Hate Speech Detection Using a Traditional Ensemble Model and a Neural RoBERTa Model

Siebe Brouwer
Morine Croguennec
Caroline Vandyck

1 Introduction

The growth and widespread availability of the internet and social media platforms allows people to interact with each other regardless of time or location (Hee et al., 2018). However, apart from friendly interactions, it also enables hate speech that could possibly reach a large crowd. The exact definition of hate speech differs from platform to platform and dataset to dataset (Fortuna and Nunes, 2018; MacAvaney et al., 2019). According to the UN, it entails "denial of the values of tolerance, inclusion, diversity and the very essence of the human rights norms and principles"(Cited May 2022). Although this happens online and not face-to-face, "when left unchecked, expressions of hatred can [...] harm social cohesion, peace and development, as it lays the ground for conflicts and tensions, wide scale human rights violations, including atrocity crimes." (UN, Cited May 2022) Accordingly, it is of importance to detect and eliminate harmful content.

As it is unfeasible to detect all hate speech manually, models detecting hate speech can be of great use. Although they have been proven to be accurate and helpful, there is still room for improvement. In that context, many shared tasks and competitions emerge. In regards to the Natural Language Processing course of the MA Digital Text Analysis at the University of Antwerp, a competition similar to the OffensEval 2019 shared task organised at SemEval 2019 was held (Zampieri et al., 2019b). The same dataset (Zampieri et al., 2019a) was used, expanded with extra test sets. Our task was to create one non-neural and one neural model, capable of classifying posts either as offensive or non-offensive.

For the non-neural part of the task, we created an ensemble model using models based on an SGD classifier, a logistic regression classifier and two versions of SVM classifiers: a Nu-Support Vector Classifier and a C-Support Vector Classifier. All of the models were trained on features and preprocessed posts that were adjusted for the model, in addition all parameters and weights were optimised. For the neural part of the task, five RoBERTa models were given differently preprocessed posts as input and, using a majority vote, we aimed to return a final label. However, due to a coding error, the labelling was done by a single RoBERTa.

2 Related Research

2.1 Features

Many features have already been explored when detecting hate speech. Amongst others, Schmidt, Wiegand (2017) and Alorainy et al. (2018) claim that it is beneficial to expand the textual, surface features with other features as well. This is an overview of the state-of-the-art features that inspired us and how we implemented them.

Bag of Words (BoW) A common way to represent a text, is by creating a BoW, which entails generating a frequency dictionary based on the training data. This can be done by using either word or character n-grams. Whereas a BoW created on the basis of words loses the syntactic sequence and consequentially the semantic context, a BoW created on the basis of character n-grams partially overcomes this (Burnap and Williams, 2016). Next to that, character n-grams also overcome typing errors or spelling variation, a common characteristic of online writings (Schmidt and Wiegand, 2017; Fortuna and Nunes, 2018; Alorainy et al., 2018). Often, an n-gram range of 1-5 is used (Alorainy et al., 2018; MacAvaney et al., 2019; Burnap and Williams, 2016), this range was also explored in our models. Next to that, we explored word n-grams as well as character n-grams.

TF-IDF A variation on the 'regular' BoW is the term frequency-inverse document frequency (TF-IDF). It differs from a BoW because it does not calculate just the frequency, but "the importance of a word in a document within a corpus and increases in proportion to the number of times that a word appears in the document" (Fortuna and Nunes, 2018). Just like a BoW, this can be built on either word or character n-grams. A TF-IDF based on character n-grams seems to be a recommended method for classifying hate speech (Markov and Daelemans, 2021; Alorainy et al., 2018; MacAvaney et al., 2019; Burnap and Williams, 2016). We explored both the regular BoW (using the sklearn's CountVectorizer) and the TF-IDF.

