalanced Toxic Comments Classification Using Data Augmentation and Deep Learning," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Orlando, FL: IEEE, Dec. 2018, pp. 875–878. doi: 10.1109/ICMLA.2018.00141.

[7] Z. Mossie and J.-H. Wang, "Vulnerable community identification using hate speech detection on social media," *Information Processing & Management*, vol. 57, no. 3, p. 102087, May 2020, doi: 10.1016/j.ipm.2019.102087.

[8] A. Obadimu, E. Mead, M. N. Hussain, and N. Agarwal, "Identifying Toxicity Within YouTube Video Comment," in *Social, Cultural, and Behavioral Modeling*, vol. 11549, R. Thomson, H. Bisgin, C. Dancy, and A. Hyder, Eds., in Lecture Notes in Computer Science, vol. 11549. , Cham: Springer International Publishing, 2019, pp. 214–223. doi: 10.1007/978-3-030-21741-9_22.

[9] A. Abbasi, A. R. Javed, F. Iqbal, N. Kryvinska, and Z. Jalil, "Deep learning for religious and continent-based toxic content detection and classification," *Sci Rep*, vol. 12, no. 1, p. 17478, Oct. 2022, doi: 10.1038/s41598-022-22523-3.

[10] E. Yazgili and M. Baykara, "Cyberbullying and Detection Methods," in *2019 1st International Informatics and Software Engineering Conference (UBMYK)*, Ankara, Turkey: IEEE, Nov. 2019, pp. 1–5. doi: 10.1109/UBMYK48245.2019.8965514.

[11] F. Mohammad, "Is preprocessing of text really worth your time for online comment classification?" arXiv, Aug. 29, 2018. doi: 10.48550/arXiv.1806.02908.

[12] “jigsaw-toxic-comment-classification-challenge.” Accessed: Jan. 05, 2024. [Online]. Available: https://www.kaggle.com/datasets/julian3833/jigsaw-toxic-comment-classification-challenge

APPENDIX

Importing Prerequisite

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras import utils
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, MaxPooling1D, Dropout, LSTM, Bidirectional, SpatialDropout1D, TextVectorization
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model

    import os
    for dirname, _, filenames in os.walk('/kaggle/input'):
        for filename in filenames:
      print(os.path.join(dirname, filename))
    train_data = pd.read_csv('/kaggle/input/jigsaw-toxic-comment-classification-challenge/train.csv',index_col='id')
```

Pre-processing for cleaning data

```
    import nltk
    from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize
    nltk.download('stopwords')

    def remove_english_stopwords(text):
        stop_words = set(stopwords.words('english'))
        word_tokens = word_tokenize(text)
        filtered_text = [word for word in word_tokens if not word in stop_words]
        return ' '.join(filtered_text)

    def remove_unnecessary_char(text):
        text = re.sub('\n',' ',text) # Remove every '\n'
        text = re.sub('((www\.[^\s]+)|(https?://[^\s]+)|(http?://[^\s]+))',' ',text) # Remove every URL
        text = re.sub('  +', ' ', text) # Remove extra spaces
        return text

    def remove_nonaplhanumeric(text):
        text = re.sub('[^0-9a-zA-Z]+', ' ', text)
        return text
    train_data['comment_text'] = train_data['comment_text'].apply(preprocess)
    train_data.head()
```

Changing Train data by adding comment length

```
    train_data['comment_length'] = train_data['comment_text'].apply(lambda row: len(row))
```

```
train_data.head()
```

Showing comment length of toxic data on train dataset

```
import seaborn as sns
sns.displot(
    data=train_data,
    x="comment_length",
    hue='toxic',
    multiple="stack",
)
print(
    'max', toxic_data['comment_length'].max(),
    'min', toxic_data['comment_length'].min(),
    'mean', toxic_data['comment_length'].mean(),
    'median', toxic_data['comment_length'].median(),
    '75%', toxic_data['comment_length'].quantile(0.75),
)
```

Table joined on test data

```
test_data = test_data.join(test_labels_data)
test_data = test_data[test_data['toxic'] != -1]
test_data.head()
```

LSTM Model

```
model_lstm = Sequential([
    encoder,
    Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=128,
        mask_zero=True,
        input_length=max_comment_len,
    ),
    SpatialDropout1D(0.5),
    LSTM(40, return_sequences=True),
    LSTM(40),
    Dense(6, activation='sigmoid'),
])
model_lstm.compile(optimizer='adam',      loss='binary_crossentropy',
metrics=['accuracy', 'AUC'])

model_lstm_save_path = '/kaggle/working/toxic_model_lstm'
checkpoint_callback_lstm = ModelCheckpoint(
    model_lstm_save_path,
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1,
    save_format='tf',
)
history_lstm = model_lstm.fit(
    x_train,
    y_train,
    validation_data=(x_val,y_val),
    epochs=5,
    batch_size=512,
```

