Applications of Machine Learning: Natural Language Processing and Computer Vision

CMT316 - Assessment 2

9 May 2023

Hate Speech Detection

Using neural networks

Group NO: 1

Supervisor: Dr Yipeng Qin

Group members:

#	NAME	ID	EMAIL
1	Osama Mohammed Aloraini	21118923	alorainiom@cardiff.ac.uk
2	Tok Emin Safa	22107200	toke@cardiff.ac.uk
3	Sawant Uday	22074262	sawantu@cardiff.ac.uk
4	Md Akib Al Jawad Arnob	22099173	ArnobM@cardiff.ac.uk
5	Kadam Swapnil	22071956	KadamS1@cardiff.ac.uk
6	Huajian Li	22081164	Lih92@cardiff.ac.uk
7	Deshpande Kaustubh	22055417	deshpandek@cardiff.ac.uk

Table of Contents

1. Introduction

Hate speech has become a critical issue on social media platforms such as Twitter, leading to harassment and real-life violence. Detecting and preventing hate speech on these platforms is needed. One effective approach is using machine learning algorithms to automatically identify hate speech patterns in tweets. This can help social media companies take action against offenders and protect their users. Our group project analysed and processed the given dataset using neural networks to detect hate speech.

2. Description of the Task / Dataset

The provided datasets are obtained from (Basile et al., 2019) which consist of six text files for hate speech detection in tweets, categorised into training, validation, and test sets. Each set has a file for tweet text and a file for labels, with 0 and 1 indicating "not hate" and "hate." Tweets and labels are extracted into a data frame for analysis, allowing to conduct statistical data analysis and insight into the given dataset

2.1. Dataset description analysis

With extracted data in data frames, tweet distribution across training, test, and validation sets can be analysed. **Figure 1** shows 9000 tweets in the train set, 2970 tweets in the test set, and 1,000 tweets in the validation set. Therefore, the sizes of the validation and test sets represent 11% and 33% of the train set, respectively. Furthermore, **Figure 2** reveals more hate tweets than non-hate tweets in all three datasets. In the training set, there are 3783 non-hate tweets and 5217 hate tweets, creating a partly unbalanced dataset with approximately 42% non-hate tweets and 58% hate tweets.

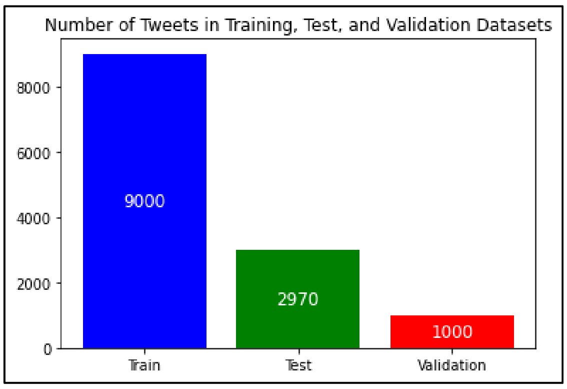


Figure 1

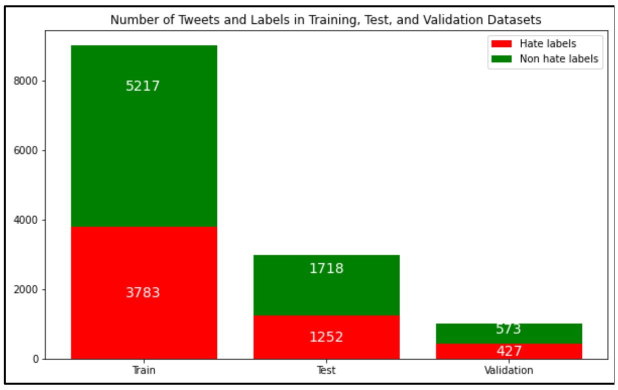


Figure 2

2.2. Existing Issues and Noise in the Train Set

Figure 3 and **Figure 4** reveal noise in the train set, necessitating data pre-processing, since the presence of numbers, capitalised words, hashtag and mention symbols, and stop words may affect the model's performance.

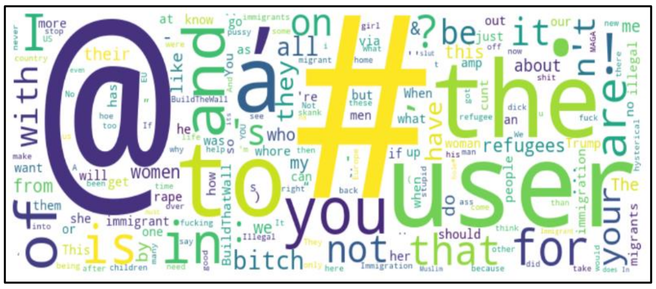


Figure 3

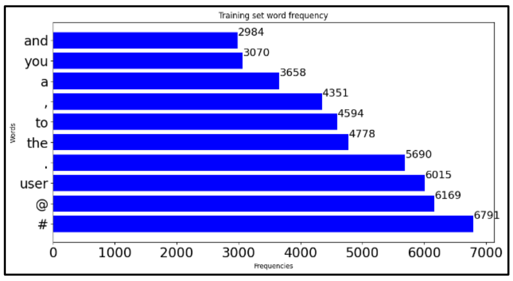


Figure 4

2.3. Train Set Hashtags Analysis

Since Twitter hashtags enable sharing and trending topics, it is important to conduct an analysis of the existing hashtags in the dataset. **Figure 5** shows common hashtags in the training set. To analyse the correlation between hashtags and their labels, **Figure 6** displays the distribution of hate and non-hate speech labels among the top 10 hashtags. The ratio between hate and non-hate is approximately similar for all top 10 hashtags, indicating that having tweets with offensive hashtags does not necessarily imply hate speech.

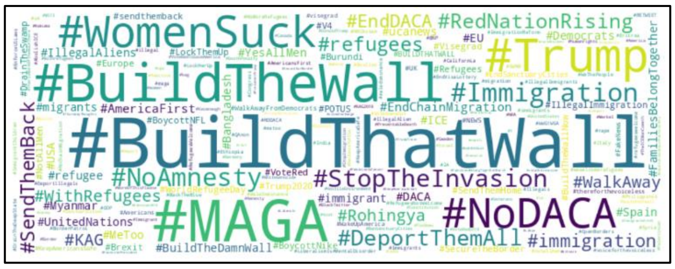


Figure 5

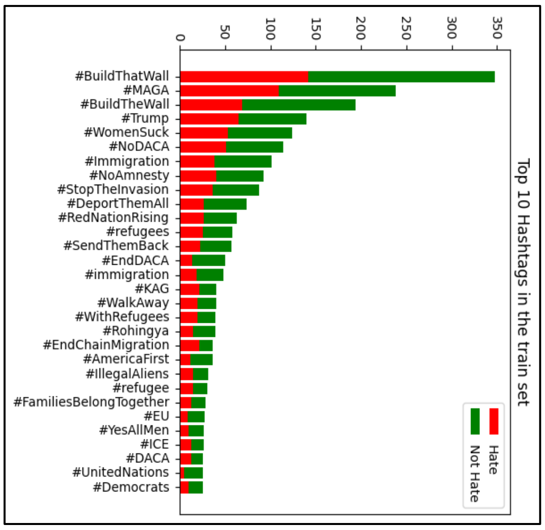


Figure 6

2.4. Train Set After Cleaning

Figure 7 and **Figure 8** show common tokens in the training set, including some stop words such as "you," "your," "they," and "user" as mentions. Therefore, it can be argued that keeping certain stop words is relevant to the context of hate speech targeting individuals or groups. Similarly, converting punctuations like "?" into "question" serves a purpose similar to that of stop words.