Other surface level features Apart from the text itself, there are other surface features relating to it, such as capitalisation (Burnap and Williams, 2016; MacAvaney et al., 2019), interpunction (Alorainy et al., 2018; MacAvaney et al., 2019), and the frequency of URL's, @ mentions and hashtags (Davidson et al., 2017; Gambino and Pirrone, 2020). Because of that, the influence of casefolding, leaving out the punctuation or adding it as an extra feature, as well as the frequency of URL's, mentions and hashtags were also explored. Lastly, there is another surface level feature typical for online texts, namely emojis. We wanted to make sure that our models took these into account as well. Accordingly, we used the demojize function from the emoji package to translate every emoji into its corresponding textual string.

Parts-of-speech (POS) Since tagging the parts-of-speech of the words in a text can uncover the role of a word in its context (Fortuna and Nunes, 2018), using this linguistic feature has proved to be beneficial as well. Several researchers have either used POS in their representation of the text or as an additional feature (Markov and Daelemans, 2021; Alorainy et al., 2018). We opt for adding nltk's universal tagset as an extra textual feature. Apart from translating every word to its corresponding POS, we added custom tags for URL's, @ mentions and hashtags to incorporate these features as well, just as Gambino and Pirrone (2020) did.

Lemmatization Another feature that could possibly improve a model, is lemmatizing the words of a text. Just as with POS tagging, several researchers have either used lemmas in their representation of the text or as an additional feature (Markov and Daelemans, 2021; Markov et al., 2021; Hee et al., 2018). This could be beneficial because all variants of the same word get transformed into the same root form. This could lead to less sparse data. We explored whether our models improved by using lemmatized tweets or by adding them as an extra feature.

Sentiment Lexicon Using a sentiment lexicon improves results as well (Alorainy et al., 2018; Markov and Daelemans, 2021). We will create extra feature columns using Vader, a lexicon and rule-based sentiment analysis library that returns a sentiment score between -1 (maximally negative) and 1 (maximally positive); and using the AFINN lexicon, which returns a similar score but in the range of -5 to 5. Vader proved to be highly accurate, even more than human annotators (Hutto and Gilbert, 2014). Next to that, AFINN proved to be very promising as well (Årup Nielsen, 2011). Both lexicons were created particularly for microblogs (such as tweets).

2.2 Models

As already mentioned, our traditional model is an ensemble model based on an SGD classifier, a logistic regression classifier, and two SVM models. We opted for these models since they have proved to yield accurate results in the past. For instance, Markov and Daelemans (2021), Burnap and Williams (2016), and MacAvaney et al. (2019), used the SVM algorithm in their research and it obtained excellent results. Next to that, in researches by, amongst others, Alorainy et al. (2018) and Davidson et al. (2017), the logistic regression model yielded reliable results as well. Lastly, previous research has also shown that the SGD classifier is promising (Sharif et al., 2020), especially since the gradient of the loss is estimated each sample at a time.

For the neural model we base our methodology on that of a 2021 paper on the improving of cross-domain hate speech detection by Markov and Daelemans. They brought our attention to the pretrained language model RoBERTa (Liu et al., 2019). Markov and Daelemans then combined this RoBERTa with other models through a hard majority-voting method. This inspired us to apply a hard majority vote to the results of 5 RoBERTas which each underwent a unique preprocessing of train and test data. At first the inclusion of Berts (Devlin et al., 2019) within the majority

vote was also intended, however the top 5 RoBERTas all outperformed the very best Bert. Unfortunately, the majority vote's result was not exported correctly. Instead, the result calculated by a single RoBERTa was wrongfully selected for the result exportation.

3 Methodology

3.1 Data

We trained our models on the Offensive Language Identification Dataset (OLID) (Zampieri et al., 2019a) and evaluated them on four more datasets, of which one was in-domain.

3.1.1 Training Data

OLID The OLID training corpus consists of 13240 annotated English tweets and was created in the sphere of the OffensEval 2019 shared task organised at SemEval 2019 (Zampieri et al., 2019b). The task included differentiating offensive from non-offensive tweets, categorising offence types and identifying the target. Accordingly, the tweets are annotated hierarchically on those three levels by crowdsourced, experienced annotators. Every tweet was annotated by two annotators and in case of disagreement, a third annotator was called upon. Tweets were regarded offensive if they contained "any form of non-acceptable language (profanity) or a targeted offence, which can be veiled or direct. This includes insults, threats, and posts containing profane language or swear words" (Zampieri et al., 2019a). As stated before, we only focus on the more coarse grained classification, namely whether a tweet is offensive or not.