```
84.         callbacks=[checkpoint_callback_lstm],
85.     )
86.
87.     plt.plot(history_lstm.history['accuracy'],
88.              label='Training accuracy')
89.     plt.plot(history_lstm.history['val_accuracy'],
90.              label='Validation accuracy')
91.     plt.xlabel('Epochs')
92.     plt.ylabel('Accuracy')
93.     plt.legend()
94.     plt.show()
95.     plt.plot(history_lstm.history['loss'],
96.              label='Training loss')
97.     plt.plot(history_lstm.history['val_loss'],
98.              label='Validation loss')
99.     plt.xlabel('Epochs')
100.    plt.ylabel('Loss')
101.    plt.legend()
102.    plt.show()
103.
104.    saved_model = load_model(model_lstm_save_path)
105.    saved_model.evaluate(x_val, y_val, verbose=1)
```

BiLSTM Model

```
106.    model_bidirectional = Sequential([
107.        encoder,
108.        Embedding(
109.            input_dim=len(encoder.get_vocabulary()),
110.            output_dim=128,
111.            mask_zero=True,
112.            input_length=max_comment_len,
113.        ),
114.        SpatialDropout1D(0.5),
115.        Bidirectional(LSTM(40, return_sequences=True)),
116.        Bidirectional(LSTM(40)),
117.        Dense(6, activation='sigmoid'),
118.    ])
119.    model_bidirectional.compile(optimizer='adam',
    loss='binary_crossentropy', metrics=['accuracy', 'AUC'])
120.
121.    model_bidir_save_path = '/kaggle/working/toxic_model_bidir'
122.    checkpoint_callback_lstm = ModelCheckpoint(
123.        model_bidir_save_path,
124.        monitor='val_accuracy',
125.        save_best_only=True,
126.        verbose=1,
127.        save_format='tf',
128.    )
129.    history_bidir = model_bidirectional.fit(
130.        x_train,
131.        y_train,
132.        validation_data=(x_val,y_val),
133.        epochs=5,
134.        batch_size=512,
135.        callbacks=[checkpoint_callback_bidir],
136.    )
```

```
137.
138.   plt.plot(history_bidir.history['accuracy'],
139.            label='Training accuracy')
140.   plt.plot(history_bidir.history['val_accuracy'],
141.            label='Validation accuracy')
142.   plt.xlabel('Epochs')
143.   plt.ylabel('Accuracy')
144.   plt.legend()
145.   plt.show()
146.   plt.plot(history_bidir.history['loss'],
147.            label='Training loss')
148.   plt.plot(history_bidir.history['val_loss'],
149.            label='Validation loss')
150.   plt.xlabel('Epochs')
151.   plt.ylabel('Loss')
152.   plt.legend()
153.   plt.show()
154.
155.   saved_model = load_model(model_bidir_save_path)
156.   saved_model.evaluate(x_val, y_val, verbose=1)
157.
```

GRU Model

```
158.   model_gru = Sequential([
159.       encoder,
160.       Embedding(
161.           input_dim=len(encoder.get_vocabulary()),
162.           output_dim=128,
163.           mask_zero=True,
164.           input_length=max_comment_len,
165.       ),
166.       SpatialDropout1D(0.5),
167.       GRU(20, return_sequences=True),
168.       BatchNormalization(),
169.       GRU(20),
170.       Dense(6, activation='sigmoid'),
171.   ])
172.   model_gru_save_path = '/kaggle/working/toxic_model_gru'
173.   checkpoint_callback_gru = ModelCheckpoint(
174.       model_gru_save_path,
175.       monitor='val_accuracy',
176.       save_best_only=True,
177.       verbose=1,
178.       save_format='tf',
179.   )
180.
181.   model_gru.compile(optimizer='adam',      loss='binary_crossentropy',
   metrics=['accuracy', 'AUC'])
182.
183.   history_gru = model_gru.fit(
184.       x_train,
185.       y_train,
186.       epochs=5,
187.       batch_size=512,
188.       validation_data=(x_val,y_val),
189.       callbacks=[checkpoint_callback_gru],
```

```
)
plt.plot(history_gru.history['accuracy'],
         label='Training accuracy')
plt.plot(history_gru.history['val_accuracy'],
         label='Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
plt.plot(history_gru.history['loss'],
         label='Training loss')
plt.plot(history_gru.history['val_loss'],
         label='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
saved_model = load_model(model_gru_save_path)
saved_model.evaluate(x_val, y_val, verbose=1)
```

BiGRU Model

