Supervisor, Dimosthenis Antypas

<u>Group leader</u>

Hossein Ramezanian 1667866

<u>Memebers</u>

Abdullah Barayan, 2012898

Abdullah Alshatti, 21105464

Hassan Khan 1866602

Aishwarya Paode 2087690

CMT - 316

Group 7

Introduction

This report is a technical and practical discussion for solving a hate speech classification problem by implementing modern machine learning techniques. The dataset used for this project is in [1], under the hate repository which is a text classification problem.

Dataset and Descriptive analysis

This sub-section aims to deliver a clear and concise discussion and analysis of the provided dataset. To begin, the dataset utilized for this project is "SemEval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter", a twitter based text file derived from tweets [1]. The project aims to facilitate the data via modern machine learning techniques to generate classifiers that recognize hate speech in given text. The data was collected using tweets posted from July 2018 to September 2018. The information is given in 6 files for three different sets (Training set, Testing set and Evaluation set); each data set has two files, a textual file and a label-identifier. The textual file consists of one tweet per line, while in the label file each line is the labelling of the corresponding tweet, 0 is used to label non-hate speech tweets and 1 is used to label tweets that contain hate speech. For each data set, the corresponding two files were integrated and converted to a CSV file for easier processing such that it could be used with panda package data frames.

We started our dataset analysis by observing the number of records in all the datasets. Following that, we analyzed the length of the given tweets in the training dataset. Upon this brief observation, it was found that most of the tweets lie under 300 words, as is evident in the given graph in the file containing the code. During the analysis, we also checked whether there is an imbalance in the dataset and found that our training dataset has around a 6:4 ratio of non-hate tweets to hate tweets.

Statistics of dataset splits

Training set:

The total number of data entries in the training set is 9000. There are 5217 entries labelled as non-hateful and 3783 entries labelled as hate speech. This constitutes almost 58% non-hateful to 42% hate speech, as shown in Fig 1.

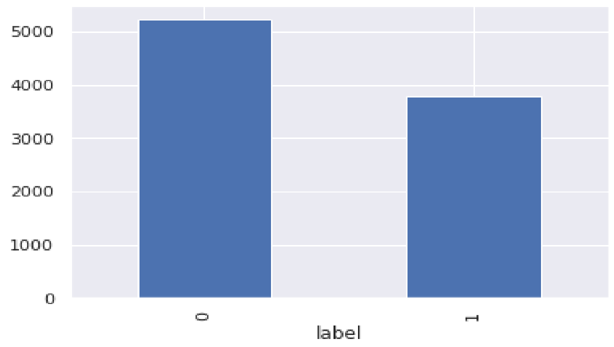
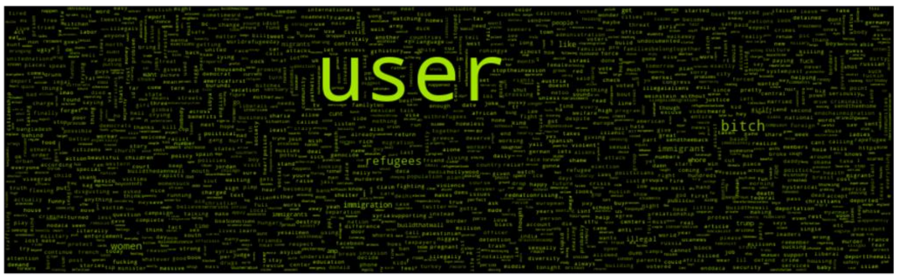


Figure 1

Word cloud for the training set:

The word "User" is the most common word in the dataset text, this due to the fact that this word replaces the user names mentioned in the original tweets.



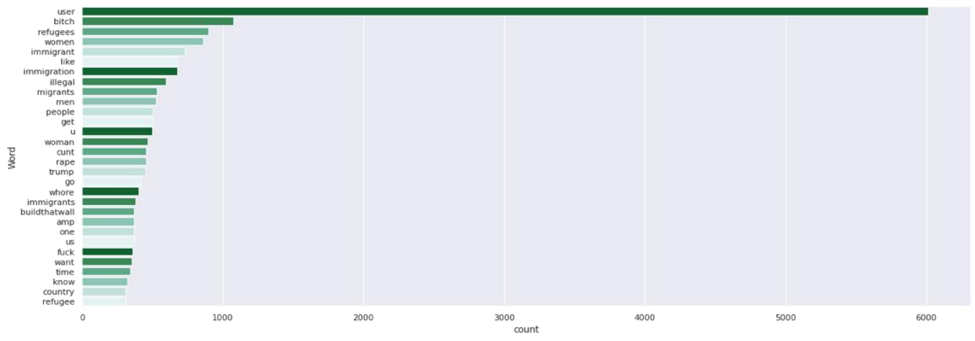


Figure 2

After processing and cleaning the data we get the following overall word cloud of the training set for both hate and non-hate classes.

