Hate speech detection, an RNN approach with GloVe

Haakon Jacobsen
Informatics, NTNU / 2021
haakonrj@stud.ntnu.no

Abstract

This paper presents an RNN-based approach for automatic hate speech detection of tweets using the user-friendly deep learning API Keras. It also provides a method to replace the simple embedding layers in Keras with the pretrained GloVe model to improve performance. The paper provides reflections on the results with regards to the problem of distinguishing hate speech from other offensive language.

1 Introduction

Social media platforms like Twitter, Facebook and Instagram have a big influence on our daily life. Unfortunately, these platforms ptovide a stage for people to express hateful and offensive content targeted at groups and individuals. With the continuous growth of social media, online hate speech has evolved with it (Tontodimamma, Nissi, Sarra et. al. 2020). This makes the use of automatic hate speech detection even more important. Big social media platforms like Facebook and Twitter have received continuous critique for how they tackle the issue. The critique has come from both organizations and individuals, and most recently a boycott was carried out by one of the biggest organizations within the world of football, the English Football Association (FA), also joined by the United European Football Association (UEFA), under the hashtag #StopOnlineAbuse (BBC, 2021).

As a part of a project in the course TDT4310 at the Norwegian University of Science and Technology (NTNU), this paper seeks to give a method for automatic hate speech detection by utilizing neural networks using the deep learning API Keras (Keras, 2021). This paper wishes to uncover how the use of emojis impact hate speech detection, and the project is focused on how hate speech distinguishes from other types of offensive language.

In this paper we will explore four neural network models to predict whether twitter posts are considered as hate speech, offensive or neither. This is done to evaluate the hypothesis that the dividing line between hate speech and offensive language is complex and difficult to detect.

The report will first introduce relevant background terminology and methods before it presents related work around hate speech- and offensive language detection. Then the experimental setup will be explained, including the architecture of the models. After this the result and evaluation of the findings will be presented and discussed. Lastly the report gives a conclusion and present potential future works.

2 Background

In this section important terminology and methods that are used throughout this paper will be presented.

2.1 Hate speech

What defines hate speech? Well, this is one of the trickier questions to answer since the term has no formal definition. Hate speech definitions can vary from a broad definition including hateful speech targeted towards all types of groups and individuals, to hateful speech targeted towards certain types of groups based on sex, race, religion, or sexual orientation. Hate speech is illegal in many countries including Canada, Germany, the United Kingdom and Norway, though under different formulation. In this paper hate speech is defined based on the definition provided in the classification of the dataset, which is *language that is used to expresses hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group* (Davidson, Warmsley, Macy and Weber, 2017 p. 512).

2.2 Spelling variations

The dataset used in this report, and twitter data in general, contains a lot of noisy data and is prone to *spelling variations,* which Pitsilis et. al. (2018, p. 4732) refers to as either intentional or unintentional. We humans will from time to time write words that are not grammatically correct. If there is no intention behind this, we call it unintentional. However, if the user writes wrong intentionally, like *"u fkn b1tch"*, the correct representation would be *"you fu*king b*tch.* This is a common method to save characters, and/or to avoid detection models like the one we are building.

2.3 Unbalanced datasets, under- and oversampling

Unbalanced datasets is, within data classification, a state where there exists a significant disproportion between the classes of a dataset, leading to majority- and majority classes. To deal with unbalanced datasets Kotsiantis et al. (2006, pp. 26-27) suggest two simple approaches, under- and oversample. Under-sampling is a method where we balance the distribution between classes by reducing the number of elements in the majority classes. Oversampling is another method where we balance the distribution by replicating the minority class. Both methods have drawbacks concerning data loss and overfitting, respectively (Kotsiantis et al., 2006, pp. 26-27).

2.4 Embedding layer

In neural networks, a embedding layer is often used as the first hidden layer. In short, the embedding layer tries to map similar words together in an n-dimensional vector space. This means that words with similar meaning can be mapped in a similar way, or simply put, they will be close to each other in the vector space. Mapping words in a vector space makes it possible to find words with similar representation/meaning, which can help capturing the context of a word in a text.

