FINAL PROJECT REPORT – GROUP 7

Detecting Hate Speech and Offensive Language on Twitter using Machine Learning

By Nitin Kumar

1. INTRODUCTION OF THE PROJECT

A key challenge for automatic hate-speech detection on social media is the separation of hate speech from other instances of offensive language. What constitutes hate speech and when does it differ from offensive language? No formal definition exists but there is a consensus that it is speech that targets disadvantaged social groups in a manner that is potentially harmful to them (Jacobs and Potter 2000; Walker 1994). In the United States, hate speech is protected under the free speech provisions of the First Amendment, but it has been extensively debated in the legal sphere and with regards to speech codes on college campuses. In many countries, including the United Kingdom, Canada, and France, there are laws prohibiting hate speech, which tends to be defined as speech that targets minority groups in a way that could promote violence or social disorder. People convicted of using hate speech can often face large fines and even imprisonment. These laws extend to the internet and social media, leading many sites to create their own provisions against hate speech. Both Facebook and Twitter have responded to criticism for not doing enough to prevent hate speech on their sites by instituting policies to prohibit the use of their platforms for attacks on people Copyright c 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved. based on characteristics like race, ethnicity, gender, and sexual orientation, or threats of violence towards others. Drawing upon these definitions, we define hate speech as 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. In extreme cases this may also be language that threatens or incites violence, but limiting our definition only to such cases would exclude a large proportion of hate speech. Importantly, our definition does not include all instances of offensive language because people often use terms that are highly offensive to certain groups but in a qualitatively different manner. For example some African Americans often use the term n*gga2 in everyday language online (Warner and Hirschberg 2012), people use terms like h*e and b*tch when quoting rap lyrics, and teenagers use homophobic slurs like f*g as they play video games. Such language is prevalent on social media (Wang et al. 2014), making this boundary condition crucial for any usable hate speech detection system . Previous work on hate speech detection has identified this problem but many studies still tend to conflate hate speech and offensive language. In this paper we label tweets into three categories: hate speech, offensive language, or neither. We train a model to differentiate between these categories and then analyze the results in order to better understand how we can distinguish between them. Our results show that fine-grained labels can help in

the task of hate speech detection and highlights some of the key challenges to accurate classification. We conclude that future work must better account for context and the heterogeneity in hate speech usage.

2. PROJECT PROBLEMS

A key challenge for hate-speech detection on social media is the separation of hate speech from other instances of offensive language. Lexical detection methods tend to have low precision because they classify all messages containing particular terms as hate speech and previous work using supervised learning has failed to distinguish between the two categories. We used a crowd-sourced hate speech lexicon to collect tweets containing hate speech keywords. We use crowd-sourcing to label a sample of these tweets into two categories: those containing hate speech as negative and those with neither as neutral. We train a multi-class classifier to distinguish between these different categories. Close analysis of the predictions and the errors shows when we can reliably separate hate speech from other offensive language and when this differentiation is more difficult. We find that racist and homophobic tweets are more likely to be classified as hate speech but that sexist tweets are generally classified as offensive. Tweets without explicit hate keywords are also more difficult to classify. Various machine learning approaches have been made in order to tackle the problem of toxic language. Majority of the approaches deal with feature extraction from the text. Lexical features such as dictionaries and bag-of-words were used in some studies. It was observed that these features fail to understand the context of the sentences. N-gram based approaches were also used which shows comparatively better results. Although lexical features perform well in detecting offensive entities, without considering the syntactical structure of the whole sentence, they fail to distinguish sentences ‘offensiveness which contain same words but in different orders.

Linguistic features such as parts-of-speech has also been used in hate speech detection problem, as shown in ; these approaches consist in detecting the category of the word, for instance, personal pronoun (PRP) and Verb non-3rd person. There have been several studies on sentiment-based methods to detect abusive language published in the last few years. One example is the work which applies sentiment analysis to detect bullying in tweets and use Latent Dirichlet Allocation (LDA) topic models to identify relevant topics in these texts.

3. KEY CONTRIBUTIONS

The review on the related work done in this field shows that the models trained after extracting N-gram features from text give better results . Also, the TFIDF approach on the bag of-words features also show promising results. Based on the review of features and the prominent classifiers used for text classification in the past work, we decided to extract n-grams from the text and weight them according to their TFIDF values. We

feed these features to a machine learning algorithm to perform classification. Given the set of tweets, the aim of this work is to classify them into two categories: Negative and Neutral.

4. MACHINE LEARNING

Our main goal for this project is to provide the public with open and simple visualizations about the proliferation of hate speech on Twitter using (behind the scenes but open to everyone to explore) machine learning algorithms.