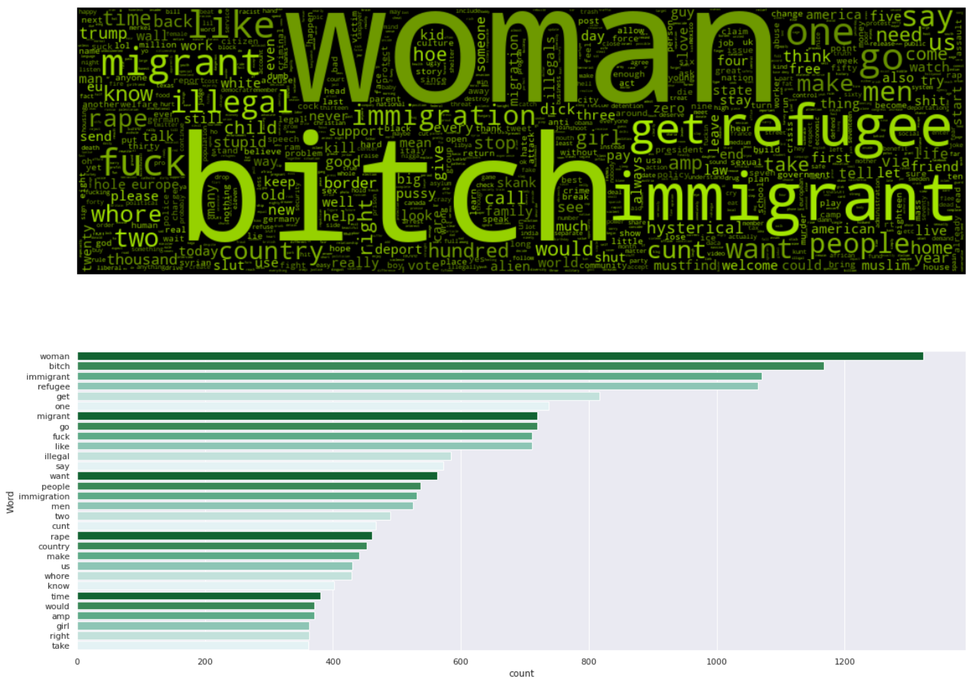


Figure 3

Word cloud for non-hate speech text in the training set:

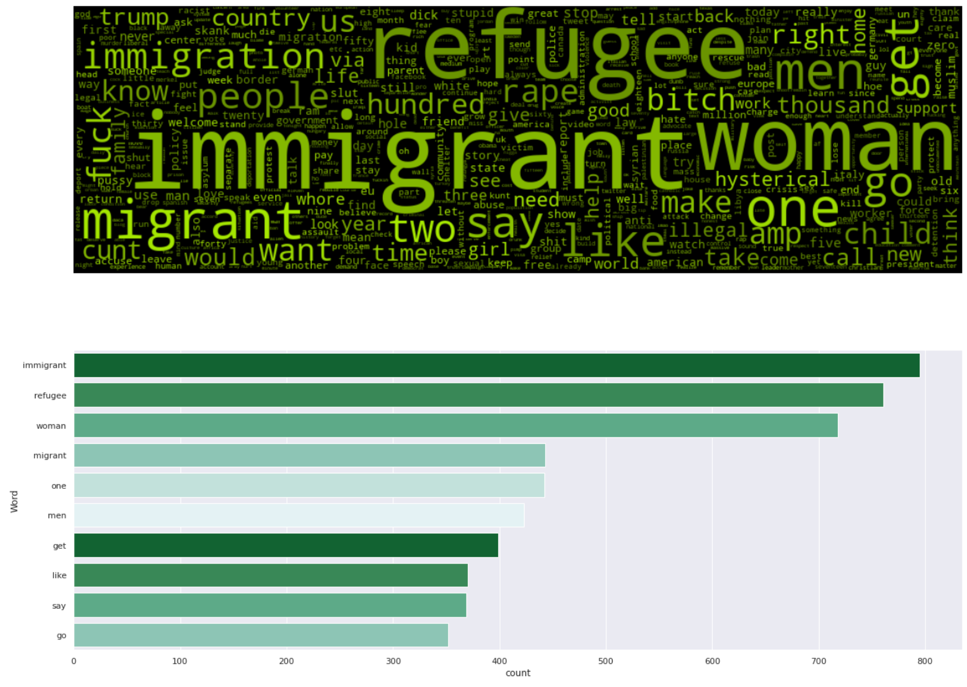
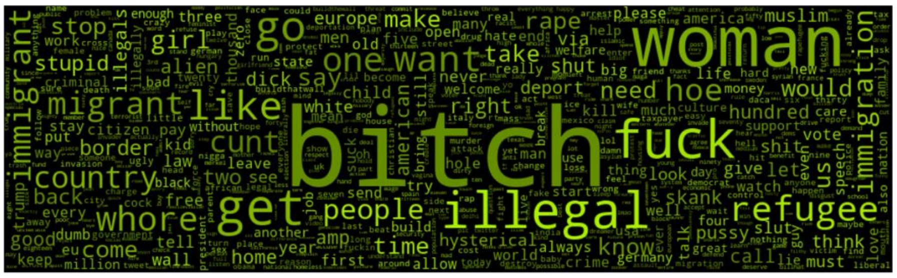


Figure 4

Word cloud for hate speech text in the training set:



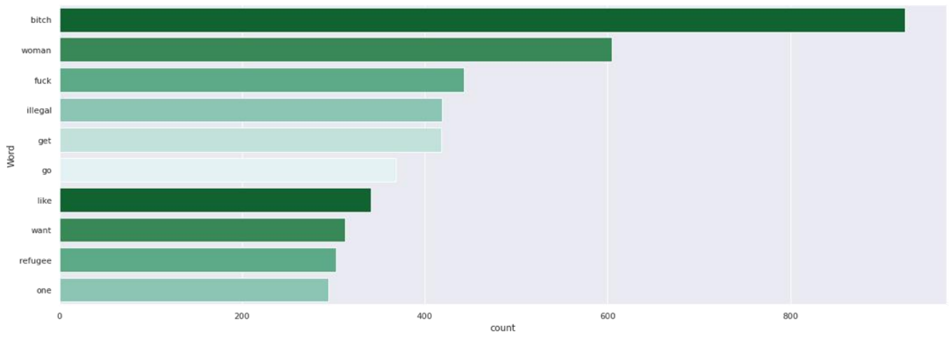


Figure 5

Test set:

The total number of data entries in the test set is 3000. There are 1718 entries labelled as non-hateful and 1252 entries labelled as hate speech. This constitutes almost 57% as non-hateful to 43% hate speech, as shown in Fig 6.