2.5 RNN, LSTM and GRU

Recurrent Neural Networks (RNNs) are a specific type of neural network that primarily have been proven to be effective when dealing with sequential data (Bengford, Blbro and Ojea, 2018, p. 280). Two well-known RNN's are the Long Short Term Memory (LSTM) model, and its younger brother Gated Recurrent Unit (GRU). These models will be used during experimentation of this report, and are implemented using the Keras library (Keras, 2021a). This paper is not focused on presenting a detailed description of the models, but rather how they can be implemented easily.

3 Related work

Within the field of automated hate speech detection, researchers have had different architectural approaches and different themes they wish to shed light on.

Davidson et al. (2017) looked at the problem of distinguishing hate speech and other instances of offensive language in tweets. Their experimental setup consisted of traditional machine learning approaches, using Support Vector Machines (SVM) and Logistic Regression. One of their main findings was that tweets with the highest predicted probabilities of being hate speech tend to contain multiple racist or homophobic slurs (p. 514). In their findings, 40% of the hate speech was misclassified by the detection model. This display the problem of distinguishing offensive language and hate speech. Their approach did not utilize neural networks, which has grown popular in more recent research.

Among neural network approaches we find Convolutional Neural Network (CNN) approaches such as Gambäck and Sikdar (2017), RNN approaches in Pitsilis et. al. (2018), and research which combines both architectures; Gambäck and Meyer (2019). Both Gambäck and Sikdar (2017) and Pitsilis (2018) focused on detecting hate speech from twitter data, although using different neural network models.

Gambäck and Skidar (2017, p. 88) provided results showing that CNN models outperformed Logistic Regression, which is one of the best performing methods in Davidson et. al. (2017). However, the data Gambäck and Skidar (2017) used was focused on hate speech related to 'racism' and/or 'sexism' and they found that the model did not identify many tweets as hate speech (p. 88). They explained that this was likely due to insufficient training instances of hate speech, as the dataset used was significantly unbalanced.

Pitsilis et. al. (2018) experimented with the RNN model type LSTM and based their hate speech data on the same classifications as Gambäck and Skidar (2017), namely racism and/or sexism or neither. In their work they added features such as user's previous tendency to post hatefully (p. 4733). They stated that their approach outperformed the state-of-the-art approaches, and that no other model had achieved better performance when classifying short messages (p. 4741).

As the two previous mentions focused on twitter data, Gambäck and Meyer (2019) focused on creating a platform independent hate speech detector, which target the weakness of models dependent on author features. Their work focused exclusively on text input, since additional information of authors may be lacking due to the difference in metadata stored by different platforms (p. 146). In their experiments they combined an architecture of both CNN and LSTM and used the same hate speech classifications as the two previous mentioned.

4 Experimental setup

In this section we will take a closer look at how we can go from raw data in the dataset, to a functional model predicting hate speech based on text input. We will look at the whole process of how data is being imported, cleaned, vectorized and passed into the neural network for training or evaluation. We will also look at implementations done to tune parameters for optimizing the model.

4.1 Dataset

The dataset used in this project is the dataset provided by Davidson et al. (2017). It consists of 24 783 tweets labeled either as hate speech, offensive or neither. The dataset was a random sample of 25 000 from 85.4 million tweets pulled from 33 458 Twitter users All these users had expressed hateful words or phrases found in the *Hatebase.org* lexicon. The Tweets was then manually tagged by CrowdFlower workers, who were given the definition in [2.1]. All workers were also specifically instructed that a presence of an offensive word did not necessarily indicate a tweet is hate speech (Davidson et al., 2017, p. 513). Table 1: illustrates the dataset with a count column of how many workers classified each tweet, and three columns for what they classified it as. The class column indicates the class of a tweet were the values; 0 = hate speech, 1 = offensive and 2 = neither. Note that the class *neither* is also referred to as *not offensive* later in this report.

	count	hate_speech	offensive	neither	class	tweet
0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't …
1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn bad for …
…	…	…	…	…	…	…
25 295	6	0	6	0	1	youu got wild bitches tellin you lies
25 296	3	0	0	3	2	"~~Ruffled | Ntac Eileen Dahlia - Beautiful color combination...

