SNLP SS19 – PROJECT REPORT

Team Members:

Mossad Helali **2571699**

Ahmed Sohail Anwari **2571606**

Rayhanul Islam Rumel **2576541**

Table of Contents

Introduction & Task Definition

Unfortunately, hate speech has always been an unavoidable aspect of internet platforms that allow and encourage free speech. Terms like *Keyboard Warriors* were coined to describe the category of people who continuously post aggressive comments or posts on Internet platforms. The problem with Internet hate speech is not only in that it offends said minority or target group, but it has also in some, not rare, cases lead to serious hate crimes. Therefore, automatically detecting hate speech, especially on social media platforms, plays a vital role in preventing such offences and crimes.

However, this problem involves many challenges to overcome. For one, the definition of hate speech is not agreed upon: are slurs always hateful? What about implied hate that the recipient might not even understand? Another challenge is understanding the social media jargon, or for our problem, what is called *Twitterese.* The social media contain numerous words that might seem gibberish to the uninitiated while, in fact, they are strongly indicative of hate speech.

In this report, we document our attempt at attacking the problem of detecting hateful tweets. We first describe our approach to the problem, then provide the experimental results and conduct an error analysis. We then propose an improved approach and show preliminary results, before writing out our conclusions.

Proposed Approach

In this section, we describe our proposed approach. As per the given requirements, our model should be a feature-based classifier that uses surface-level features. To comply with the requirements, we discard all neural-network-based classifiers, even though the state-of-the-art for NLP tasks is obtained using a form of a neural model either for the whole system or only for the feature extraction module. Consequently, we looked for papers on either sentiment analysis or hate speech detection that use surface-level features. Our method is mainly based on [1], where we adapted most of their features and used a similar classifier. We also used conclusions derived in [2] to select some feature categories, which we describe in the features subsection.

Preprocessing

Our dataset is split into train, dev and test sets, with 11420, 1000 and 1000 data points, respectively. The tweets are labelled either OFF indicating offensive or NOT indicating otherwise. The distribution of the two classes is even in dev and test sets, while in train set it approximately 30-70, with the 30% being OFF tweets. Indeed, we can notice at first glance that the problem, and hence the data, is biased towards the NOT class since most tweets on the Internet are not offensive (hopefully?). Hence, we noticed from our preliminary experiments, that the models are biased towards the NOT class. As a remedy, we only used the first 3400 NOT tweets to make the train set a 50-50 split. We realize that throwing data away is not a

recommended practice, but we left this part of the problem with the intent of returning to it if time allows. For preprocessing, we tokenize the tweets using the TwitterNLP tool. Since the tool struggles with multiple emojis in the same word, we run a simple script to split the emojis. Our script is not perfect, however, since emoji encodings are complicated. For example, some emojis are constructed using a combination of two or three characters/emojis. Examples include country flags and emojis of men/women with skin color, rather than the standard yellow.

Features

Our features are mainly based on [1] with some adaptation. To explain them more easily, we group them into five categories which we describe below. We append the feature vectors from all categories to get the final feature vector for each tweet, which is of length 16436. We apply no normalization or feature selection. The feature categories are:

1. Lexicons

   Lexicon datasets map words to their sentiment score. A positive score indicates a positive sentiment and vice versa. This annotation of words can either be manual or automatic. We used two automatically generated lexicon datasets provided by the authors of [1]. We refer to them as NRC and Sentiment140 datasets. Both datasets provide scores for unigrams and bigrams. For each of these datasets we calculate the following for each tweet for both unigrams and bigrams:

   A. The number of grams with a positive score.
   B. The number of grams with a negative score.
   C. The sum of the scores of all grams.
   D. The sum of the scores of grams with negative scores.
   E. The maximum score.
   F. The minimum score.
   G. The score of the last gram with a positive score.

   By grams we mean either unigram or bigram. If the gram is not in the dataset, we either ignore (cases A, B) or insert 0 (other cases). The final feature vector from this category is of length 28.