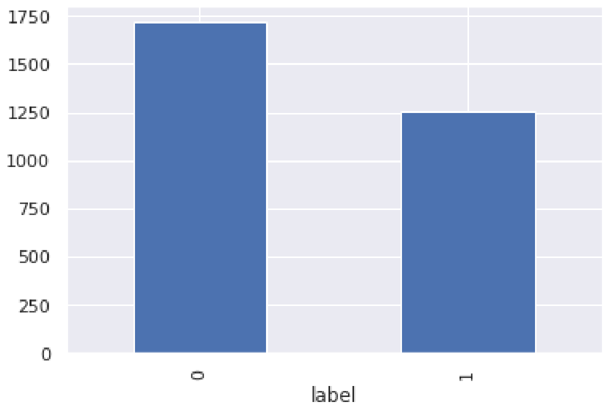
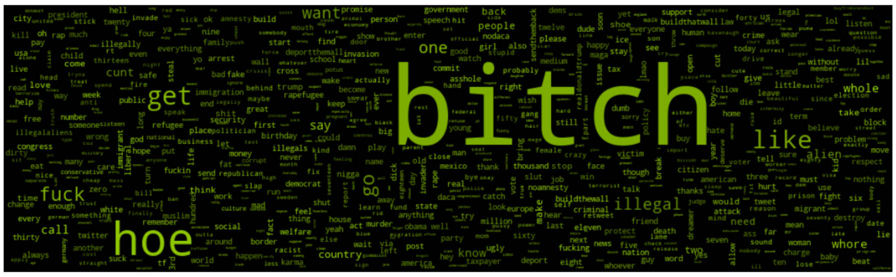


Figure 6

The overall word cloud of the test set:



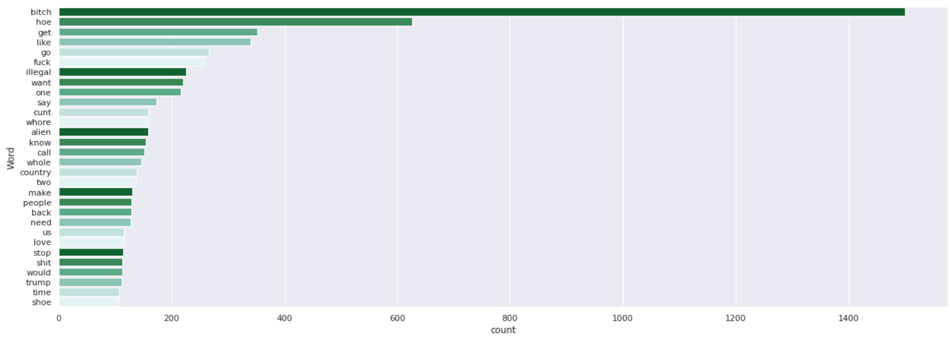
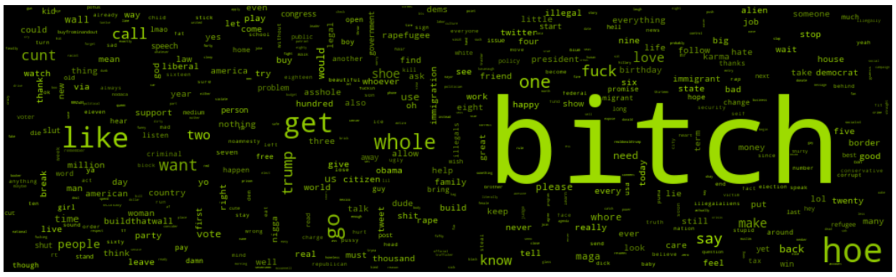


Figure 7

Word cloud for non-hate speech text in the test set:



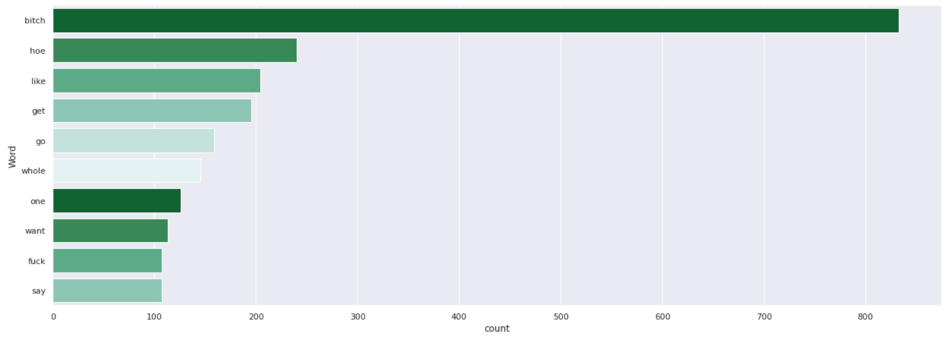
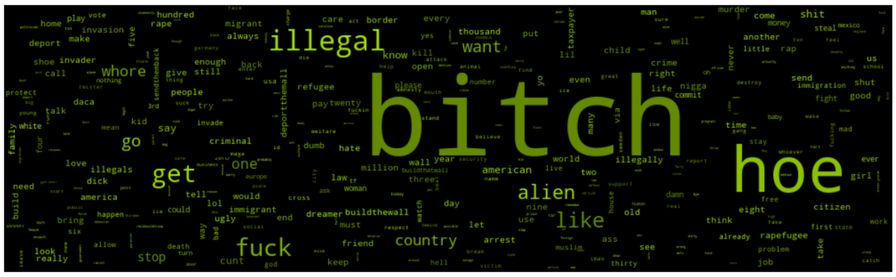