Table 1: Dataset from Davidson et. al (2017)

One problem of the dataset is the unbalanced class label distribution. There exist a lot more tweets labeled as offensive than hate speech and neither, which is illustrated in Figure 1. A low occurrence of one class label can cause models to not detect it, as they are not sufficiently trained. This point is also made by Gambäck and Skidar (2017, s. 88), concerning their own dataset.

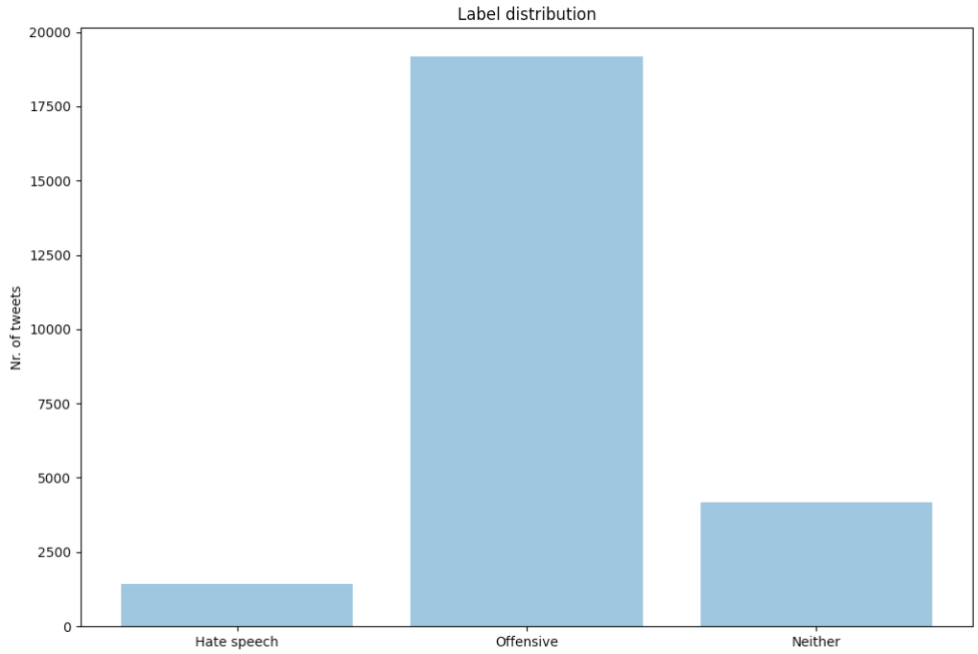

Figure 1: The dataset distribution of tweets based on label classification.

4.2 Data import and selection

When importing the data I extracted all tweets based on their respective class labels, providing 3 lists of tuples containing hate speech, offensive and not offensive tweets, including their respective class label from 0 to 2. The lists were then sliced to match the shortest list by the method of undersampling (Kotsiantis et al, 2006, s. 26). This gives us an equal distribution of 1/3 for each class label. The lists are then added together and shuffled, giving us a balanced dataset ready for cleaning. Note that the word "class" and "label" are used interchangeably throughout this report, where class address the type of tweets in general, while the labels maps tweets to a class.

4.3 Cleaning

Dealing with user generated data such as tweets requires a lot of cleaning for the model to perform well. In this section we will look at the approach used to clean individual tweets from the dataset.

Regular Expression (RegEx): First, we can use RegEx to filter noise such as URL links, @Usernames, before we split up concatenated words like *"should've"* to *"should have"*. We remove @Username because we are not interested in who's saying it, or to whom a tweet is directed. This information can however be valuable if the author belongs to an exposed group, and writing things that would normally be considered hateful coming from people outside that group (Gambäck and Meyer, 2019). Since this requires feature extraction of usernames and the metadata of the users are limited, I follow the same approach as Gambäck and Mayer (2019), by only considering textual input.