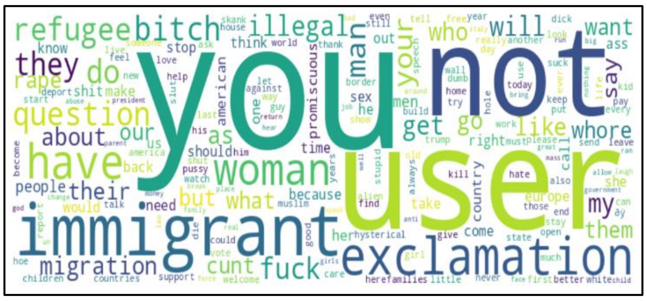


Figure 7

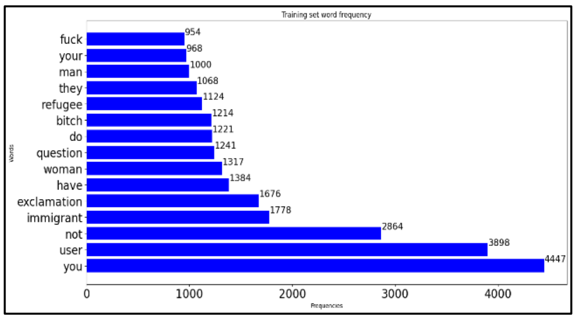


Figure 8

2.5. Existing Issues and Noise in the Validation Set

Figure 9 and **Figure 10** show a smiler issue in the train set was found in the validation set as it has many meaningless data that need to be cleaned.

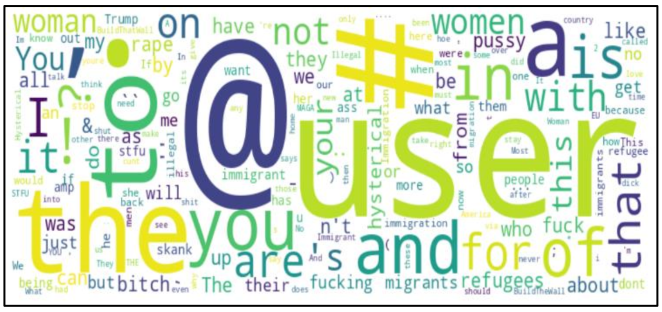


Figure 9

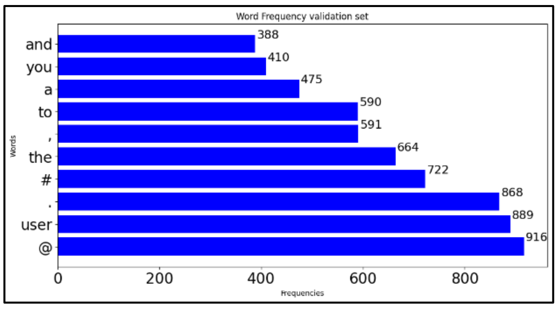


Figure 10

2.6. Validation Set Hashtags Analysis

Figure 11 and **Figure 12** show that the hate and not-hate label distribution among hashtags in train and validation sets is similar, with minor fluctuations. Some hashtags differ between sets, such as "#Rohingya", "#FamiliesBelongTogether", and "#Democrats" in the train set only, and "#POTUS", "#VoteRed", and "#ThursdayThoughts" in the validation set only. Therefore, the train set's hashtag removal suggestion should be applied to the validation set.

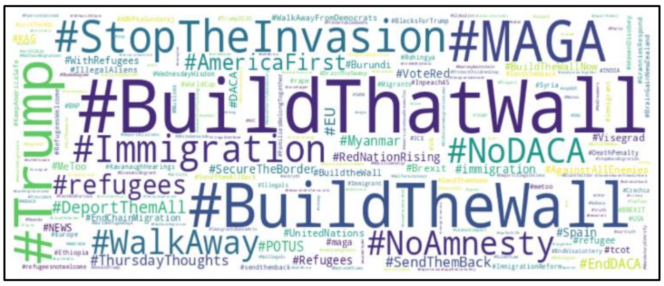


Figure 11

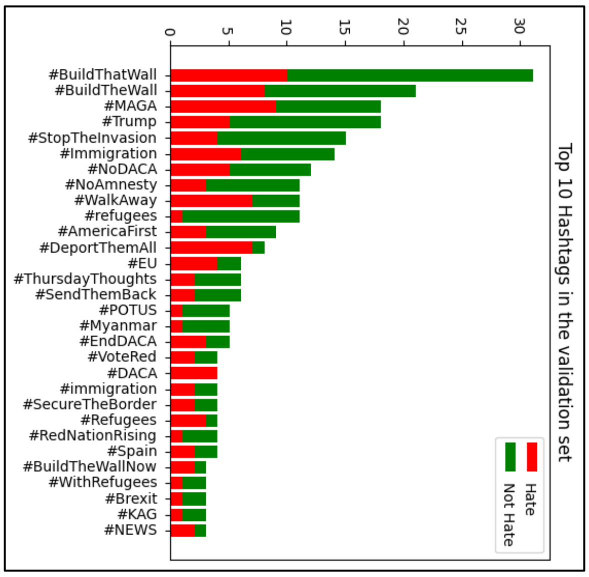


Figure 12

2.7. Validation Set After Cleaning

Figure 13 and **Figure 14** show the most common words in the validation set after removing the noise data with the same process that has been done in the train set.

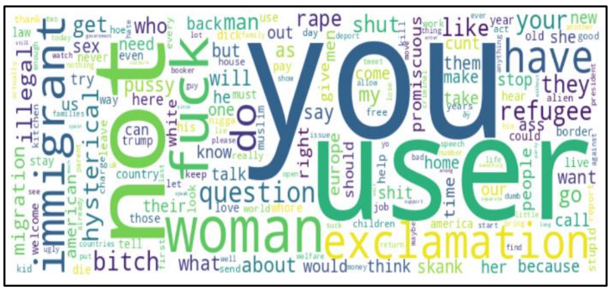


Figure 13

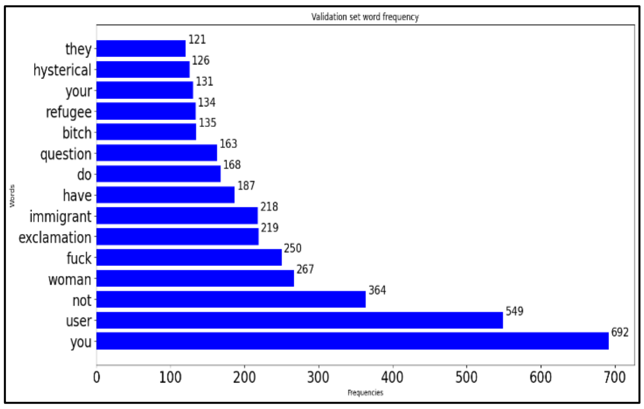


Figure 14

2.8. Existing Issues and Noise in the Test Set

Figure 15 and **Figure 16** show a similar issue in the train set was found in the test set as it has many meaningless data that need to be cleaned.