Figure 8

Word cloud for hate speech text in the test set:



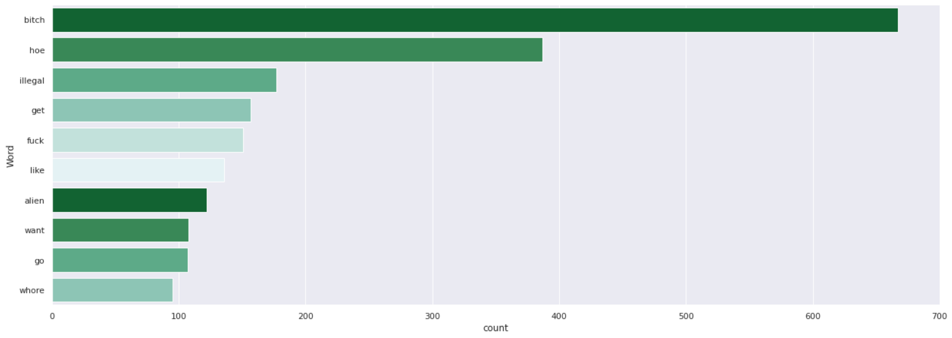


Figure 9

Evaluation set:

The total number of data entries in the evaluation set is 1000. There are 573 entries labelled as non-hateful and 427 entries labelled as hate speech. This constitutes almost 57% as non-hateful to 43% hate speech, as shown in Fig 10.

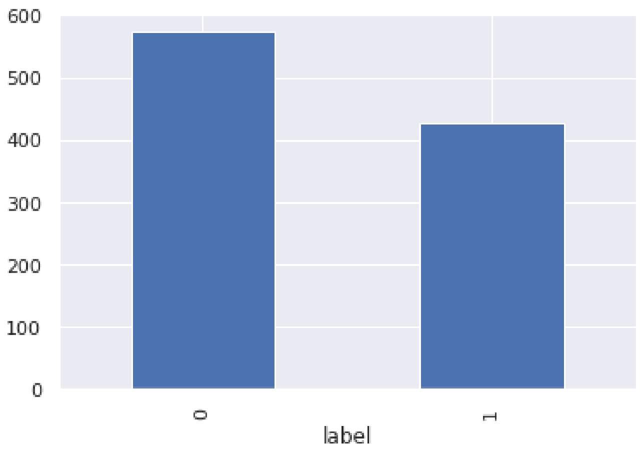
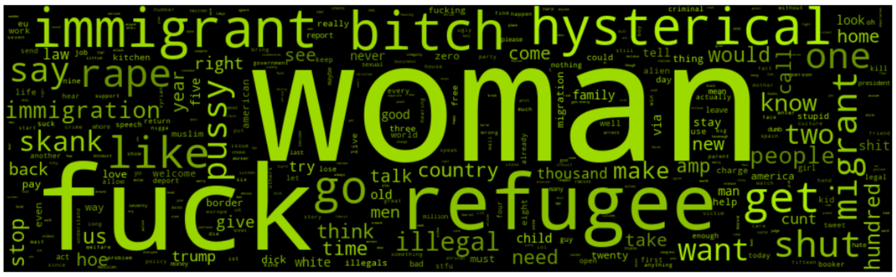


Figure 10

The overall word cloud of the evaluation set:



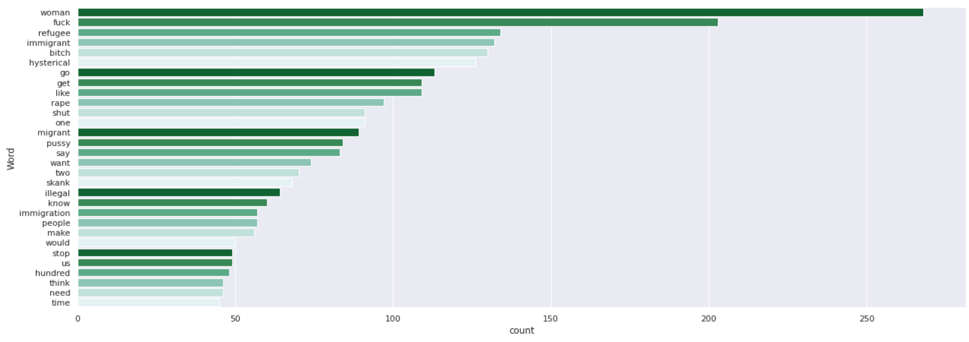
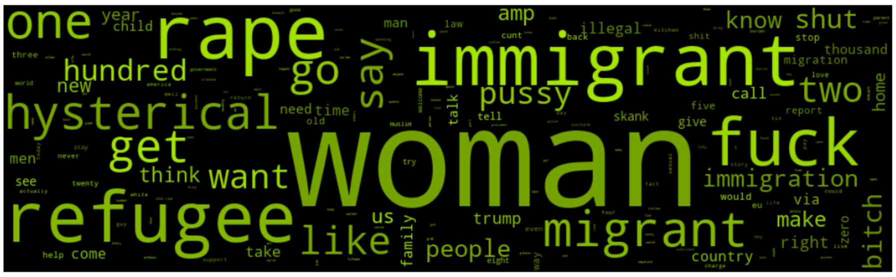


Figure 11

Word cloud for non-hate speech text in the evaluation set:



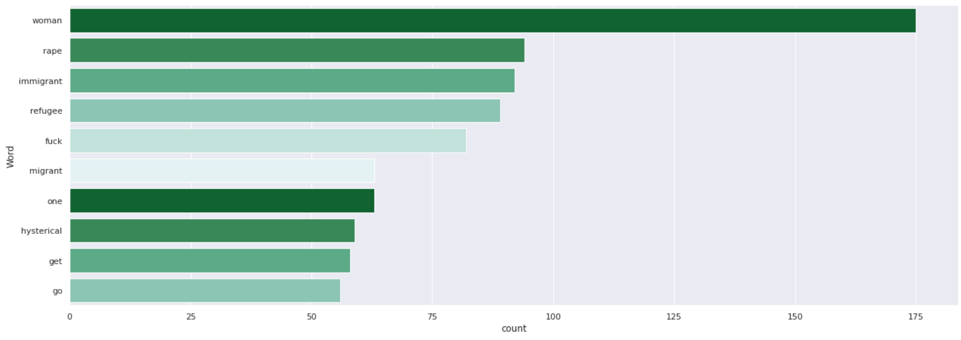
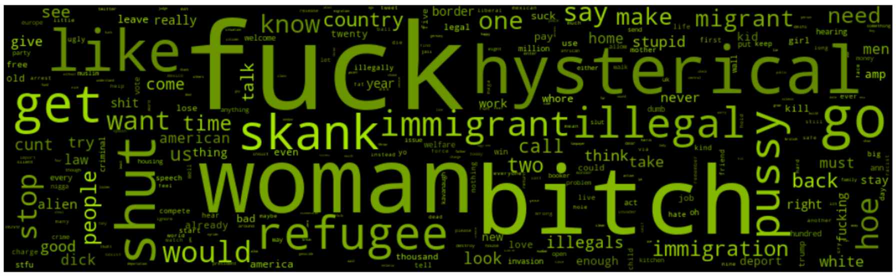


Figure 12

Word cloud for hate speech text in the evaluation set:



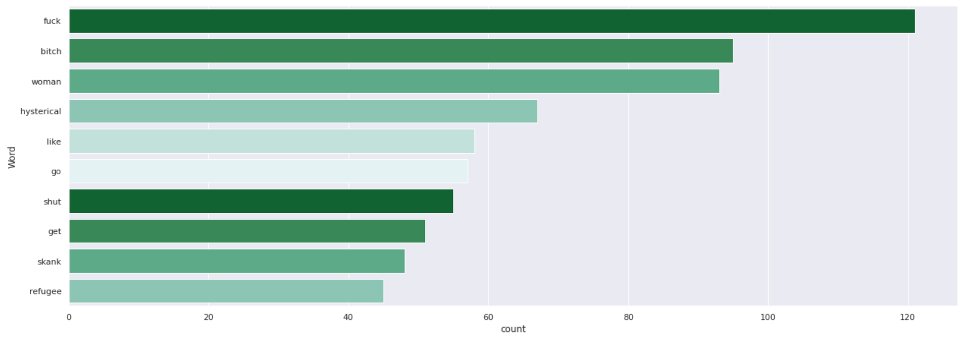


Figure 13

It is apparent from the word clouds and the word frequencies that the most frequent words in both of the data classes are quite similar, which makes it inefficient to only relay on word count as a classification method. This inclines us to use a deeper approach to construct the classification model such as using neural networks.

Related work

There has been a rapid development in the automated detection of hate speech in recent years. The interest has increased with the growing influence of social media platforms. This literature review focuses on studies that reveal hate speech in the textual context of social media.

[2]used the Hate Speech Detection dataset created by [3]; the data set consists of English tweets annotated to three classes "HATE", "OFFENSIVE", and "OK". character n-grams, word n-grams and word skip-grams features and a linear SVM classifier were applied. The best result was obtained by a character 4-gram model achieving 78% accuracy.

[3]collected and combined three Hate Speech Detection datasets of tweets. The dataset was split into train, validation and test sets. Then these sets are cleaned by removing user mentions, URLs and hashtags. Unigrams, semantic features, sentimental and patterns features and an SVM, Random Forest and g434 classifier were applied. It has reached an accuracy equal to 87.4% for classifying tweets into offensive and non-offensive, and accuracy equal to 78.4% for classifying tweets into hateful, offensive and clean.

[4] extracted data from Italian public Facebook groups and pages. The collected comments were annotated into three classes "No hate", "weak hate", and "Strong hate". Bidirectional LSTM and SVM were trained on ten-fold cross-validation. Two experiments were conducted first on three classes, "strong hate", "weak hate", and "no hate.", and second in the binary class "hate" and "no hate." The result on the binary classification performance was much higher than the three classes classification.

[5]used the Hate Speech Identification dataset. This dataset features English tweets that have been classified into three classes: "Hate", "Offensive", and "Neither". Word embedding features and CNN were applied. The model shows an overall precision of 91% and F1 of 90%

[6] Proposed to develop a voting ensemble of logistic regression, decision trees, and support vector machines classifier. The proposed method was evaluated against several widely used machine learning methods and showed improvement with an average Precision score of 93.7% and an F1 score of 94.2%.

As such and according the research and understanding obtained from these articles this project was influenced to use the following techniques in Data preprocessing:

1. Tokenization
2. Stop word, emoji removal and general text cleaning
3. URL and hashtag removal
4. Replacing abbreviation and spelling correction.

For the approach of this project, we have considered multiple approaches and saw an opportunity to use the following models for efficient and streamlined implementation:

1. RNN based classifiers
    a. Simple RNN
    b. Bi-directional LSTM
2. DNN based classifier.

An added benefit of choosing multiple models was to provide the reader with a comprehensive analysis of the performance of chosen implementation on the given circumstance of the data and the relative low processing units that the team uses, without resorting to hyper and multi threading techniques that were out of scope of this assessment.

Data pre-processing

The performance of the classifier is directly related to the quality of the dataset. The dataset is cleaned and normalised at this step to remove the typical noise and inconsistencies found in real-world data. Moreover, to solve redundancy in text duplicates and missing values were deleted and dealt with accordingly.