Emoji conversion: The dataset from Davidson et. al (2017) contains emojis converted in html decimal, which needs be converted back to character format. To convert we can use the TweetTokenizer from nltk (NLTK Project, 2021). Emojis can also be removed in the settings, and we will take a closer look at how emojis affect the results in [5.3].

Lower and split up: We can lower the sentence to remove capital variations before we tokenize the sentence to look at each token separately.

Clean each token: To clean each token we start by filtering out all stop words and words with a length less than three. We then remove the hashtag sign for all hashtags, keeping only the hashtag content. To solve spelling variations mentioned in [2.1], we can follow the approach suggested by Desai and Narvekar (2015, pp. 128 - 130). For each word we identify if the word exists in the vocabulary, which is the English vocabulary in this case. If the word is a not-in-vocabulary (NIV) word, we process the word by checking for common slang replacement. In this experiment I used a slang words dictionary I created from different resources including (Kshirsager, 2021). If no word is found, Desai and Narvekar (2015) adds a last step to check for the most probable word replacement. I initially tested with a correction method from TextBlob (GeekForGeeks, 2019), but this was later discarded when its prediction replaced a lot of the slang words with completely different words, manipulating the meaning of the text.

Lemmatizing or Stemming: After searching for each word and matching it, the word is lemmatized or stemmed based on a setting in the program.

Tokenization and Vectorization: Since neural networks only accepts numbers as input we must represent tweets and labels as numbers. To do this we can use the Tokenizer provided by TensorFlow (2021) in Keras. We can create a indexed vocabulary of the all the words and their frequency. Tweets are replaced with their respective index number. E.g., *'hi my name is…'* transforms to [340, 57, 800, 21…], where the number 340 represent the word 'hi', 57 represent the word 'my' ect. Each tweet is then padded to get an equal length of all tweets. The length of each tweet can be adjusted within the settings and the size of the vocabulary is limited to a setting for a maximal number of words.

4.4 Deep learning model

During experiments, four models were tested. The difference in these models were mainly in the embedding layer, and the hidden layer.

4.4.1 Embedding layer

I added an embedding layer as the input layer for all four models. During experiments I used two types of embedding layers: Keras own build in embedding, and the GloVe embedding (Pennington et al., 2014), which is trained on 2 billion tweets. The GloVe embedding was chosen as an attempt to increase the performance from the built in Keras embedding.

4.4.2 LSTM and GRU layer

I used two RNN's for experimenting. These two consists of the LSTM model, and the GRU model. The models were implemented using Keras, which makes working with neural networks easier. Both models are tested with the two embedding layers mentioned above. This gives us a total of 4 experimental models; 1. Keras + LSTM, 2. GloVe + LSTM, 3. Keras + GRU, 4. GloVe + GRU.

4.4.3 Dropout and Dense layer

All experimental models use a Dropout layer with a dropout rate of 0.5. The last layer is a Dense layer with softmax activation, creating three output layers to predict the class label of a tweet (0 = hate speech, 1 = offensive and 2 = neither). All models use categorical_crossentropy, the adma optimizer and is evaluated based on loss, accuracy, precision, recall and f1-score. The performance of each model is displayed in Table 2.

4.5 Global settings

The setup contains different settings that can influence the result. This section seeks to summarize all of them:

1. **MAX_LEN:** Defines the maximal token length of a tweet. The setting is used for padding tweets and set a limit to tweet length. The setting was set to 30 during experimentation.
2. **MAX_WORDS:** Defines the maximum number of words in the vocabulary created in tokenization. The setting was set to 30 000 during experimenting.
3. **USE_EMOJI:** Defines if emojis should be kept during cleaning. Experimentation consisted of both using emojis and filtering emojis.
4. **LEM_OVER_STEM:** Decides whether the words should be lemmatized or stemmed during cleaning. Both settings were experimented.
5. **TEST_SIZE:** Defines what proportion of the dataset should be used for testing. During experiments I used 25%.