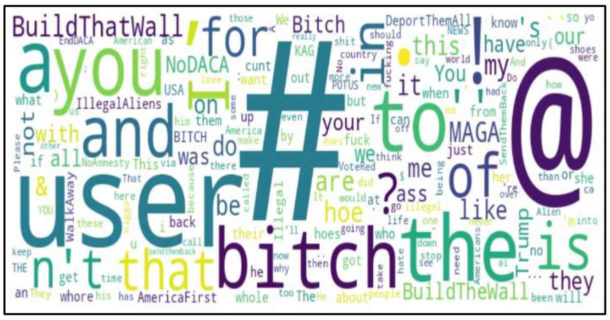


Figure 15

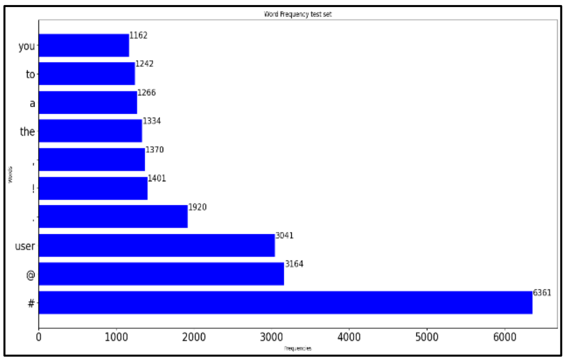


Figure 16

2.9. Test set hashtags analysis

Figure 17 and **Figure 18** show that the distribution of hate and not-hate labels among hashtags in test and train sets is similar, with minor fluctuations. Some hashtags differ between sets, such as "#refugees" and "#WithRefugees" in the train set only, and "#NEWS", "#VoteRed", and "#POTUS" in the test set only. This might potentially impact the model's performance, as the model may consider train set hashtags not present in the test set.

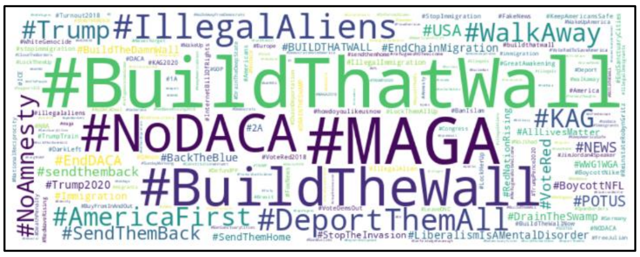


Figure 17

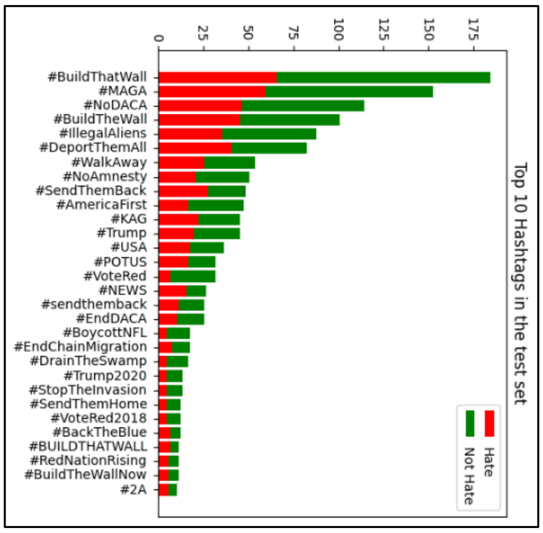


Figure 18

2.10. Test Set After Cleaning

Figute 19 and **Figure 20** show the most common words in the test set after removing the noise data with the same process that has been done in the train set.

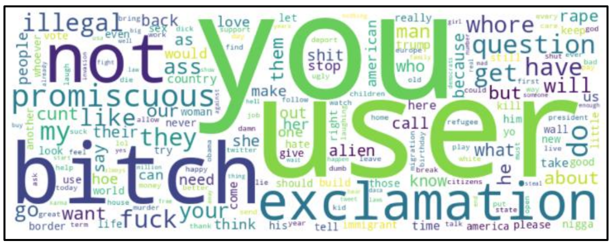


Figure 19

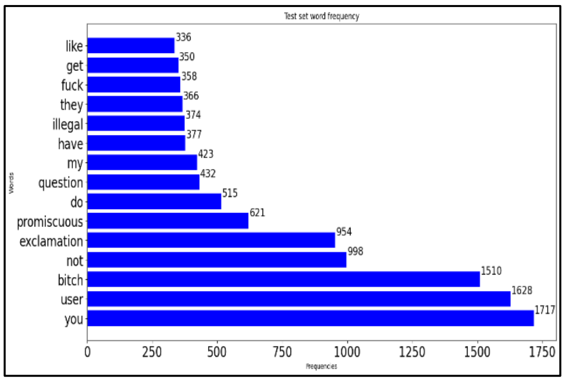


Figure 20

2.11. Hashtags Occurrence Among All Data Sets

The following figures (**Figure 21**, **Figure 22**, and **Figure 23**) show the top three existing hashtags in all three datasets. However, **Figure 23** reveals that the test set has more occurring hashtags than the train set, even though the test set represents about 33% of the train set.

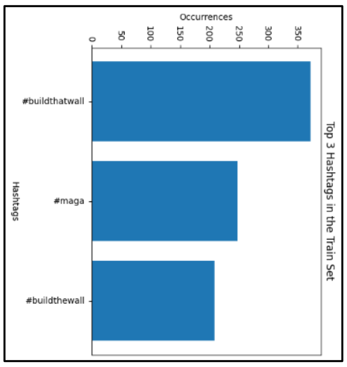


Figure 21

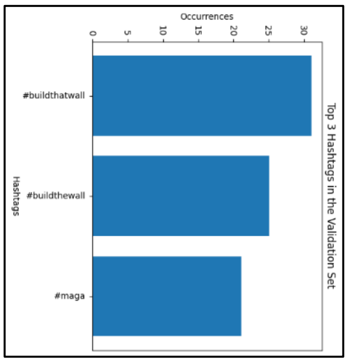


Figure 22

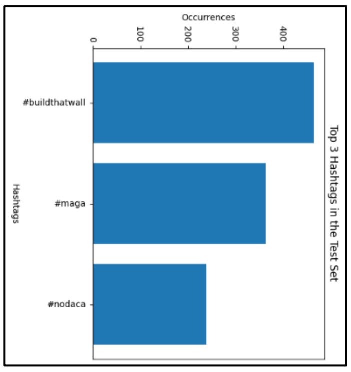


Figure 23

2.12. Description analysis conclusion

In conclusion, all three datasets contain a significant amount of noisy data, including capitalised letters and words, punctuation marks, hashtags, mention symbols, emojis, URLs, and stop words. Furthermore, there are many misspelt and non-base form words. Additionally, it can be concluded that having tweets with offensive hashtags does not always indicate hate speech on Twitter, and the train set has some hashtags that are not in the other datasets. Finally, the test set has more frequent hashtag occurrences than the train set, which could impact the model's performance.

3. Text Processing

A `TextProcessor` class has been developed for text processing. In this process, text data is pre-processed, cleaned, and transformed to remove inconsistencies and noise, to standardise and simplify text, and to prepare it for analysis and modelling. This stage is critical to improve the accuracy and efficiency. Relevant examples are provided below each step as described below.