Before fitting the models on the tweets, we normalised html specific characters and other non-utf8 encodings which were sometimes still present in the dataset. After that, during the training and optimization stage of our traditional models, we used 80% of this dataset as a training set and used the remaining 20% as a test set. In the training and optimization stage of our neural models, 20% of the training data was set apart as well as shuffled at the same random state for all 5 models. The statistics of this dataset (and the four test sets) can be found in Table 1.

3.1.2 Test Data

OLID The first test set are the remaining 860 in-domain tweets of the OLID corpus (Zampieri et al., 2019a). When evaluating our models on the test sets, all 13240 tweets were used for the training.

Ruddit The second test set is a sample of approximately 1200 English posts from the Ruddit dataset (Hada et al., 2021). This dataset consists of posts from several subreddits of Reddit. The posts were annotated by crowdsourced US annotators with a high approval rate. The posts all received a score from -1 (maximally supportive) to 1 (maximally offensive). This score was assigned by best-worst scaling. Every annotator was presented a four tuple of posts and had to rank them from maximally supportive to maximally offensive. Every four tuple was annotated by six annotators. For our research, posts that received a score lower than -0.4 were interpreted as not offensive and posts that received a score higher than 0.4 were interpreted as offensive. That way, borderline cases were excluded.

Wikipedia The third test dataset is a sample of English Wikipedia comments used in the first Jigsaw competition on Kaggle (AI, 2018). The teams had to predict probabilities for different types of (non-)toxic posts. The comments were manually annotated. The sample that we used contains non-toxic comments, moderately offensive comments and severely offensive comments.

TEXTGAIN The fourth dataset was obtained from the AI company TEXTGAIN. It consists of English football tweets that have been manually labelled as offensive or not offensive. Once again, when loading in the test data, we normalised html specific characters and other non-utf8 encodings which were still be present in the dataset.

3.2 Baselines

The results of four baseline models were handed to us. Two baselines are traditional models and two are neural models. We compared the performance of our own models to these baselines, our goal was to construct models that outperform the baselines.

3.2.1 Traditional Baselines

The first baseline is a model that attributes every instance to the most frequent class. The classifier does not perform well. In Table 2, the obtained macro-averaged F1 score for every test set is shown. We show and evaluate on the macro averaged F1-score since all classes need to be treated equally in regards to their overall performance. The "Total" column refers to the mean of the first three test sets. The TEXTGAIN dataset is handled

	NOT	OFF
OLID (train)	8840	4400
OLID (test)	620	240
Ruddit	664	543
Wikipedia	600	600
TEXTGAIN	1088	188

Table 1: Statistics datasets

separately. The classifier performs relatively better on the TEXTGAIN dataset because it consists of more non-offensive tweets.

The second baseline is a SpaCy BoW-model that makes use of a TfidfVectorizer. The vectorizer uses a maximum of 5000 features, unigrams and bigrams, and lowercased characters. The model performs adequately.

3.2.2 Neural Baselines

The baselines for our neural network are the English pretrained language models Bert and HateBert (Caselli et al., 2020). Bert is a bidirectional transformer that was pretrained on the Toronto Book Corpus and Wikipedia. Both models perform very similarly and well. Hatebert was created by Caselli et al. in 2020 by retraining the general Bert base uncased model on the Reddit Abusive Language English (RAL-E) dataset, which consists of English comments from banned communities. Caselli et al. report that the macro-averaged F1 score increases in comparison to the general Bert. They claim that using "data from a different social media platform does not harm the fine-tuning stage of the retrained model" (2020).