5 Experiments and results

In this section we will take a closer look at how the LSTM and GRU models perform with different parameters and settings. We first look at the optimal epoch count, the difference in the two embedding layers, and how the LSTM compare to the GRU. We then look at the program's different settings; emoji vs. no-emoji and lemmatization vs. stemming. Then we take a more detailed look at how the best model predict hate speech, offensive and non-offensive tweets in a confusion matrix.

5.1 Tuning parameters

The model has been tested with different number of epochs to prevent overfitting. To find the optimal number of epochs, I added early stopping, which shortly described checks if a metric (e.g. accuracy) continues to increase for test data as epoch number increase for the training data. In my implementation early stopping was set to minimize the loss, setting the max epochs to 100 for each model with a patience of 5 epoch periods before stopping. The lowest loss result after stopping was then restored, giving an optimal epoch distribution of; 25 for (LSTM + GloVe), 5 for (LSTM + Keras), 7 for (GRU + GloVe) and 4 for (GRU + Keras). For bach_size I experimented with both 32 and 64, which gave the same results, thus 32 have been chosen in my implementation.

5.2 Model results

After tuning all models for optimal Epochs, we get the results presented in Table: 2. The models were evaluated based on loss, accuracy, precision, recall and F1-score. The settings were set to no emojis, stemming, max word of 30 000, max length of 30 characters and a test size of 25%.

	Keras + LSTM *Epochs: 25*	***GloVe + LSTM*** *Epochs: 5*	***Keras + GRU*** *Epochs: 7*	***GloVe + GRU*** *Epochs: 2*
LOSS	0.5086	**0.4408**	0.4797	0.4510
ACCURACY	0.8061	**0.8434**	0.8145	0.8397
PRECISION	0.8193	**0.8496**	0.8330	0.8459
RECALL	0.7903	0.8322	0.7996	**0.8341**
F1-Score	0.8045	**0.8408**	0.8159	0.8399

Table 2: Evaluation of the four models (No emojis with stemming)

The result show that the GloVe embedding had significant impact in all major metrics for both the LSTM and GRU model compared with the Keras embedding. The model with the overall best performance was the GloVe + LSTM model, achieving best results in loss, accuracy, precision and F1-score. However, the results are very similar to the results in the GloVe + GRU model, which scored highest on the recall metric. The best performing model had an accuracy of 84,34% and an F1-score of ≈ 84%, which was about same as GloVe + GRU. The following test will adapt the GloVe + LSTM model.

5.3 Settings, emojis and lemmatization

In Table 3 we can see the small negative differences caused by including emojis in the cleaning process. The model showed better results when using stemming over lemmatization, and no combination worked better than excluding emojis and stemming the words. Better performance with stemming was surprising, because lemmatizing words actually insures you get a word that exists, which stemming do not. However, lemmatization and emojis increased hate speech detection (Figure: 2c). And the model was better at predicting tweets belonging to the neither class when using emojis (Figure: 2b).

	GloVe + LSTM *No Emoji* *Stemming*	***GloVe + LSTM*** *With Emoji* *Stemming*	***GloVe + LSTM*** *No Emoji* *Lemmatization*	***GloVe + LSTM*** *With Emoji* *Lemmatization*
LOSS	0.4408	0.4406	0.4440	**0.4373**
ACCURACY	**0.8434**	0.8359	0.8369	0.8313
PRECISION	**0.8496**	0.8415	0.8433	0.8415
RECALL	**0.8322**	0.8313	0.8275	0.8266
F1-Score	**0.8408**	0.8363	0.8353	0.8340

Table 3: Evaluation the GloVe + LSTM model with different settings.