3.1. Preliminary Processing

In preliminary text processing, exclamation and question marks are replaced with their respective names, and mention signs are removed for better analysis and accuracy. Words starting with e- followed by a dash and URLs are modified for consistency.

```
tp.process("Hey! @user ? e-visa")
>> hey exclamation user question evisa

tp.process("Hey!! @user @user @user ??")
>> hey exclamation user question

tp.process("visit http://example.com")
>> visit
```

3.2. Replacing Contractions

In text processing, replacing contractions with their full forms can improve accuracy and clarity. The Contractions library in Python provides a "fix" function to replace contractions with their full forms.

```
tp.process("can't wouldn't you're")
>> cannot would not you are
```

3.3. Replacing Hashtags

Hashtags are replaced, considering capitalization, by applying a ratio of uppercase to lowercase characters. If the majority of characters are capital letters, we simply return the string without any modifications. Otherwise, we split the hashtag into separate words based on the capital letters.

```
tp.process("#BuildThatWall #MAGA")
>> Build That Wall MAGA
```

3.4. Replacing Digits Within Words

Digits within words are replaced using a dictionary mapping them to letters. Words containing digits are identified with a regular expression (`r"\b[a-zA-Z]\w*\d\w*[a-zA-Z]\b"`) and checked against the dictionary for replacement. For instance, digit '1' mapped to the letter 'i'.

```
tp.process("BU1LD TH4T W4LL N0W")
>> build that wall now
```

3.5. Replacing Emojis

Emojis are replaced with their respective names using the emojis library's mapping. In addition, a list of common emojis is replaced directly from the important_emojis dictionary, bypassing the default dictionary, to name emojis with slang meanings. For example, the red cross emoji is named "no" instead of "cross_mark", ensuring that all emojis are replaced with related words.

```
tp.process("say ❌ refuge 🇬🇧😄")
>> say no refuge united_kingdom grinning_face_with_smiling_eyes
```

3.6. Word Tokenizing

Tokenizing implies dividing text into discrete parts called tokens. Tokenization is conducted after most cleaning steps, utilizing a regex tokenizer with the expression: `(?u)\b\w\w+\b`, which matches sequences of two or more word characters. Tokenization returns a list of words.

```
TOKENIZER = nltk.RegexpTokenizer(r'(?u)\b\w\w+\b')
```

The remaining steps in our text processing pipeline are token-based, which means that they work with the individual tokens produced by the tokenizer.

3.7. Removing Stopwords

Stopwords such as "the", "and", "a", and "in" are frequently used but do not carry significant meaning to the model. Removing them can reduce noise in the data and improve model accuracy.

```
nltk.corpus.stopwords.words('english')
```

However, a list of words has been created to be kept during stopwords removal, as some stopwords are used to target individuals (e.g. against, who, he, she).

```
tp.process("men are the causes of all problems with it")
>> men causes problems
```

3.8. Replacing Abbreviations

To handle the informal language and jargon often found in tweets, a dictionary of abbreviations was created by analysing the training dataset. During processing, each token in the text is checked against this dictionary, and if a match is found, the abbreviation is replaced with its corresponding full form. Having abbreviation pattern like "gr8" is not replaced in the digit replacement step because the digit "8" is not in the digits dictionary, but it will be replaced in the abbreviation replacement step.

```
tp.process("bf u met gr8")
>> boy-friend you met great
```

3.9. Replacing Important Words and Similarity Check

This step involves using a list of important words and slangs obtained from a training dataset, and eliminating repeated characters in each token using regex such as “refugeeees” will be refugees. Next, the `difflib.SequenceMatcher` library is used to calculate similarity scores between a given token and words in an important words list. If the maximum similarity score is greater than a certain threshold (e.g. 0.8), the token is replaced with the corresponding important word.

```
tp.process("send back all refuuuggeeeees to their countrryyyyy!!")
>> send back all refugee to their country exclamation
```

3.10. Lemmatizing

This step involves reducing inflected words to their base or dictionary form, which helps to standardise the text to improve model performance.

```
tp.process("building walls")
>> build wall
```

However, the TextProcessor class has arguments to enable or disable text processing functions, which can help compare the accuracy of different combinations of processing steps and select the best parameters. Additionally, the TextProcessor has default dictionaries and lists for processing steps such as important words, emojis, and abbreviations, which can be easily initialised with custom values for testing and experimentation. This allows for easy testing of different processing step combinations and parameters to optimise model performance.

4. CNN Implementation For Hate Speech Detection

4.1. CNN usage and justification

CNN (Convolutional neural network) is a type of deep learning algorithm which has shown to be effective in image and text classification tasks. It is particularly useful in tasks where there are local patterns that need to be identified in data, such as NLP (natural language processing) tasks like sentiment analysis and hate speech detection.

4.2. Word embedding

Word embeddings capture semantic and syntactic relationships among words, allowing for better generalisation and similarity comparison among words. Words that are similar in meaning like ‘king’ and ‘queen’ are likely to have similar word embeddings while words with dissimilar meaning like ‘hate’ and ‘cheer’’ are likely to have very different embeddings.

4.3. Combining word embedding with CNN

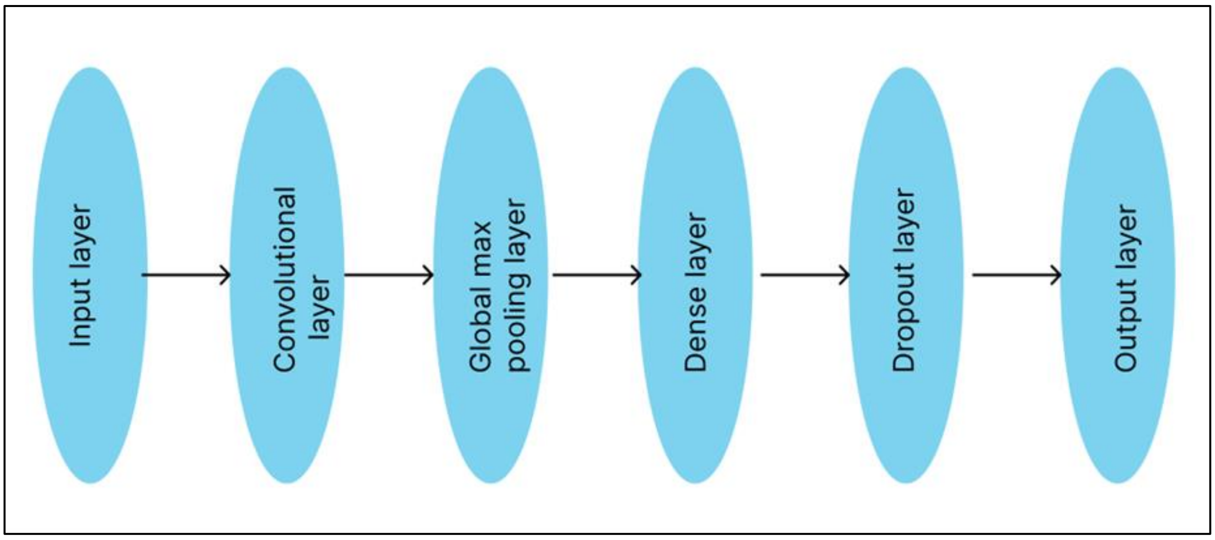
Combining word embedding with CNN is an approach that can improve both efficiency and accuracy of the model. It offers various advantages such as:

capturing semantic meaning: CNN models can leverage the word embeddings feature of input text which capture the semantic meaning of words and phrases and provide more accurate and meaningful representation of the words.

Efficient use of data: Word embeddings can reduce the dimensionality of the text data and provide a more efficient representation of the input text. This can help to reduce the amount of data needed to train and improve the performance of the model.

Capturing local and global context: CNN models are designed to capture the local patterns in the input data while word embeddings capture the global context of the input text. Combining both with lead generates a model which captures both contexts.

4.4. CNN Implementation detail



Input layer: The input layer takes in the text data in the form of sequence of word embeddings. The input shape is specified as (max_sequence_length, embedding_dimensions) where ‘max_sequence_length’ is the length of the input sequence whereas ‘embedding_dimensions’ is the dimensionality of the word embeddings.

Convolutional layer: This layer applies a set of filters on the input data to extract various features. The number of filters and kernel size in this layer are specified as hyperparameters which can be tuned to maximise the information gain. The activation function used in this layer is ‘ReLU’.

Global max pooling layer: This layer takes the maximum value from each feature map of the previous layer, reducing the dimensionality of the data and retaining the most significant information.

Dense layer: This is a fully connected layer that applies learnable weights to generate a new set of features from the output of the previous pooling layer, with the number of units that specified as a hyperparameter, and uses ReLU as the activation function.