2. Word clusters

   In addition to tweet tokenization and POS-tagging, the CMU TwitterNLP tool also provides word clusters. The authors cluster all 217K words of their dataset into 1000 hierarchical clusters. We observe from the cluster that strongly negative words (i.e. slurs) are clustered together and compliment words have their own clusters as well. For each tweet, we indicate the absence or presence of each cluster in the tweet, i.e. a feature vector of ones and zeros with the length of the number of clusters. Upon checking the number of clusters in the provided dataset, we notice that there are 938 clusters rather than 1000. Therefore, the final feature vector of this category is of length 938.

3. POS tags
   The CMU TwitterNLP provides POS-tagging with twitter-specific tags, such as hashtags (#), mentions (@) and emoticons (E). For each tweet, we count the number of occurrences of each of the 25 different tags. The final feature vector of this category is of length 25.

4. N-grams
   For the papers we went through, we noticed their surface-level features almost always included the word or character n-gram features. In addition, more than one analysis, including [1] and [2] pointed out that n-gram features are the most important ones, giving the highest performance gains. However, in our case, our computational resources are limited, and for a project with such a relatively small scale, we required that the system should be runnable on a day-to-day machine in a reasonable amount of time (i.e. order of magnitude of minutes not days). This requirement was demanded also for the sake of reproducibility and testing by our wonderful (and demanding?) grading team. Therefore, we couldn't afford to use the word and character n-gram features the way they were used in the research papers we read, which are typically long-term projects. Consequently, we focused only on one group of n-gram features, which is character trigrams. [1] used both (1,2,3,4) word n-grams and (3,4,5) character n-grams. This would result in a feature vector of size of at least one million, which would take a few days to train on a day-to-day machine. Fortunately, [2] have conducted an analysis comparing word n-grams to character n-grams and concluded that character n-grams are more indicative for detecting hate speech. We use this conclusion and restrict our use of character n-grams to only trigrams. Again, this is for the sake of time efficiency; otherwise, we believe that adding 4-grams would also increase the performance. For each tweet, we indicate whether or not it contains any of the 15441 character trigrams. That is, we have a feature vector of length 15441 of ones and zeros. This feature acts as a sort of raw representation for the tweet.

5. Word structure
   Finally, in this category, we group some surface-level features that were used in [1]. These features are:
   - The number of words with all upper-case letters, (excluding @USER).
   - The number of contiguous sequences of '!' or '?' and whether the last token contains '!' or '?'.
   - The number of elongated words, i.e. words with one character repeated more than two times, e.g. 'sooooo'.

   The final feature vector of this category is of length 4.

Classifier

As was used by several papers on hate speech detection and sentiment analysis, including [1], we used a support vector machine classifier with the linear kernel. The reasoning is that it is a powerful classifier with only one hyperparameter to tweak, C. Another reason is that SVM is

robust against features with high dimensions. In addition, our preliminary experiments with other classifiers also hint towards this direction. The classifiers we tried include random forest, AdaBoost, KNN and logistic regression, which gave comparable results to SVM at best, after a lot of hyperparameter tweaking. However, SVM takes more training time in comparison.

Experimental Results

In this section, we demonstrate our results using the approach discussed in the previous section. The baseline requirement is to have a classifier that achieves at least 70% accuracy on the dev set. Our classifier barely achieved this requirement with an accuracy of 86% on train set and 71% on dev set. Table 1 shows the confusion matrices for the train and dev sets, respectively. We would like to remind that we are confident that adding character 4-gram features would easily improve the accuracy. However, this would drastically increase the time and space complexities of our system. This accuracy was obtained using the value of C=0.065 and the runtime for both feature extraction and model training is around 25 minutes on a day-to-day machine. In Figure 1, we show the effect of changing the hyperparameter C on the accuracy of both the train and dev sets. Note the slight overfitting our model suffers from at the chosen C value. In the next section, we shed some light on the different types of errors our model makes.