3.3 Experiments

3.3.1 Traditional Classification Experiment

We made an ensemble model by use of the voting classifier. Within the ensemble, four traditional models were combined. These included a linear model with a stochastic gradient descent (SGD) classifier, a logistic regression (LR) model, a linear support vector machine (SVM1) using a c-support vector classifier and a linear support vector machine (SVM2) using a nu-support vector classifier. All models had the same random state, 42, for the sake of reproducibility. Their class weights were set to 'balanced', this was necessary since the train and test datasets had an uneven division of output classes. Except for these two parameters, each model made use of personalised preprocessing steps and optimal features according to multiple gridsearches with a cross-validation of 2.

SGD The stochastic gradient descent model made use of early stopping to prevent overfitting, counting 3 iterations without improvement before the training of the model was stopped. The penalty parameter was set to 'elasticnet', meaning that the model combined L1 and L2 priors as regularizers. To calculate the loss, a logistic regression probabilistic classifier was implemented. The last parameter within the stochastic gradient descent classifier that differed from the default was alpha, the constant that multiplies the regularisation term which was assigned a value of 0.001 instead of 0.0001.

Next, multiple input columns were selected and then transformed before fitting the model on them. These consist of three textual columns and four numeric columns. The first textual column included the tweets from the train set, the only preprocessing applied to this column were demoijisation and the normalisation of encodings. The second textual input column consisted of the previously mentioned customised POS tags, which were derived from the demojized tweets. A column containing lemmatised demojised tweets was added as well. The textual inputs were vectorised and scaled for optimal interpretation by the model. The demojised tweets and lemmas were vectorised by a CountVectorizer. This vectorizer performed an analysis of lowercased characters within an n-gram range of 2 to 3 on a maximum amount of features equaling 3500. The POS tag column was vectorised by a TfidfVectorizer that performed an analysis of words with an n-gram range of 1 implementing an nltk tokenizer on a maximum amount of features equaling 8 words. All of the textual input was scaled by use of a MaxAbsScaler which is suited to the expected level of sparsity in our data.

The first numeric input column includes the Afinn score for each tweet. In our case, the scores of each word in each tweet were added to one another to form one final score. Aside from this score, vaderSentiment was used to create the other numeric input columns, namely: 'pos', 'neg', 'neu' and 'compound'. "The 'pos', 'neu', and 'neg' scores are ratios for proportions of text that fall in each category (so these should all add up to be 1 or close to it with float operation)" (Hutto, Cited May 2022). The numeric inputs were scaled using a MinMaxScaler which normalises the num-

bers between -1 and 1. The SGD model was first fitted and tested on the training data so we could measure its F1 score, this score had a value of 0.71.

Logistic Regression The LR model using a logistic regression classifier as mentioned before, had a balanced class weight and random state of 42. The scoring was set to 'f1_macro' since our goal was to get that score as high as possible. We also opted for a 2-fold cross-validation generator to prevent overfitting and in doing so, make sure that the model would perform better when tested on unknown data. L1 regularisation worked in combination with the saga solver. The saga solver works faster than other solvers and thus is better suited to larger datasets. In addition, since all processors were used, the model should run fairly fast. The last parameter that differed from the default to be discussed, is the multi class parameter for which 'auto' was replaced by 'ovr' since we were dealing with a binary classification task.

Again, multiple input columns were selected and then transformed before fitting the model on them. The LR model was fitted on less input columns than the SGD model was, namely two textual columns and two numeric columns. The textual input included demojized tweets with their hashtag signs removed and a POS tag column based on demojized tweets without punctuation. They were both vectorized and scaled. The scaler used was the MaxAbsScaler. The tweets were vectorized by means of a TfidfVectorizer that performed an analysis of words within an n-gram range of 1 to 2 implementing an nltk tokenizer on a maximum amount of features equaling 3500 words. We chose to remove the hashtags, since they sometimes do not really add value. The POS tags were vectorised by a CountVectorizer analysing words with an n-gram range of 1 while implementing the same tokenizer on a maximum amount of features equaling 100 words. The vectorizers were not very different from their defaults since gridsearches pointed out the basic parameters as optimal, based on the general F1 score. The numeric inputs were the Afinn score for each tweet and the vaderSentiment based 'neg' column, both were scaled with a MinMaxScaler. The LR model, like the SGD was first fitted, then tested resulting in an F1 score of 0.71.