Text pre-processing

Removing stop-words

Stop words are very common words present in all texts. Words such as "a", "the", "and" are usually helping build ideas but do not carry any semantic meaning. As a result, it is usual practice to eliminate stop words. There are several stop-word libraries available, including NLTK and spaCy. We used the stop words list from the Natural Language Toolkit NLTK to remove it from the text.

Removal of user mentions, URLs, hashtags

Tweets usually include user mentions (@user), URLs and the hashtag sign (#hashtag). These also have been removed from the text, where they do not supply any information to the classifier and can be seen as noise.

Removing emojis

Twitter users usually use emojis such as 😋,😴 and 🤤 to express emotion. These emojis provide helpful information for humans but do not provide much information to machines regarding hate speech detection. Therefore, we have removed these emojis from the tweets.

Text cleaning and removing punctuation

Punctuation, special characters, single characters, and white spaces are removed using Python Regex to remove noise from the data.

Expanding Contractions

Contractions are shortened versions of words commonly used by social media users to minimise character counts. Such as” don’t” and “I’d” Converting each contraction to its expanded, original form aids in text normalisation. The Contractions module was used to expand contractions in the tweet text.

Spelling correction

Users often misspell words when writing in social media. Correcting spelling helps reduce a word from having more than one expression. To correct spelling errors in tweets, we used spellchecker.

Lowercase

The text case is being normalised so that all words are in lowercase. As a result, machine models do not treat capitalised words "Hi" differently than their lowercase "hi."

Replacing abbreviation and slang

Twitter's character limits encourage online users to use abbreviations and slang. It is critical to handle such theses abbreviations and slang in tweets by replacing them with their actual word meaning. We used a dictionary of these abbreviations and slang and replaced them with their actual meaning.

Lemmatization

This step's goal is to assign a word to its base form. We employed the NLTK WordNet Lemmatizer to convert words based on their part of speech to their actual root.

Remove missing values

In this stage, we will remove the missing values as it was noticed that there are missing tweets in the dataset provided to us. In addition, it was noticed that after text pre-processing, there would be missing values, such as those that tweets had only a user mention, so after cleaning, the tweet fields will be empty.

Remove duplicate values

When analyzing the data, a group of duplicate tweets was found, and therefore we deleted these repeated tweets and kept only one of them.

Implementation and Results

Implementation was initiated by doing a brief dataset analysis following by the vectorization of the datasets. After that, we trained a stacked model that contains just a simple RNN layer followed by a dense layer. To improve the results, we used another stacked model that replaced the simple RNN layer with two LSTM layers. In the end, we implemented a pure DNN model which was composed of just two dense layers. We used the *Keras* library for the implementation of the models.

In between model training and data preprocessing steps, there were a couple of necessary steps to improve the whole process. Those steps were dataset analysis and Vectorization of datasets before feeding them into the models.

Vectorization & Padding

In order to feed textual data to any mode, we need to vectorize our data into some meaningful numerical form and we achieve this by using vectorization. In our project, we used two models, i.e: a simple tokenizer provided by *Keras* library in the first two models, and TFIDF(term frequency-inverse document frequency) vectorizer in the last models.

During dataset analysis, we found out that most of the tweets were under 300 words, and very few of them exceeded this number. To feed tweets of the same length into our model, we implemented padding (a process that would truncate the longer tweets and appended 0s in shorter tweets to make all of them of equal length)

Structure of the First model

Our first model consists of two layers. The first layer is a simple Recurrent Neural Network (RNN) layer consisting of 256 units, followed by a dense layer. The reason to choose RNNs over other neural networks like CNNs was that RNNs are considered more suitable for text classification problems whereas CNNs are preferred for image recognition and classification. The reason for that is that RNN algorithms consider outputs of previous stages to decide the output of the next stage.

Training and Evaluation

We trained the model for 20 epochs with a batch size of 32. The training and validation loss was quite high at the start but both improved with each epoch. The training and validation accuracies were also improved over the same period with some fluctuations. At the end of the 20th epoch, training accuracy was around 63% while validation accuracy was approximately 69%. On evaluating the model against the validation dataset, we got ~54%

Structure of the Second model

To improve our results, we decided to replace the simple RNN layer with two LSTM layers. Long-short term memory (or LSMT) is an enhanced version of RNN. LSTMs are capable of learning long-term dependencies by retaining information for a longer period, thus offering better results in NLP problems. But LSTMs take more time in training as they have memory cells and do more operations in comparison with simple RNN.

To avoid overfitting the model on the training dataset, we added a dropout parameter in both LSTM layers. Both LSTM layers consist of 32 units and are made Bidirectional to allow back and forth flow. The model also contains an embedding layer as the first layer. There are also some pre-defined embeddings available like gloVe but our layer would create embeddings based on our dataset.

Training and Evaluation

The model trained with several epochs and batch size as of the first model but, as expected, there were significant changes in its behavior. With each epoch, the training accuracy seemed to improve while training loss kept decreasing but the case was completely opposite with the validation dataset. Surprisingly, the model accuracy on the testing dataset decreased to 52%

despite the model showing excellent accuracy over training data. We can argue that the model overfitted the training dataset despite our efforts to minimize it.

Structure of the Third model

The third model is implemented in a completely different manner from the first two models. We used a Tf-Idf tokenizer to vectorize datasets rather than a simple tokenizer. The model consists of just two dense layers. Dense layers are considered part of deep neural networks (or DNNs). DNNs are also known as feed-forward networks as data only moves forward. As there are no memory cells in DNNs and data only feeds forward, they are much quicker to train.