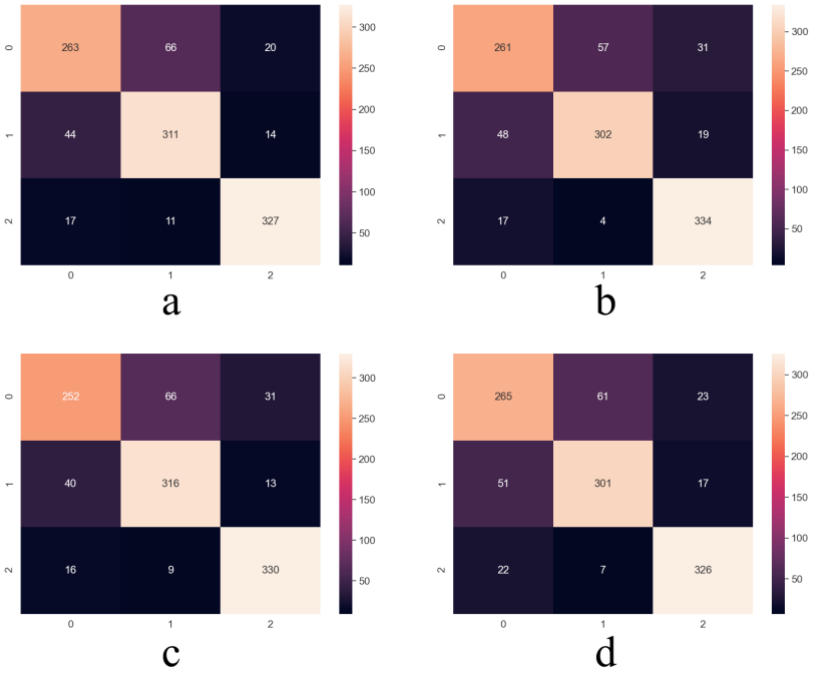

Figure 2: Four confusion matrix of the different results based on the GloVe + LSTM model with the settings in Table 3.

5.4 Misclassifications

From all subfigures in Figure 2, we can see a clear pattern. The model preforms better when classifying the neither class, with an overall accuracy of 92%. On the other end, we can see the model struggles more when classifying hate speech, with a accuracy of only 75%. Lemmatization increases misclassification of offensive tweets by 8% when comparing Figure: 2c and 2d. Studying the GloVe + LSTM model in Figure: 2a we discover that 19% of the tweets labelled as hate speech was misclassified as offensive.

5.5 Detected vs. not detected hate speech

If we look at the most common words in the tweets where the model succeeded in classifying hate speech, we can see the most common words are; *f*ggot* and *n*gger*. Two words strongly associated with racist and homophobic speech. This represents the same findings of Davidson et al. (2017, p. 514), which found racial or homophobic slurs to be contained in the tweets with highest probability of being hate speech. If we look at the most common words where the model predicted hate speech to be offensive, we can see the most common words are *n*gga, b*tch* and *h*e.* The word n*igga is used a lot in both hate speech and offensive language.

Figure 3a: Word cloud of the most common words in successful predictions of hate speech.

Figure 3b: Word cloud of the most common words in hate speech predicted as offensive.

6 Evaluation and discussion

During experimentation I discovered that the best performing model was bad at distinguishing hate speech from offensive tweets. However, when predicting if something is offensive or not, it preformed much better. This is the same problem Davidson et al. (2017)[p. 513] had, where 40% of hate speech was misclassified as offensive. Drawing the line between hate speech and offensive tweets can be difficult, and seemingly more difficult than drawing the line between offensive and not offensive tweets. The difficulties could be caused by the problem of word ambiguity, which Pitsilis et. al. (2018, p. 4732) describe as word having different meaning in different context. It can also be cause by bad cleaning of data, as the slang dictionary used in the cleaning process is far from sufficient, missing a lot of intentional misspelling for slur words, which the WordClouds in Figure 3a and 3b also reveal.

The word cloud indicate that the model predicts sentences with the word b*tch and n*gga as both hate speech and offensive. Davidson et al. (2017, p. 512) state that racist and homophobic expressions are often classified as hate speech, and that sexism is usually classified as offensive. Although this is a concern related to the manually labeling of the dataset, we still see the word b*tch and n*gga being used a lot in the models misclassified hate speech tweets, meaning these words overlap a lot.

Although my model has a lower F1-Score of (84%) compared to Davidson et al. (2017), which achieved 90%, my model has a higher score when classifying hate speech (75%). This might be caused by the undersampling done on the dataset, as their model might be trained on more offensive data than mine. It is worth mentioning that the undersampling of the dataset caused it to go from 24 000 tweets to around 4000, which potentially caused the model to miss out on a lot of defining data. With a test data split of 75/25, the test instances are few, and changes pointed out in the confusion matrix has to be taken with a grain of salt.