SVM1 The SVM1 model had a c-support vector classifier with a balanced class weight and random state of 42. The probability was changed from the default to 'True', so the SVM1 could be used in the ensemble later on. The regularisation parameter, resulting from a gridsearch, was 0.1 and the kernel type to be used in the algorithm was 'linear' as opposed to the standard, which is 'rbf'.

The numeric input columns used for this model are the Afinn score, the 'pos' score and the 'compound' score. They were scaled by use of a MinMaxScaler. The other input columns were demojised tweets without punctuation and POS on the basic tweets which were both vectorised by a TfidfVectorizer. The TfidfVectorizer for the tweets analysed lowercased characters within an n-gram range of 2 to 3 and with a maximum frequency of 3200. The vectorizer for the POS tags analysed words with a maximum frequency value of 8 and an n-gram range of 1. When tested, the model obtained an F1 score of 0.72.

SVM2 The SVM2 model contained a nu-support vector classifier with a balanced class weight and random state of 42. The probability was changed from the default to 'True' for the same reasons as the previous SVM. This model's parameters were all equal to the default. This model was initially made, simply to try an extra model within the ensemble and was meant to be adjusted, but it ended up having the largest influence on the ensemble and the highest F1 scores the ensemble obtained were all achieved by including this model. Even when trying to improve it, the default seemed to be the best option.

Almost all input columns, their scaling and their vectorisation were the same as the SVM1 except for the textual input, the tweet column used consisted of the demojised tweets with their hashtags removed. These tweets were vectorised and transformed on a lowercased character basis by a TfidfVectorizer with a maximum frequency parameter of 3500 and an n-gram range of 2 to 3. The POS tag vectorizer only differs from the one in the SVM1 model in that the n-gram range for the POS tags in this model ranges from 2 to 3 and in that it analyses characters as opposed to words. When tested, the model obtained an F1 score of 0.72.

Ensemble The ensemble model used all four of the abovementioned models as its estimators within a VotingClassifier. These models were chosen based on trial and error, after fitting over 30 ensembles, this combination of models and weights ended up being the best. The VotingClassifier im-

plemented soft voting which means that it predicted the class label based on the argmax of the sums of the predicted probabilities and is recommended for an ensemble of well-calibrated classifiers, which we had. Hard voting, its alternative, yielded worse results. The weights for each model were manually set: the SGD model, the LR model and the SVM2 model had a weight of 3 whereas the SVM1 model had a weight of 2. On the train subset, the model gained an F1 score of 0.74. This was clearly a better overall result than any single traditional model we made could have returned, so forming the ensemble model ended up being a profitable choice.

3.3.2 Neural Classification Experiment

5 RoBERTas The first model runs on the tweet column for both train and test data – roberta_tweet. For the second model all hashtags and cases of @USER are removed – roberta_hashtag_tweet – and the exact same is true for the third model, only this time the data is tokenized using the SpaCy tokenizer – roberta_token_tweet. The fourth model shares the same preprocessing steps as the third with an extra step; emojis are replaced with their textual equivalent – roberta_token_demojize. Finally, for the fifth model all punctuation is removed from the content of the tweet column and is then lemmatised – roberta_lemma.

Beyond the preprocessing, every RoBERTa model is the same; the model is a classification model from the simpletransformers[1] library and is set to run a maximum of 6 epochs with a batch size of 64 and a learning rate of 1e-5. At first, we altered the max sequence length to be larger than the maximum length of a tweet, with the idea to have the model be capable of handling inputs of every size. Still, we ended up using the default value of 128, because it garnered better results. In addition, an early stopping was included with a delta of 0.01 – the improvement over the best evaluation loss which is necessary for a checkpoint to be considered better – and a patience of 3. Through the use of 'evaluate_during_training' (at 83 steps) these values seemed to result in an optimal decrease and stabilisation of training loss and validation loss after running the model on the roberta-base from huggingface[2].