Training and Evaluation

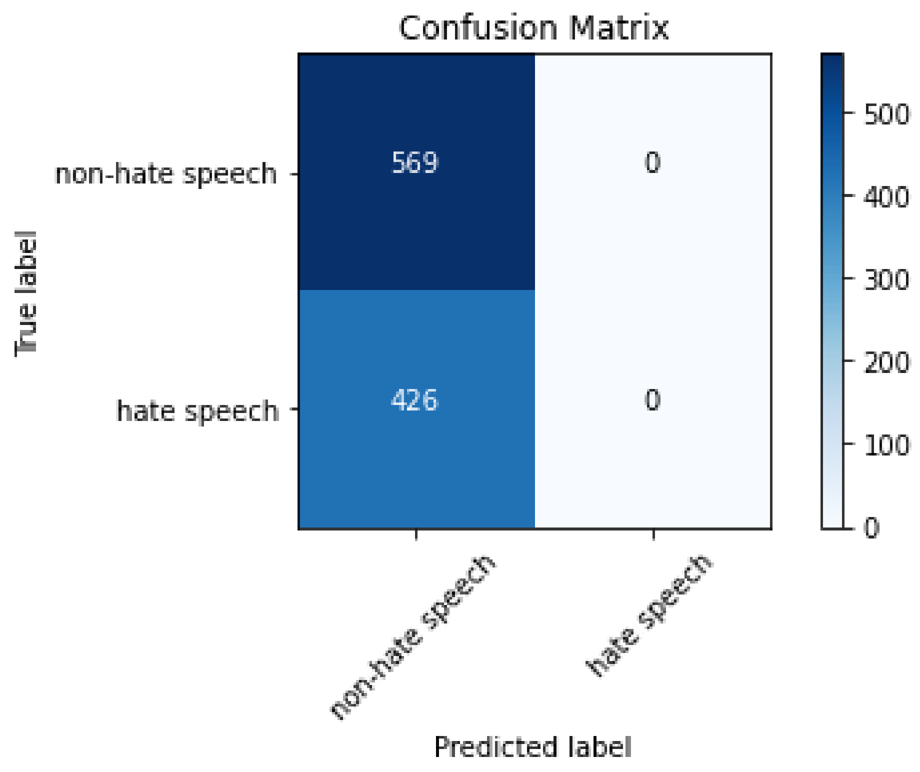
The model was trained for 20 epochs and on a batch size of 32. To our surprise, both testing and validation accuracies and losses kept improving with each epoch. When the unseen data was tested on the model, it gave better accuracy as well in comparison to the other two models.

Error analysis

The error analysis was performed using the given evaluation set.

Model 1:

Model1 is a stacked model that contains one embedding layer, one simple RNN layer and one dense layer.



Although the model has a an accuracy of 57%, by looking at the confusion matrix for model-1 we can clearly see that the model only tries to detect non-hate speech text without the classification of hate speech text. This is indicated by having no True Positives or False Positives in the confusion matrix.

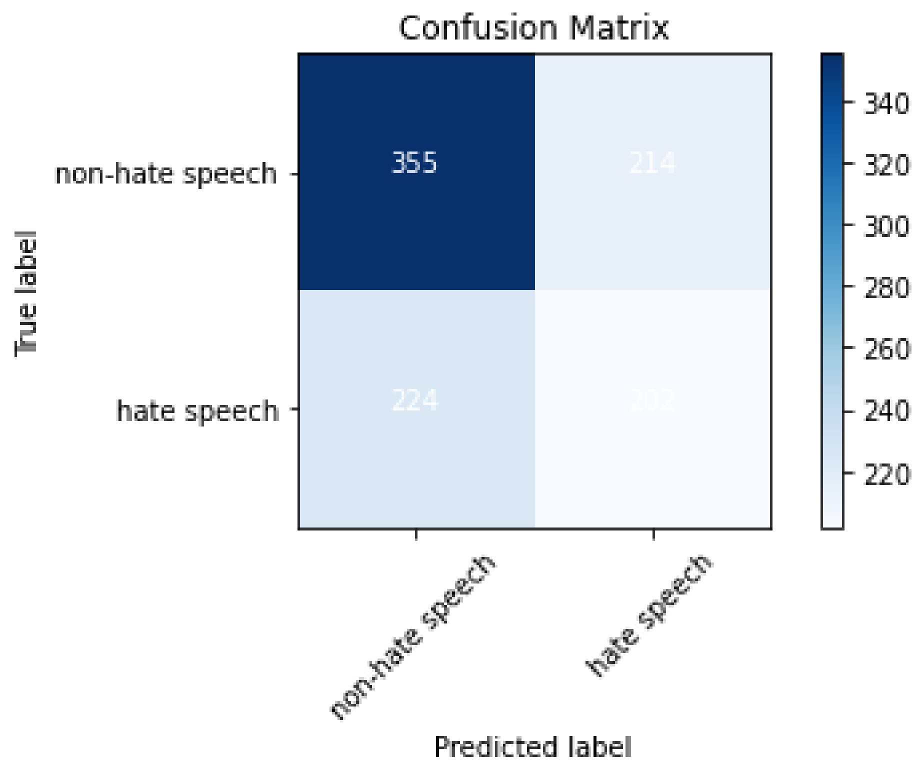
From the analysis of the errors, we can conclude that this model doesn't classify texts containing hate speech correctly as there are no predictions given for hate speech text since all text are classified as non-hate speech by the model.

Table 1: Model 1 measurements

Measurement	Formula	Value
Precision	$\frac{TP}{(TP + FP)}$	NA
Recall	$\frac{TP}{(TP + FN)}$	0
F-measure	$2 \times \frac{Precision \times Recall}{Precision + Recall}$	NA
Accuracy	$\frac{TP + TN}{(TP + TN + FP + FN)}$	0.57

Model 2:

In model2, we have added two LSTM layers apart from embedding layer and dense layer. To reduce overfitting, added dropout and bidirectional flow as well.