Train	Pred NOT	Pred OFF	Total
Actual NOT	3086	314	3400
Actual OFF	595	2805	3400
Total	3681	3119	6800

Dev	Pred NOT	Pred OFF	Total
Actual NOT	374	126	500
Actual OFF	164	336	500
Total	538	462	1000

Table 1: Confusion Matrices for Train and Dev Sets

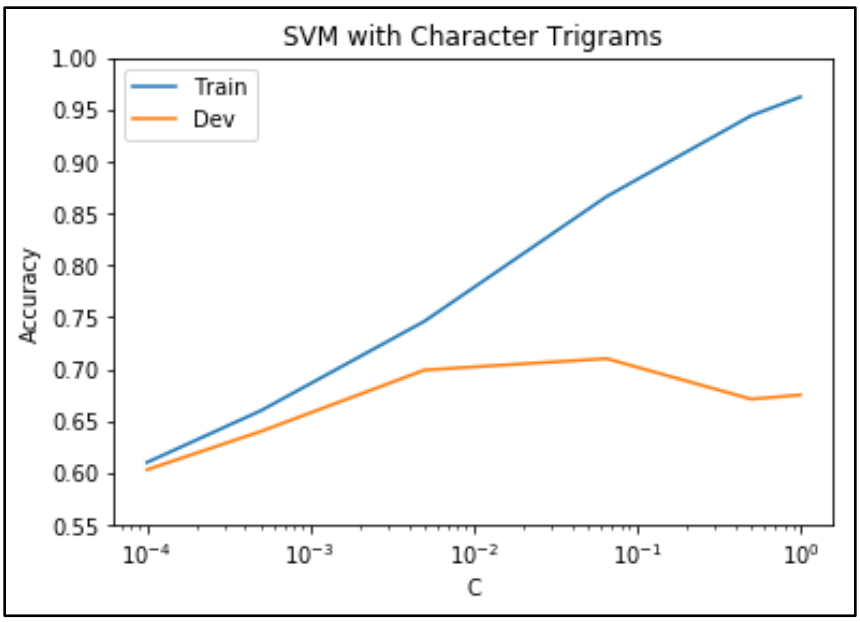

Figure 1: The Effect of Parameter C on Train and Dev Accuracy of Initial Model

Error Analysis

In order to conduct a proper error analysis, it is not only important to look at the cases where misclassification occurred, but also if the data was correctly labelled in the first place. In this section, we investigate some of the occurred errors and broadly categorize them by their reason of occurrence.

Before we delve into the analysis, it is important to note that we are forced to assume that all tweets are independent. This is not true in real data, as tweets can not only be stand-alone tweets, but can also be the replies to other tweets. This means that the context can be shared between the tweets. Replies can be deemed offensive if they support an original offensive tweet, even if the reply has no offensive term.

We also believe that in the training data, reply tweets are labelled based on the context of the original tweet as well, since there are cases where this is blatantly clear. We also believe that this is a weakness in the training data as tweet independence assumption should have been consistent throughout the process.

1. Mislabeled tweets in the training data

Mislabeled tweets can be sub-divided into two categories, as explained below:

- Contextual Labelling:

These are the tweets that seem to be the replies to other tweets and are labelled keeping the context in mind. Hence, without the knowledge of the original tweet, it is impossible to determine if the tweet is offensive or not. Most of them do not have any offensive terms in it

and hence under tweet independence assumption are not offensive. However, they are mislabeled in the data. Table 2 shows some examples of these tweets.

Tweet	Label	Prediction
@USER He is! 🌞	OFF	NOT
@USER She is thick though? Wtf 😕	NOT	OFF
@USER @USER Damn straight he is. Nothing but good vibes coming through November 100100100👆🏽□👆🏽□👆🏽□	OFF	NOT

Table 2: Examples of contextual labelling on tweets.