7. Conclusion and future work

In this report I have looked at an RNN approach using LSTM and GRU to detect hate speech from twitter data. The main findings are that both the LSTM and GRU model increase performance when paired with a GloVe embedding layer. The LSTM and GRU had very similar results when paired with the GloVe embedding, but the best performing model comes up short when differentiating between hate speech and other offensive tweets, although predicting non offensive tweets with a higher precision.

In future works it would be interesting to compare the GloVe embedding with different types of embedding layers like word2vec and FastText. It would also be interesting to extract author related features like the one implemented by Pitsilis et al. (2018). Testing the model on different datasets would also be interesting as this can test its ability to generalize on other data.

References

BBC. (2021, 30. April) Social media boycott: Football clubs, players & sporting bodies begin protest. URL: https://www.bbc.com/sport/56936797

Benjamin Bengford, Rebecca Bilbro and Rony Ojeda. Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning (1st. ed.). O'Reilly Media, Inc. 2018.

Thomas Davidson, Dana Warmsley, Michael W. Macy and Ingmar Weber. Automated Hate Speech Detection and the Problem of Offensive Language. *Proceedings of the 11th International AAAI Conference on Web and Social Media*, pages 512-515, Montreal, Canada. 2017. http://arxiv.org/abs/1703.04009

Neelmay Desai and Narvekar Meera. Normalization of Noisy Text Data. *Procedia Computer Science*, 45, pages 127-13. 2015. https://doi.org/10.1016/j.procs.2015.03.104

Björn Gambäck Johannes and Skjeggestad Meyer. A Platform Agnostic Dual-Strand Hate Speech Detector. *Association for Computational Linguistics*, pages 146—156. Florence, Italy. August 2019. ACL. https://doi.org/10.18653/v1/W19-3516

Björn Gambäck and Utpal Kumar Sikdar. Using convolutional neural networks to classify hate-speech. In *Proceedings of the 1st Workshop on Abusive Language Online*, pages 85–90, Vancouver, Canada. 2017. ACL. http://dx.doi.org/10.18653/v1/W17-3013

GeekForGeeks (2019), Python TextBlob Correction Method. URL: https://www.geeksforgeeks.org/python-textblob-correct-method/

Keras. (2021). Simple. Flexible. Powerful. URL: https://keras.io

Sotiris Kotsiantis, D. Kanellopoulos and P. Pintelas. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pages 25–36. 2006. https://www.researchgate.net/publication/228084509_Handling_imbalanced_datasets_A_review

Vedant Kshirsager. (2020). Twitter-Hate-Speech-Detection. GitHub. URL: https://github.com/vedant-95/Twitter-Hate-Speech-Detection/blob/master/Data%20Cleaning.ipynb

NLTK Project. (2021). NLTK 3.5.2 documentation. URL: https://www.nltk.org/api/nltk.tokenize.html

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. *In Processdings of the Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar. 2014. ACL. http://dx.doi.org/10.3115/v1/D14-1162

Georgios K. Pitsilis, Heir Ramampiaro and Helge Langseth. Effective hate-speech detection in Twitter data using recurrent neural networks. *Applied Intelligence* 48**,** pages 4730–4742. 2018. https://doi.org/10.1007/s10489-018-1242-y

TensorFlow. (2021). Tensorflow API Docs. URL: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer

Alice Tontodimamma, Eugenia Nissi, Sarra, Annaliana Sarra and Lara Fintanella. Thirty years of research into hate speech: topics of interest and their evolution. *Scientometrics* 126, pages 157–179. 2021. https://doi.org/10.1007/s11192-020-03737-6

UEFA. (2021, 29 April) *UEFA announces joining the Social Media Boycott.* URL: https://www.uefa.com/insideuefa/mediaservices/mediareleases/news/0268-12280374487a-c04dc481ab1c-1000--uefa-announces-joining-the-social-media-boycott/