Dropout layer: The dropout layer randomly drops 40% of units in the dense layer during training to prevent overfitting by reducing the interdependence of neurons. The percentage of units to be dropped is a specified hyperparameter.

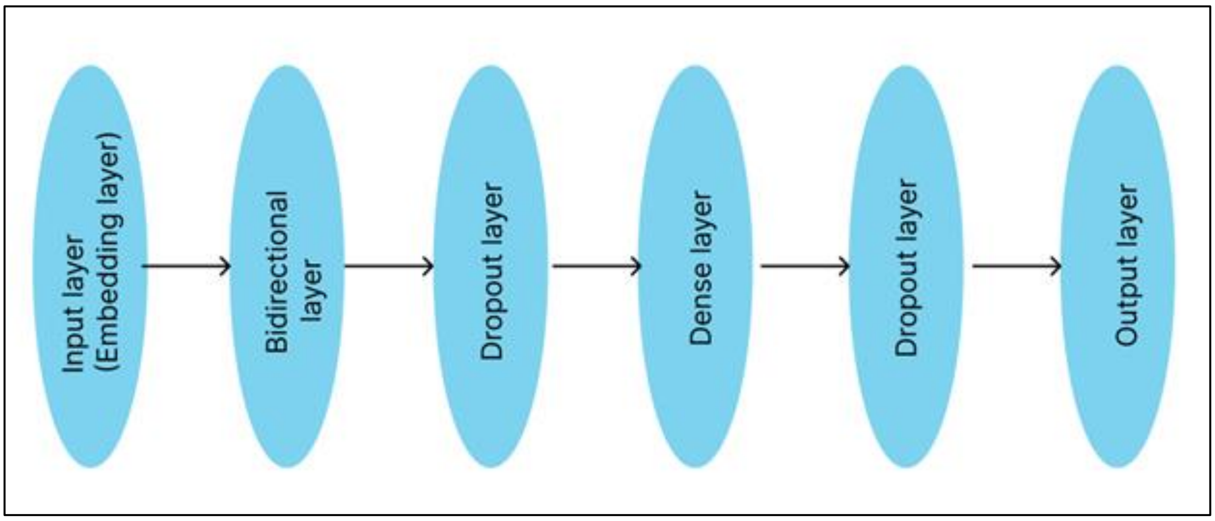
Output layer: The output layer has two units for the binary classification task, ‘hate’ or ‘non-hate’ speech, with softmax as the activation function, guaranteeing that the output probabilities sum to 1.

5. LSTM Implementation for Hate Speech Detection

5.1. LSTM usage and justification

Long short-term memory (LSTM) model was created to improve the existing RNN by resolving its (deficiencies/flaws). LSTM has a feedback connection system which helps it retain previous information while processing a sequence of data. This makes it an ideal choice for when the data is in a sequential format, e.g. Text data. Bidirectional neural networks are used when we want to incorporate information from both past and future contexts in a sequence.

5.2. LSTM Implementation detail



Embedding layer: The embedding layer converts tokenized text to 45-dimensional vectors using the encoder's vocabulary and supports masking with (mask_zero=True) to ignore padding tokens (represented as zeros) and prevent their impact on the model's learning process.

Bidirectional layer: This layer employs a bidirectional LSTM with 60 hidden units, allowing the model to learn from both past and future context in the input sequence, helping it understand and learn from the input sequence more effectively. This helps the model to better understand the underlying patterns and relationships in the text data. The two LSTM's hidden states are then combined before being passed to the next layer.

Dropout layer (1st instance): The first dropout layer is applied with a rate of 0.4, which helps in reducing overfitting by randomly dropping 40% of the input units during training. This forces the model to learn more robust features and prevents it from relying too much on any single input unit.

Dense layer: A fully connected layer with 128 neurons is used, employing the ReLU activation function and L1/L2 regularization to control model complexity and prevent overfitting by penalizing large weight values.

Dropout layer (2nd instance): The second dropout layer, with a rate of 0.4, helps prevent overfitting during training, promoting a more generalized understanding of the input data.

Output layer: The last layer is a dense layer with a sigmoid activation function, L1 and L2 regularization, and a single neuron that outputs the probability of the target class, allowing the model to output the probability of input text being hate speech or not.

6. Comparison Between CNN and LSTM

LSTM and CNN are both types of deep learning models used in various applications such as NLP, image recognition and time-series analysis. LSTM is a type of recurrent neural network whereas CNN is a type of feedforward neural network that is mainly used for image and signal processing.

LSTM is specifically designed to process sequential data by allowing information to flow through memory blocks and a sequence of cells which helps the model dependencies and long-patterns in sequential data. On the other hand, CNN is designed to extract features from a fixed-size input such as images, by applying convolutional filters to the input and pooling the output features.

Feature	CNN	LSTM
Architecture	Convolutional layers	Recurrent layers with memory cells
Primary Focus	Spatial feature extraction	Sequential data and long-range dependencies
Handling Sequential Data	• Limited receptive field. • struggles with long-range dependencies.	• Designed for sequential data. • captures long-range dependencies effectively
Context Learning	Spatial hierarchies of features	Both past and future context in input sequence
Suitability for Text Data	Less effective for capturing context and word relationships	Better at understanding context and relationships in text data
Training Time and Resources	Generally faster and less	May require more training time and resources due to recurrent nature
Use Cases	• Image classification. • Spatial feature detection.	• Text classification. • Language modeling. • Translation.

While both LSTM and CNN have their own strengths and weaknesses, here are some advantages of LSTM over CNN:

Handling Sequential Data: LSTM is ideal for sequential data, like time-series data, natural language text, and audio signals. It can capture dependencies between events occurring at different times, making it essential for tasks like speech recognition, language translation, and sentiment analysis

Memory Retention: LSTM retains memory over long periods, which is useful for tasks requiring the model to remember previous parts of the sequence, whereas CNN processes each input independently and cannot retain memory for long durations.

6.1. Final verdict

For accurate predictions where the order of input data and long-range dependencies are crucial, an LSTM-based model is preferred for text classification tasks. When classifying hate speech or non-hate speech, the bidirectional LSTM is suitable as it can understand the context and relationships between words in a sequence, capturing both past and future contexts, making it more suitable than a CNN.

7. Hyperparameter Tuning (Experimental setting)

7.1. Hyperparameters Meaning

Hyperparameters are parameters that must be set before training a machine learning model, as they cannot be learned from the training data. Optimizing these hyperparameters can enhance our model's performance. These hyperparameters play a crucial role in controlling the learning process and the model's behaviour, significantly impacting the overall performance.

7.2. Hyperparameters in CNN Model

In the CNN model, several parameters can be hyper-tuned, such as number of filters, kernel size in the convolutional layer, number of dense units in the dense layer, and the dropout rate in the dropout layer.

7.3. Hyperparameters in LSTM Model

In LSTM model, several parameters can be fine-tuned, including: the size of the embedding output dimensions in the input layer, number of LSTM units in the bidirectional layer, number of dense units in the first dense layer, the dropout rate in both dropout layers, the L1 and L2 regularisation rates in the dense layers, and the learning rate of the model.

7.4. Importance of hyperparameters tuning

The number of layers and nodes in a model significantly impacts its performance by affecting the ability to learn from input data. Adding more layers enables the model to learn complex and abstract features, while having more nodes per layer capture more fine-grained details. However, more layers and nodes can result in overfitting, where the model memorizes training data instead of generalizing to new data. Therefore, optimizing the combination of layers and nodes is crucial to balance the model's learning of complex features and prevent overfitting.

7.5. Optimising hyperparameters:

Keras Tuner offers various tuners, including RandomSearch, Bayesian Optimization, and Hyperband. In this case, the **RandomSearch** tuner was used to randomly select hyperparameter combinations from a specified range and produce the best results.