- Wrong label assignment:

These are the tweets in the training data which are clearly mislabeled. Tweets like these not only make the training harder but also decreases the accuracy of the model. Examples of these tweets can be seen in Table 3.

Tweet	Label	Prediction
@USER She is such a £ucking liar! You turned on the lights and did not notice it was not YOUR apartment? You had to go look at the number? WTF is she on?	NOT	OFF
@USER @USER @USER @USER @USER She is so calm & patient. That's great..keep it up. Don't mind people bc everyone has their own viewpoints on each aspects of life.	OFF	NOT

Table 3: Examples of tweets that are clearly mislabeled.

Due to these mislabeled tweets, not only is the training process negatively affected but the reported accuracy also suffers. This factor should be kept in mind when assessing the results.

2. Errors due to bad/limited features

Sometimes, the features we use prove to be too limited or inapplicable to correctly classify a certain tweet. A description of the features is given in the features section. Some examples of misclassification along with its explanation are given below

Tweet	Label	Prediction
@USER Carrey is looking more like a tranny every day?!	OFF	NOT

Here the offensive word 'tranny' is only slightly offensive in the NRC dataset (-0.08) and in Sentiment140 (-0.19) it is not enough to label this tweet as offensive.

Tweet	Label	Prediction
@USER She is a parasitic lump	OFF	NOT

Here the word “parasitic” is extremely offensive. However, this exact word is not present in both NRC and Sentiment140. It is important to note that the word “parasite” is present in NRC and has a very negative score (-4.999). This means that the performance of the model could have been improved if we refined the features further by stemming or lemmatizing words.

Tweet	Label	Prediction
@USER Ah .. classic antifa .. behaviour of trashpeople . =)	OFF	NOT

Here, the joining of the words “trash” and “people” results in it being a non-dictionary word. A stricter dictionary word policy (which makes sure that the lack of white space is catered as well) would have performed better.

3. Errors due to model optimization

Some errors occur not due to the features but due to the model itself, for example, the hyperparameters are not tuned well enough. These are the kind of errors that we expected our model to capture but it failed to do so. Some examples of these are given as follows.

Tweet	Label	Prediction
@USER @USER He is a pedo.	OFF	NOT
@USER He is so full of BS!	OFF	NOT

These errors seem to be avoidable with better optimization and tuning since the features are capable enough to mark them as offensive.

4. Unavoidable errors:

These are the type of errors that are too specific to be captured by a feature. Hence, in our view, they are unavoidable and will always result in misclassification. Usually, these errors occur because the tweet consists of offensive words that are not present in the dictionary because they are either made up or are from another language. An example is given below.

Tweet	Label	Prediction
@USER @USER achichincle lamebotas!	OFF	NOT

This tweet contains non-dictionary words and the model will always be unable to classify it based on features. It would always be as good as random classification.

Model Extension

In this section, we try to extend our model to further increase performance. Doing so, we give up the requirement that all features must be surface-level and allow to use more generalized features.

Word Embeddings

We remind the reader that our previous feature vector was of length 16436, out of which 15441 were only for character trigram features. We mentioned that this feature category can be thought of as a raw representation for the tweet. This representation suffers from a problem that affects both time and space complexities, which is sparsity. Assuming the worst case, that is a tweet of max length (280 characters) with no spaces and all trigrams are unique, the trigram feature vector has 280 - 3 + 1 = 278 ones and 15159 zeros, which is a clear waste of resources. These reasons exactly coincide with those that motivated the invention of word embeddings such as Word2Vec, FastText and GloVe. Therefore, we deemed it suitable to use word embeddings as features instead of character trigram features. We used GloVe embeddings that were trained specifically on a Twitter dataset [4]. We used the conclusions in [3], who tested different strategies of combining the embeddings for words in the Tweet. Based on their analysis, they concluded that summing the individual representation gives the best performance gain. We hope, therefore, by using word embeddings that we can get a reasonable performance gain.