Majority Vote All 5 RoBERTas each return a prediction for each tweet: Offensive or non-offensive. Through the majority vote, this means a tweet is only labelled as offensive – or non-offensive – when 3 or more RoBERTas label it as such. In theory, this sounds advantageous, since it allows us to apply 5 models instead of 1. However, after getting the results through the OLID train and test data it was very important to consider whether the hard majority vote was actually worthwhile. As can be seen when comparing Table 2 and Table 3, the RoBERTas with the highest macro average also had the lowest recall of offensive data. After the majority vote, however, while the macro average remains equally strong, the recall of offensive data increases. This happens at the cost of the recall of non-offensive data. However, less is lost than is gained through the recall of offensive data. Thus, we consider the majority vote to be worthwhile.

	OLID (train)	
	Accuracy	Macro avg F1 score
roberta_token_demojize	0.82	0.80
roberta_token_tweet	0.82	0.80
roberta_hashtag_tweet	0.81	0.79
roberta_tweet	0.81	0.79
roberta_lemma	0.80	0.78
majority vote	0.82	0.80

Table 2: F1 scores for the best performing RoBERTas + Majority Vote

	OLID (train)			
	Precision		Recall	
	OFF	NOT	OFF	NOT
roberta_token_demojize	0.77	0.85	0.69	0.89
roberta_token_tweet	0.75	0.85	0.71	0.87
roberta_hashtag_tweet	0.70	0.87	0.77	0.83
roberta_tweet	0.70	0.87	0.77	0.83
roberta_lemma	0.69	0.87	0.76	0.82
majority vote	0.73	0.86	0.74	0.86

Table 3: Precision and recall for the best performing RoBERTas + Majority Vote

Exporting the Data Normally, the column with all the labelling decided after the majority vote would then receive the appropriate namings (OFF and NON) and id-assignment, but through a coding error made by Siebe in the final stages of the project, this instead happened to the very first column in the majority vote dataframe. This column was roberta_tweet. This means that the values assigned to 'majority vote' in Table 2 and Table 3 were determined when this error was not in place and still support our initial hypotheses, but that the results from our project onward do not reflect our original aspirations regarding the neural model.

[1] https://simpletransformers.ai/

[2] https://huggingface.co/

	OLID	Ruddit	Wikipedia	Total	TEXTGAIN
Our models					
Neural RoBERTa	0.803	0.713	0.912	0.808	0.504
Traditional Ensemble	0.753	0.661	0.874	0.760	0.434
Baselines					
Most frequent class	0.28	0.31	0.33	0.30	0.46
SpaCy BoW-model	0.70	0.66	0.86	0.73	0.54
Bert	0.807	0.703	0.896	0.803	0.472
HateBert	0.819	0.683	0.907	0.807	0.481

Table 4: Results of the baselines and our models

4 Results and Error Analysis

After being fitted on the same OLID training set, our traditional ensemble model and neural model (a single RoBERTa and not our majority vote) predicted the labels for the four different test sets assigned to us. Both models obtained the best macro averaged F1 scores for the Wikipedia dataset (AI, 2018), and performed the worst on the TEXTGAIN dataset, containing football tweets. More specifically, our traditional model obtained F1 scores ranging from 0.434 (TEXTGAIN) to 0.874 (Wikipedia). Our neural model obtained F1 scores ranging from 0.504 (TEXTGAIN) to 0.912 (Wikipedia). This is a rather large discrepancy, which may be caused by the fact that the TEXTGAIN dataset's golden standard only contained 14.73 per cent offensive labels (see Table 1). This means that the class distribution differed majorly from our training data, which made it hard for our models to predict the labels correctly.

Aside from the predictions based on the TEXTGAIN dataset, our models did not perform particularly well on the Ruddit (Hada et al., 2021) dataset. The results definitely improved in comparison to the TEXTGAIN data, but our traditional and neural model obtained an F1 score of respectively 0.661 and 0.713. It is quite surprising that our models performed less well (in comparison) on the Ruddit dataset, since all the borderline cases were excluded. However, this could be because the Ruddit dataset contains quite some posts that consist only of the text '[deleted]' and were labelled by the annotators as 'offensive'. The content of the original post was most likely deleted because it was offensive, but because our models could not access that content anymore, they labelled the posts as 'not offensive'. Aside from that, the format of Reddit posts differs from the format of tweets.