An experimental setting was performed to identify the optimal model configuration via hyperparameter tuning. The tuner was run for a maximum of 5 trials, with 3 executions performed under each trial, in order to obtain the best values. The aim was to find the best model and hyperparameters for the highest validation accuracy.

8. Results

After deriving the supposedly best set of hyperparameters from the hyperparameter tuning, another model was created (experimental model) to test its performance. Both models were tested, i.e. the initial LSTM model and the hypertuned experimental LSTM model, on the test dataset and following are the results that obtained.

Comparing test dataset accuracy of LSTM model and experimental model

Initial model results
Test Loss: 1.0005418062210083
Test Accuracy: 0.6230613589286804
Test MSE: 0.23535001277923584
Test MAE: 0.4797063171863556
Test Precision: 0.5430038571357727
Test Recall: 0.6757188439369202

Experimental model results
Test Loss: 1.2144875526428223
Test Accuracy: 0.5448415279388428
Test MSE: 0.35238587856292725
Test MAE: 0.45002681016921997
Test Precision: 0.47744014859199524
Test Recall: 0.8282747864723206

Comparing performance of the LSTM model on all the datasets

Metric	Training set	Validation set	Test set
Loss	1.0959	0.9593	1.0005
Accuracy	0.7259	0.6720	0.6231
MSE	0.2019	0.2159	0.2354
MAE	0.4316	0.4490	0.4797
Precision	0.7888	0.6552	0.5430
Recall	0.4759	0.4895	0.6757

Comparing performance of the experimental model on all the datasets

Metric	Training set	Validation set	Test set
Loss	0.2559	0.9905	1.2145
Accuracy	0.9050	0.6640	0.5448
MSE	0.0718	0.2614	0.3524
MAE	0.1505	0.3540	0.4500
Precision	0.8665	0.5938	0.4774
Recall	0.9151	0.6745	0.8283

8.1. Results Justification:

To solve this task, a generalized model is needed that is capable of working on data collected from various groups, such as tweets posted by people from different ethnicities and locations**.** In our case, the test set's accuracy is dropping significantly since the test set's tweets contain a completely different style, with predominant African-American slangs that are not present in our training and validation datasets. Despite working with the experimental model, which achieved approximately 90% accuracy on the training set and 66% on the validation set, the accuracy kept declining, and a final accuracy of only 54% was obtained on the test set.

8.2. Proposed solution

Two proposed solutions to overcome the accuracy drop issue in the test set are:

1- Mixing all datasets (train, test, and validation) and dividing them into different sets. This approach ensures that all resulting datasets have a mix of all types of tweets, and no dataset represents a completely different style of tweets (**see appendix**).

2- Developing a generalized model that can work on all types and styles of tweets and is capable of decoding all slangs and vernacular languages.

9. Error Analysis

9.1. CNN

Test set accuracy is 46.8%, indicating a significant classification errors. True label 0 data had 551 correct predictions, but 1163 were incorrectly predicted as 1. For true label 1 data, 203 were correctly predicted as 1, but 1049 were incorrectly predicted as 0, indicating an imbalanced false negative to false positive ratio (**Figure 24**).

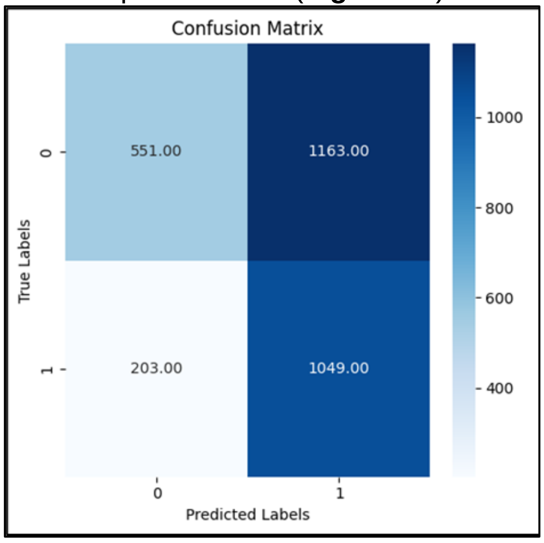
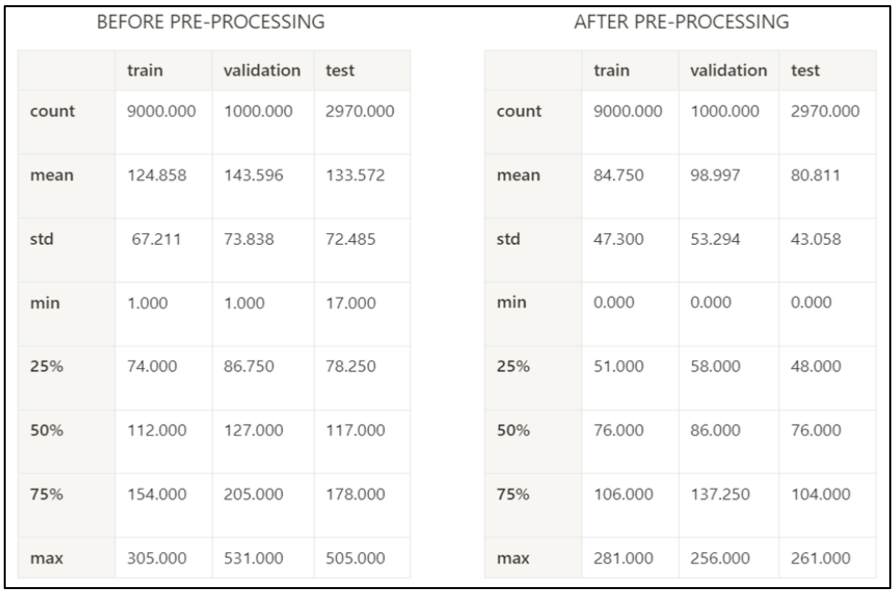


Figure 24

The model's performance highlights two key issues: relatively low accuracy and a marked imbalance in false negatives to false positives. Upon analysing misclassified data and model performance, the following factors may account for these issues:

1. Dataset length analysis showed that 25% of the data exceeded 100, while the model's max_sequence_length is set to 100, resulting in At least 25% of the data is at risk of losing some information. In the validation set, at least 25% of the data has suffered from a loss of over 37% of its information due to this limitation**.**

Figure 25 shows the length distribution of the data before and after pre-processing**.**



BEFORE PRE-PROCESSING

	train	validation	test
count	9000.000	1000.000	2970.000
mean	124.858	143.596	133.572
std	67.211	73.838	72.485
min	1.000	1.000	17.000
25%	74.000	86.750	78.250
50%	112.000	127.000	117.000
75%	154.000	205.000	178.000
max	305.000	531.000	505.000

AFTER PRE-PROCESSING

	train	validation	test
count	9000.000	1000.000	2970.000
mean	84.750	98.997	80.811
std	47.300	53.294	43.058
min	0.000	0.000	0.000
25%	51.000	58.000	48.000
50%	76.000	86.000	76.000
75%	106.000	137.250	104.000
max	281.000	256.000	261.000

Figure 25

2. Larger kernel_size needed in our CNN model to capture longer adjacent information from the dataset, as different kernel_sizes extract different sizes of features. Smaller kernel_sizes better for local features and larger kernel_sizes better for global features.

3. Setting the vocab_size to 1000 in the CNN model results in significant information loss since the dataset's actual vocabulary size is 12,516 words. Increasing vocabulary size should be considered without setting it too large to avoid overfitting and excessive training time.