```
model_bi_gru = Sequential([
    encoder,
    Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=128,
        mask_zero=True,
        input_length=max_comment_len,
    ),
    SpatialDropout1D(0.5),
    Bidirectional(GRU(20, return_sequences=True)),
    BatchNormalization(),
    Bidirectional(GRU(20)),
    Dense(6, activation='sigmoid'),
])
model_bi_gru_save_path = '/kaggle/working/toxic_model_bi_gru'
checkpoint_callback_gru = ModelCheckpoint(
    model_bi_gru_save_path,
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1,
    save_format='tf',
)

model_bi_gru.compile(optimizer='adam',   loss='binary_crossentropy',
metrics=['accuracy', 'AUC'])

history_bi_gru = model_bi_gru.fit(
    x_train,
    y_train,
    epochs=5,
    batch_size=512,
    validation_data=(x_val,y_val),
    callbacks=[checkpoint_callback_gru],
)
```

```
242.  plt.plot(history_bi_gru.history['accuracy'],
243.           label='Training accuracy')
244.  plt.plot(history_bi_gru.history['val_accuracy'],
245.           label='Validation accuracy')
246.  plt.xlabel('Epochs')
247.  plt.ylabel('Accuracy')
248.  plt.legend()
249.  plt.show()
250.  plt.plot(history_bi_gru.history['loss'],
251.           label='Training loss')
252.  plt.plot(history_bi_gru.history['val_loss'],
253.           label='Validation loss')
254.  plt.xlabel('Epochs')
255.  plt.ylabel('Loss')
256.  plt.legend()
257.  plt.show()
258.  saved_model = load_model(model_bi_gru_save_path)
259.  saved_model.evaluate(x_val, y_val, verbose=1)
```

Threshold finding

```
260.  from sklearn.metrics import classification_report, f1_score,
   precision_score, recall_score, accuracy_score
261.
262.  thresholds = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
263.
264.  predicts_val = saved_model.predict(x_val, verbose=1)
265.
266.  for threshold in thresholds:
267.    y_pred_val = (predicts_val > threshold)
268.    clr = classification_report(y_val, y_pred_val)
269.    accuracy = accuracy_score(y_val, y_pred_val)
270.    f1 = f1_score(y_val, y_pred_val, average='weighted')
271.    precision = precision_score(y_val, y_pred_val, average='weighted')
272.    recall = recall_score(y_val, y_pred_val, average='weighted')
273.    print(f"Threshold: {threshold}\nClassification Report:\n----------
   ------------\n{clr}\nF1 Score: {f1}\nPrecision: {precision}\nRecall:
   {recall}\n\nAccuracy: {accuracy}\n")
274.
275.  thresholds = [] # selected based on validation threshold
276.
277.  predicts = saved_model.predict(x_test, verbose=1)
278.
279.  y_pred = (predicts > threshold)
280.
281.  f1 = f1_score(y_test, y_pred, average='weighted')
282.  accuracy = accuracy_score(y_test, y_pred)
283.  clr = classification_report(y_test, y_pred)
284.  precision = precision_score(y_test, y_pred, average='weighted')
285.  recall = recall_score(y_test, y_pred, average='weighted')
286.  print(f"Threshold: {threshold}\nClassification Report:\n------------
   ----------\n{clr}\nF1 Score: {f1}\nPrecision: {precision}\nRecall:
   {recall}\n\nAccuracy: {accuracy}\n")
```

Testing saved model

```
test_data[test_data['toxic']==1][:5]['comment_text'].values
test_labels = saved_model.predict(test_data[test_data['toxic']==1][:5]['comment_text'].values)
for labels in test_labels:
    print([ round(lbl, 2) for lbl in labels])
```

Testing saved model with threshold

```
threshold = 0.5  # Setting our threshold here

label_explanations = [
    "toxic",
    "severe_toxic",
    "obscene",
    "threat",
    "insult",
    "identity_hate"
]

test_comments = test_data[test_data['toxic'] == 1][:5]['comment_text'].values
test_labels = saved_model.predict(test_comments)

for comment, labels in zip(test_comments, test_labels):
    binary_labels = [round(lbl) > threshold for lbl in labels]

    print("Comment:", comment)

    # Print labels and their explanations
    for label, explanation in zip(binary_labels, label_explanations):
        print(f"{explanation}: {label}")

    print()
```

PAPER NAME

TA-20.K1.0009.docx

WORD COUNT

8191 Words

CHARACTER COUNT

48167 Characters

PAGE COUNT

47 Pages

FILE SIZE

1.6MB

SUBMISSION DATE

Jan 5, 2024 4:20 PM GMT+7

REPORT DATE

Jan 5, 2024 4:21 PM GMT+7

18% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 15% Internet database
- Crossref database
- 11% Submitted Works database
- 11% Publications database
- Crossref Posted Content database