It is probable that our models performed best on the Wikipedia dataset (AI, 2018) because it contained only clear offensive or non-offensive instances. Even though the format of the data, namely Wikipedia data, was different from the tweets we trained on. This clear division made predicting the correct labels significantly easier for our model.

The models performed slightly worse when predicting the labels for the OLID (Zampieri et al., 2019a) test data, which is probably due to this data being less clearly offensive or non-offensive. Regardless, they still obtained F1 scores of 0.753 (traditional) and 0.803 (neural), most likely since the data was of similar format to the training data used and because of the similar class distribution. Overall, our models behaved as expected and with a mean of respectively 0.760 and 0.808, our traditional and neural model perform satisfactorily.

Our models also performed well in comparison to the given baselines. Table 4 shows that the traditional ensemble model made better predictions than the traditional baselines for every dataset except for TEXTGAIN. So it can be said that overall, our ensemble is the better traditional model, which is what we were trying to achieve. Though, as was to be expected, the traditional ensemble we made did not classify the labels of the test sets as well as the baselines for the neural models.

When looking at the the first three datasets and 'Total' column in Table 4, our single RoBERTa, barely outperforms the neural baselines. The model is still outdone by HateBert and Bert when predicting labels for the OLID dataset (Zampieri et al., 2019a). However, it is the front runner based on performance on the Ruddit (Hada et al., 2021) and Wikipedia (AI, 2018) datasets. These results are satisfactory, since HateBert and Bert are state-of-the-art hate speech detection models.

The TEXTGAIN dataset, again, appears to be a challenge of its own. Though this time, our model managed to cross the 0.5 margin and is second only to the traditional SpaCy BoW-model. Although our neural model was not our anticipated model, nor our preferred RoBERTa, it outperforms HateBert three times, while HateBert only outperforms ours once. Astoundingly, while setting out to test the advantages of a majority vote in hate speech detection, through an error, we end up showing the advantages of using a single RoBERTa model instead.

5 Discussion

As mentioned before every traditional model used in the ensemble had customised preprocessing steps. These ended up being the most optimal based on trial and error and the results of multiple gridsearches. Some of these 'optimal' preprocessing steps are confirmed by related research while others are not. For example, in most cases the TfidfVectorizers were deemed the best vectorizer for the tweet column and the lemma column. That was to be expected, since this vectorizer is well suited for operations regarding text data with a large vocabulary, as it gives a representation of the importance of each word or character in that document. However, the tweet column used in the stochastic gradient descent model seemed to be an exception. For the POS columns, the TfidfVectorizer initially seemed less useful since there is a limited amount of tags which all carry a similar meaning. Therefore, the CountVectorizer was often used to vectorise these columns. However, in some models a TfidfVectorizer did seem to yield better results.

The POS column's additional information to the models seemed to make them perform better. Based on the research done by Schmidt and Wiegand (2017), Fortuna and Nunes (2018) and Alorainy et al. (2018), we expected character n-grams to outperform word n-grams, but this was not always the case. Furthermore, we expected lemmas to have a positive influence on our results (Markov and Daelemans, 2021; Markov et al., 2021; Hee et al., 2018). However, overall, that did not seem to be the case.

The sentiment scores (vaderSentiment and Afinn) also added value to each model, though different vaderSentiment scores had different impacts on all the models. Other preprocessing used for the traditional models was only based on trial and error since related research made conflicting choices in their regard. As made clear in the description of the neural models, we used a different (often selfmade) column from the dataset to fit each one. These columns were chosen based on the best F1 scores obtained. This seemed to be the best modus operandi.

6 Conclusion

The models we made performed well, even if one was not our preferred choice. While there is still room for improvement, like adjusting the neural majority vote to weigh in all models in a similar way, we can say we are proud of the results we obtained. An overall conclusion we came to was that a long process of trial and error is needed to figure out what works best for each model and that more preprocessing does not always lead to better results. It is hard to say if there will ever be a neural or traditional model that can perfectly detect hate speech, though it seems unlikely since humans have not found an objective way of discerning hate speech themselves. So, a first step to improve hate speech detection could be to go back to the basics.