4. Having imbalanced dataset, with hate speech (label=1) and non-hate speech (label=0) in proportions of approximately 58% and 42% respectively, may affect the model's ability to distinguish between the two categories. This issue, can be resolved by using compute_class_weight from sklearn.utils.class_weight to redistribute the weights of each data point.

5. The epoch is currently set to 5 , but the loss has not converged after 5 epochs, as shown in **Figure 26.** Increasing the number of epochs until convergence is necessary for the model to learn more effectively.

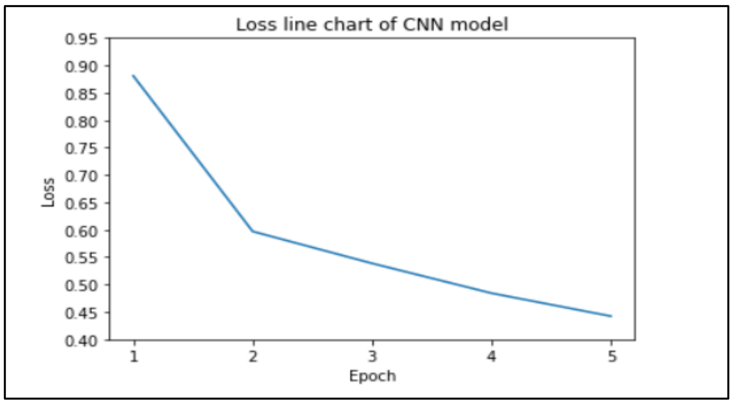


Figure 26

9.2. LSTM

The LSTM model's test set accuracy is slightly better than the CNN model at 56.7%. However, **Figure 27** shows the model has noticeable classification errors with 759 instances of actual label 0 data being predicted as 1, and 861 instances of actual label 1 data being predicted as 0, resulting in a chance for further improvement in the model's ability to accurately classify the data.

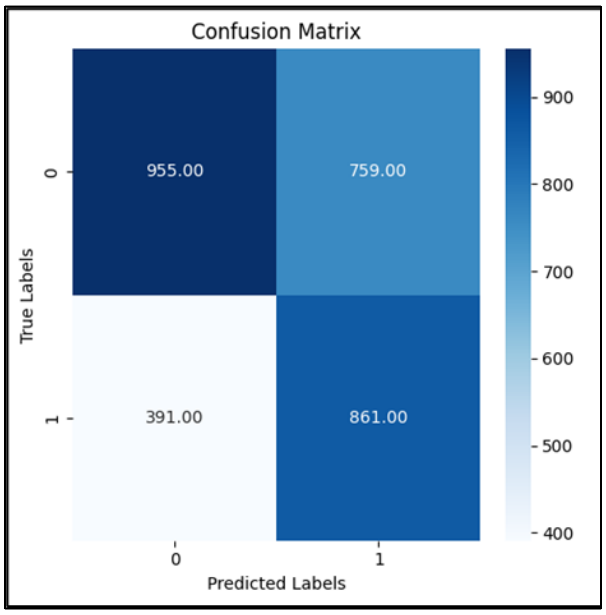


Figure 27

1. In LSTM, each word is mapped to a unique vector representation by indexing it in the vocabulary. Therefore, setting the vocab_size to the vocabulary size in LSTM will not result in overfitting. It is suggested to set the vocab_size to 12,516.

2. This issue can be resolved by using compute_class_weight from sklearn.utils.class_weight to redistribute the weights of each data point, similar to the fourth point in CNN.

3. The code sets early_stop with a patience value of 2. After testing, the model meets the dropout condition after about 15 epochs, but if the epoch is set to 7, the learning process of the model does not end. **Figure 28** shows this.

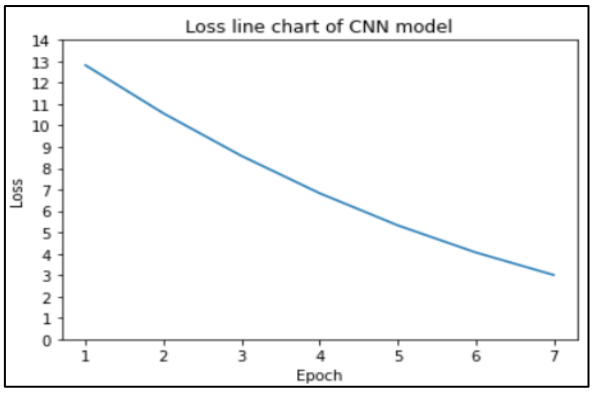
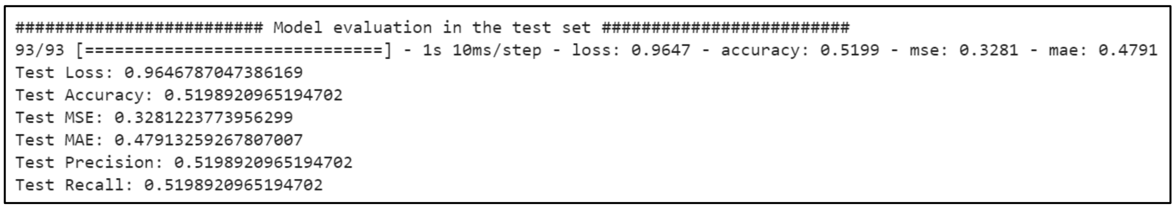


Figure 28

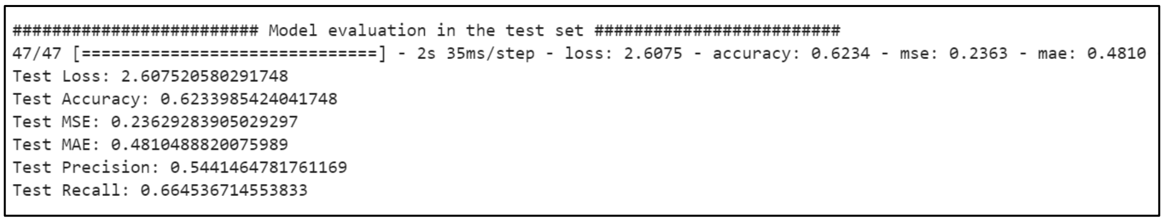
9.3. The Results After Improve

CNN model accuracy increased from 46.8% to 51.99%, and LSTM model accuracy in the test set increased from 56.7% to 62.34%, as shown in **Figure 29** and **Figure 30** respectively.



```
######################### Model evaluation in the test set #########################
93/93 [==============================] - 1s 10ms/step - loss: 0.9647 - accuracy: 0.5199 - mse: 0.3281 - mae: 0.4791
Test Loss: 0.9646787047386169
Test Accuracy: 0.5198920965194702
Test MSE: 0.3281223773956299
Test MAE: 0.47913259267807007
Test Precision: 0.5198920965194702
Test Recall: 0.5198920965194702
```

Figure 29



```
######################### Model evaluation in the test set #########################
47/47 [==============================] - 2s 35ms/step - loss: 2.6075 - accuracy: 0.6234 - mse: 0.2363 - mae: 0.4810
Test Loss: 2.607520580291748
Test Accuracy: 0.6233985424041748
Test MSE: 0.23629283905029297
Test MAE: 0.4810488820075989
Test Precision: 0.5441464781761169
Test Recall: 0.664536714553833
```

Figure 30

According to the experimental results, the mentioned parameter adjustments led to improved accuracy for both models. Based on section 2-11 revealed that each dataset contains different examples, indicating the issue of same labels, different features (non-iid data). This can lead to variations between the data in the training, validation, and test sets, making it challenging for the model to generalize to new data in the test set.

10. Literature Review

In a study by Gao and Huang (2017), two models were proposed for hate speech detection. The first model is a logistic regression model that uses n-gram features, and the second model is a neural network based on LSTM architecture with three parallel LSTM layers. The LSTM model uses binary cross-entropy as the loss function and employs dropout techniques to enhance performance and prevent overfitting.