From the confusion matrix we can see that this model does in fact classify the text to hate speech and non-hate speech. However, the accuracy of the model 2 is lower than model 1 as it has an accuracy of 56%. Precision, Recall and F-measure are low as well which are below 0.48.

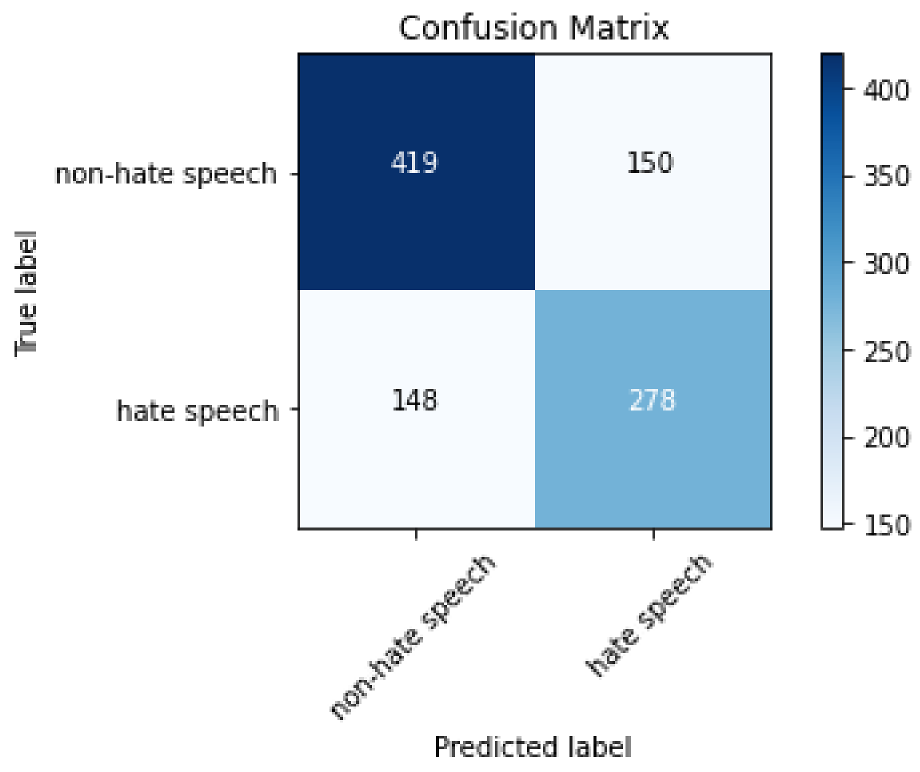
From the analysis of the model errors, we can state that this model can actually be used to classify hate speech text. Although it correctly classified 202 instances of the validation set as hate speech, it's performance is not so much better than a random classifier.

Table 2: Model 2 measurements

Measurement	Formula	Value
Precision	$\frac{TP}{(TP + FP)}$	0.48
Recall	$\frac{TP}{(TP + FN)}$	0.47
F-measure	$2 \times \frac{Precision \times Recall}{Precision + Recall}$	0.47
Accuracy	$\frac{TP + TN}{(TP + TN + FP + FN)}$	0.56

Model 3:

This model uses DMM, training and validation accuracy increased with each epoch while lose decreased, validation loss decreased as well. Better testing accuracy in comparison with the other two models.



This model gives an accuracy of 70% within a decent training time. Looking at the measurements, we can see that this model has a consistent performance across the board. The model has precision, recall and F-measure values of 0.65. Among the three models presented, this model has the best performance for classifying tweets with hate speech.

Table 3: Model 3 measurements

Measurement	Formula	Value
Precision	$\frac{TP}{(TP + FP)}$	0.65
Recall	$\frac{TP}{(TP + FN)}$	0.65
F-measure	$2 \times \frac{Precision \times Recall}{Precision + Recall}$	0.65
Accuracy	$\frac{TP + TN}{(TP + TN + FP + FN)}$	0.70

Conclusion and further improvements

To conclude, all desired models have been successfully implemented, the highest accuracy was 70% which was on DNN model. There are a few steps that can be taken to make further improvements to this project in the future. First of all, as mentioned in the report, we experienced overfitting which was discovered upon implementing the models. To overcome this problem, the models can be trained using larger datasets. The dataset provided was found to be not enough and to avoid this overfitting problem, if it wasn't the case that the data set is already pre-split, we could've taken more tweets and the overfitting would have been marginalized. Another problem we faced was the low testing accuracy. Hyperparameter tuning can be applied in order to improve accuracy. This is a rather rigorous method involving finding and then tuning the most relevant hyperparameters. In RNN and LSTM, hyperparameters can be no of units in a dense layer, dropout value, activation function, weights value, etc.

Bibliography

[1] "tweeteval: Repository for TweetEval," [Online]. Available: GitHub - cardiffnlp/tweeteval: Repository for TweetEval.

[2] D. H. S. i. S. M. S. M. /. Zampieri. [Online]. Available: https://arxiv.org/pdf/1712.06427.pdf .

[3] [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8292838.

[4] [Online]. Available: https://www.researchgate.net/publication/316971988_Hate_me_hate_me_not_Hate_speech_detection_on_Facebook/link/591af15d0f7e9beed7f5ff61/download.

[5] [Online]. Available: https://cs.vu.nl/~sbhulai/papers/paper-biere.pdf.

[6] [Online]. Available: https://www.proquest.com/docview/2652929787?pq-origsite=gscholar&fromopenview=true.