References

Jigsaw/Conversation AI. 2018. Toxic comment classification challenge. https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/overview.

Wafa Alorainy, Pete Burnap, Han Liu, and Matthew Williams. 2018. The enemy among us: Detecting hate speech with threats based 'othering' language embeddings. *ArXiv*, page 1801.07495.

Pete Burnap and Matthew L Williams. 2016. Us and them: identifying cyber hate on twitter across multiple protected characteristics. *EPJ Data Science*, 5(11):1–15.

Tommaso Caselli, Valerio Basile, Jelena Mitrovic, and Michael Granitzer. 2020. Hatebert: Retraining bert for abusive language detection in english. *Proceedings of the Fifth Workshop on Online Abuse and Harms*, pages 17–25.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. *Proceedings of the International AAAI Conference on Web and Social Media*, 11(1):1–4.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of

deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

Paula Fortuna and Sérgio Nunes. 2018. A survey on automatic detection of hate speech in text. *ACM Computing Surveys*, 51(4):85:1–85:30.

Giuseppe Gambino and Roberto Pirrone. 2020. Chilab@ haspeede 2: Enhancing hate speech detection with part-of-speech tagging. *EVALITA Evaluation of NLP and Speech Tools for Italian-December 17th, 2020*, pages 165–170.

Rishav Hada, Sohi Sudhir, Pushkar Mishra, Helen Yannakoudakis, Saif M. Mohammad, and Ekaterina Shutova. 2021. Ruddit: Norms of offensiveness for english reddit comments. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 2700–2717.

Cynthia Van Hee, Gilles Jacobs, Chris Emmery, Bart Desmet, Els Lefever, Ben Verhoeven, Guy De Pauw, Walter Daelemans, and Veronique Hoste. 2018. Automatic detection of cyberbullying in social media text. *PLoS ONE*, 13(10):e0203794.

Clayton Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proceedings of the international AAAI conference on web and social media*, 8(1):216–225.

Clayton Hutto. Cited May 2022. Vader-sentiment-analysis. https://github.com/cjhutto/vaderSentiment.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, page abs/1907.11692.

Sean MacAvaney, Hao-Ren Yao, Eugene Yang, Katina Russell, Nazli Goharian, and Ophir Frieder. 2019. Hate speech detection: Challenges and solutions. *PLoS ONE*, 14(8):e0221152.

Ilia Markov and Walter Daelemans. 2021. Improving cross-domain hate speech detection by reducing the false positive rate. *Proceedings of the Fourth Workshop on NLP for Internet Freedom: Censorship, Disinformation, and Propaganda*, pages 17–22.

Ilia Markov, Nikola Ljubešić, Darja Fišer, and Walter Daelemans. 2021. Exploring stylometric and emotion-based features for multilingual cross-domain hate speech detection. *Proceedings of the Eleventh Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 1–11.

Anna Schmidt and Michael Wiegand. 2017. A survey on hate speech detection using natural language processing. *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*, pages 1–10.

Omar Sharif, Mohammed Moshiul Hoque, A. S. M. Kayes, and Raza Nowrozy et al. 2020. Detecting suspicious texts using machine learning techniques. *Applied Sciences*, 10(6527):1–360.

UN. Cited May 2022. Impact and prevention: Why tackle hate speech? https://www.un.org/en/hate-speech/impact-and-prevention/why-tackle-hate-speech.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019a. Predicting the type and target of offensive posts in social media. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1415–1420.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019b. SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval). In *Proceedings of The 13th International Workshop on Semantic Evaluation (SemEval)*.

Finn Årup Nielsen. 2011. A new anew: Evaluation of a word list for sentiment analysis in microblogs. *Proceedings of the ESWC2011 Workshop on 'Making Sense of Microposts': Big things come in small packages*, pages 93–98.

A Supplemental Material

Our code for this project can be found in the following GitHub repository: https://github.com/SiebeBrouwer/NLP_Group2_StillProcessing