Winter and Kern (2019) explored hate speech detection in English and Spanish texts using logistic regression, SVM classifiers with n-gram-based TF-IDF as features extraction, and a convolutional neural network (CNN) model with five filter types and using a skipping words technique. The authors found that the CNN model performed slightly better than the traditional approaches.

In a study by Thi-Thuy Do et al. (2019), the authors explored the use of Bidirectional Long Short-Term Memory (Bi-LSTM) for Vietnamese hate speech detection. They employed word embeddings to train their model. The results showed that combining Bi-LSTM with Fasttext led to better accuracy.

Naseem et al. (2020) conducted a study on tweet classification, exploring twelve pre-processing techniques, such as removing URLs, emojis, and abbreviations, correcting spelling, and word segmentation. They assessed these techniques using three labelled datasets containing Twitter hate speech and abusive language, employing traditional and deep learning algorithms with various feature extraction models. The research identified the most effective pre-processing methods for optimal outcomes.

AbdelHamid et al. (2022) compared machine learning and deep learning models, including SVM, Random Forest, MLP, AraBert, GigaBERT, and Soft Voting classifier, for Levantine hate speech detection. GigaBERT achieved the highest F1-score, with significant improvements seen in classification performance when using dataset augmentation.

Badjatiya et al. (2017) used deep neural network architectures for hate speech detection and compared different methods, including CNN, LSTM, and FastText embeddings, and found that deep neural networks outperformed existing methods. The optimal approach was "LSTM + Random Embedding + GBDT," effectively capturing hatred towards target words using learned embeddings.

11. Conclusion and Future work

In conclusion, when dealing with text classification tasks where order and long-range dependencies are crucial, LSTM-based models are recommended. For hate speech detection, the bidirectional LSTM was found suitable as it captures the context and relationships between words in a sequence. Hyperparameters tuned to optimise the model performance as play a crucial role in model performance.

When evaluating the CNN and LSTM models' performance, both have noticeable classification errors, due to the imbalanced datasets in particular and limited vocabularies. Although increasing the kernel sizes and vocabulary size can improve the model's accuracy on the training set, it can also lead to overfitting and a lack of generalization to the validation and test sets, as the model becomes too adapted to the training set.

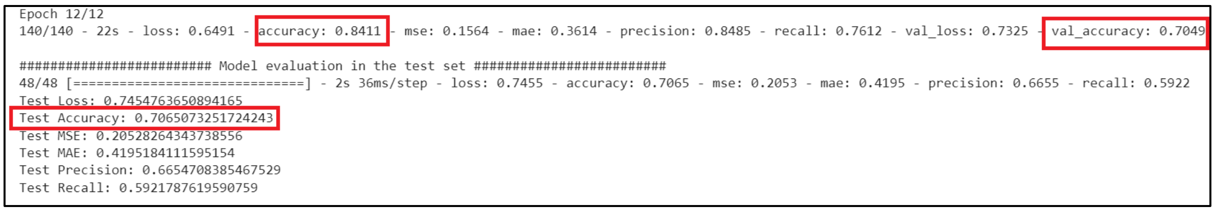
Overall, due to hyperparameters tunning and error analysis, the CNN model's accuracy increased from 46.8% to 51.99%, and the LSTM model's accuracy increased from 56.7% to 62.34%.

In terms of future work, the issue of non-iid data should be resolved by exploring techniques such as data augmentation and having more data. Therefore, increasing the dataset size and diversity could be beneficial. This can be accomplished by expanding more datasets to include tweets from different groups of people, locations, and hashtags as this would increase the model's ability to generalise and perform well on new data. Additionally, including tweets with different slangs and languages can help the model to be more robust and inclusive.

12. Appendix

Based on the suggestion to combine all three datasets (train, test, and validation) and then divide them into separate sets for training, testing, and validation, this approach was implemented using the LSTM model, which can be found at the end of the Google Colab document (under appendix title). As shown in the figure below, the LSTM model performed better on all three datasets without any exhaustive hyperparameter tuning. This led to the following:

- The accuracy in the train set was Increased from 72% to about 84%.
- The accuracy in the validation set was Increased from 67% to about 70.5%.
- Importantly, the accuracy in the test set was Increased from 62% to about 70.6%.



```
Epoch 12/12
140/140 - 22s - loss: 0.6491 - accuracy: 0.8411 - mse: 0.1564 - mae: 0.3614 - precision: 0.8485 - recall: 0.7612 - val_loss: 0.7325 - val_accuracy: 0.7049

######################### Model evaluation in the test set #########################
48/48 [==============================] - 2s 36ms/step - loss: 0.7455 - accuracy: 0.7065 - mse: 0.2053 - mae: 0.4195 - precision: 0.6655 - recall: 0.5922
Test Loss: 0.7454763650894165
Test Accuracy: 0.7065073251724243
Test MSE: 0.20528264343738556
Test MAE: 0.4195184111595154
Test Precision: 0.6654708385467529
Test Recall: 0.5921787619590759
```

In case something went wrong with code:

For any reason if the code did run, this is a shared link for the full code in google colab: https://colab.research.google.com/drive/12QgBGOK1Op-gWDrfsQsH_jtkdDsem0Y0?usp=sharing

13. References

AbdelHamid, Medyan, et al. *Levantine hate speech detection in twitter*. Social Network Analysis and Mining, 2022. *https://link.springer.com*, https://doi.org/10.1007/s13278-022-00950-4.

Badjatiya, et al. *Deep Learning for Hate Speech Detection in Tweets*. International World Wide Web Conferences Steering Committee, 2017. *dl.acm.org*, https://doi.org/10.1145/3041021.3054223.

Gao, Lei, and Ruihong Huang. *Detecting Online Hate Speech Using Context Aware Models*. 2 ed., RANLP, 2017. *arxiv.org*, https://doi.org/10.48550/arXiv.1710.07395.

Naseem, Usman, et al. *A survey of pre-processing techniques to improve short-text quality: a case study on hate speech detection on twitter*. Multimedia Tools and Applications, 2020. *link.springer.com*, https://doi.org/10.1007/s11042-020-10082-6.

Thi-Thuy Do, Hang, et al. *Hate Speech Detection on Vietnamese Social Media Text using the Bidirectional-LSTM Model*. VLSP Workshop, 2019. *arxiv.org*, https://arxiv.org/abs/1911.03648.

Winter, Kevin, and Roman Kern. *Know-Center at SemEval-2019 Task 5: Multilingual Hate Speech Detection on Twitter using CNNs*. vol. Proceedings of the 13th International Workshop on Semantic Evaluation, Association for Computational Linguistics, 2019. *aclanthology.org*, https://aclanthology.org/S19-2076/.

Basile, V., Bosco, C., Fersini, E., Nozza, D., Patti, V., Rangel Pardo, F. M., Rosso, P., & Sanguinetti, M. (2019) 'SemEval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter', in Proceedings of the 13th International Workshop on Semantic Evaluation, Minneapolis, Minnesota, USA. Available at: https://github.com/cardiffnlp/tweeteval/tree/main/datasets/hate [Accessed: 20 April 2023].

Do, H.T.T., Huynh, H.D., Van Nguyen, K., Nguyen, N.L.T. and Nguyen, A.G.T., 2019. Hate speech detection on vietnamese social media text using the bidirectional-lstm model. *arXiv preprint arXiv:1911.03648*.

Van Houdt, Greg & Mosquera, Carlos & Nápoles, Gonzalo. (2020). A Review on the Long Short-Term Memory Model. Artificial Intelligence Review. 53. 10.1007/s10462-020-09838-1.