Experimental Results

We demonstrate the performance of our improved classifier in this section. We used GloVe embeddings of size 200, the largest available. We removed the trigrams features and kept the rest of the features intact. Since the feature vector is of different size, the hyperparameter C needs to be optimized again. Figure 2 shows the performance of the SVM classifier on both train and dev sets with different values of C. We notice that indeed, there is a slight boost in accuracy by almost 3%, which is good, but below our expectations, nonetheless. However, the figure also indicates that the model shows no major sign of overfitting and can still improve on the final accuracy if C is tweaked enough. This can be easily noticed when comparing Figure 2 to Figure 1. The best value of C is 0.0005. We have stopped searching beyond the shown C values because the classifier takes too long to train (order of magnitude of hours) for C > 1. We provide our test set predictions based on this classifier.

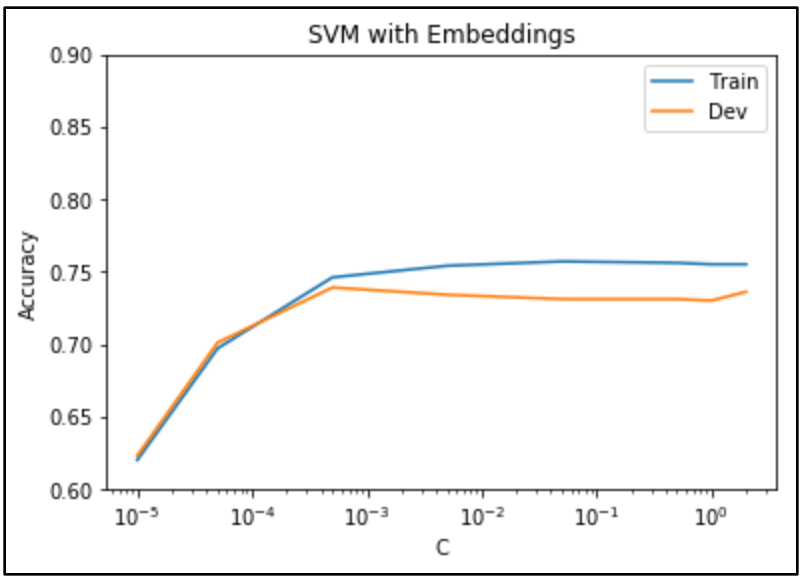

Figure 2: The Effect of Parameter C on Train and Dev Accuracy of Improved Model

Conclusion

In this report, we have detailed our attempt at tackling hate speech detection in tweets. We discussed the inherent difficulty of the task and touched on some of the faced challenges. We then presented our feature-based system that relies on surface-level features adapted from papers on sentiment analysis and hate speech detection. Our initial solution was able to achieve 71% accuracy on the dev set. Furthermore, we analyzed some of the errors our model made and conjectured about their root causes, which didn't always point at our model. We then proposed an improved model that uses word embeddings and showed that it was able to outperform our initial model, though by a small margin.

We learned from this project that although some NLP problems seem relatively easy at first, the intricates of natural language can prove them challenging. Therefore, a system working on tasks such as hate speech should be carefully designed and well-thought. Finally, all we can say after examining so many of the not-so-nice tweets to assess our model is:

"SMH ..." 

References

1. Saif M Mohammad, Svetlana Kiritchenko, andXiaodan Zhu. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets.arXivpreprint arXiv:1308.6242, 2013.

2. Waseem, Zeerak & Hovy, Dirk. (2016). Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter. 88-93. 10.18653/v1/N16-2013.

3. Petrolito R, Dell'Orletta F. (2018). Word embeddings in sentiment analysis. In: Proceedings of the Fifth Italian Conference on Computational Linguistics.

4. GloVe Embeddings Trained on Twitter Data: https://www.kaggle.com/jdpaletto/glove-global-vectors-for-word-representation/