4.1 DATA SET RESOURCE

For model training we used labelled data from Davidson that contains around 24k tweets that were classified as one of "hate speech" (0), "offensive language" (1), and "neither" (2). The labels were obtained through CrowdFlower; the original researchers kept the tweets that were labelled by at least 3 people. They were provided with our definition along with a paragraph explaining it in further detail. Users were asked to think not just about the words appearing in a given tweet but about the context in which they were used. They were instructed that the presence of a particular word, however offensive, did not necessarily indicate a tweet is hate speech. Each tweet was coded by three or more people. The intercoder-agreement score provided by CF is 92%. We use the majority decision for each tweet to assign a label. Some tweets were not assigned labels as there was no majority class. This results in a sample of 24,802 labelled tweets.

4.2 DATA SET PRE-PROCESSING

we observe data pre-processing: preliminary techniques that allow us to "clean up" tweets from all elements that create noise and that are not useful for sentiment analysis. Examples are punctuation marks, quotations (e.g. @mayasolovely), but also all those words that are very frequent in the text, but do not give a specific meaning to the single sentence such as conjunctions, pronouns etc. . This group of words is called stop words , and will be removed from any tweets.

Another important technique is that of tokenization which consists in dividing each sentence into an array of single words - or tokens - since this structure is useful for the following phases.

Finally, the last technique with regard to pre-processing is the stemming which consists in removing the suffix from each word, so as to obtain equal words from similar words that have the same meaning. As anticipated, these types of symbols do not give any added value for sentiment analysis, rather they cause noise within the texts and

therefore must be removed. We also remove urls and citations to other users, which are also not relevant.
To do this, regular expressions were used; the patterns specified within the text are recognized and deleted.
The next step is to split the text into tokens. To do this we used the nltk (Natural Language Toolkit) library, which provides many tools regarding the processing of sentences in natural language. To tokenize the tweets we used nltk's word_tokenize module: through the tokenize function, each tweet is transformed into an array of tokens.

In this phase we have also chosen to remove the stop words, to save a for loop on the whole dataset. Stop words do not give meaning added to the sentence, on the contrary they risk creating confusion within the dataset. The stop words list was always imported from the nltk library, and we added to this list another stop word, or "rt", which is very present in the tweet and corresponds to the word "retweet", which does not it is relevant for sentiment analysis.

The last phase of this pre-processing consists in stemming, or the practice of reducing all words to their root, thus removing the suffix. For example, words like "played", "playing", "playin '" are all reduced to their root "play". Again we used the nltk library through the SnowballStemmer module. There are other versions of stemmer, such as the PorterStemmer which works in a very similar way to the Snowball (the Porter makers themselves have admitted that the Snowball can be considered as an evolution), and the LancasterStemmer that we have chosen not to use because too much "aggressive".
We used a stemming process and not lemmatization, since the latter method, although it always produces words that exist in the vocabulary, is slower than the stemmer who does not bother to generate words with a complete sense.

4.3 FINDING THE BEST MACHINE LEARNING MODEL FOR THE PROBLEM OBJECTIVE

Firstly, We extract the n-gram features from the tweets and weight them according to their TFIDF values. The goal of using TFIDF is to reduce the effect of less informative tokens that appear very frequently in the data corpus. Experiments are performed on values of n ranging from one to three. Thus, we consider unigram, bigram and trigram features. The formula that is used to compute the TFIDF of term t present in document d is:

$$\mathbf{tfidf(d;\ t) = tf(t) * idf(d;\ t)}$$

Also, both L1 and L2 (Euclidean) normalization of TFIDF is considered while performing experiments. L1 normalization is defined as:

$$\mathbf{vnorm = vjv1j + jv2j + ::: + jvnj}$$

where n in the total number of documents. Similarly, L2 normalization is defined as:

$$\mathbf{vnorm = vpv12 + v22 + ::: + vn2}$$

We feed these features to machine learning models.

We consider three prominent machine learning algorithms used for text classification: Logistic Regression, Naive Bayes and Support Vector Machines. We train each model on training dataset by performing grid search for all the combinations of feature parameters and perform features validation. The performance of each algorithm is analyzed based on the average score of the accuracy for each combination of feature parameters. The performance of these three algorithms is compared.

Further, the hyperparameters of two algorithms giving best results are tuned for their respective feature parameters, which gives the best result. Again, different features cross validation is performed to measure the results for each combination of hyperparameters for that model. The model giving the highest cross validation accuracy is evaluated against the test data. We have used scikit-learn in Python for the purpose of implementation. Even we have used Textblob to find the sentiment analysis on our dataset. Textblob is a python library for processing textual data. Apart from other useful tools such as POS tagging, n-gram, The package has built-in sentiment classification. This is a so-called out-of-the-box sentiment analysis tool, and in addition to the null accuracy, We will also keep in mind of the accuracy we get from TextBlob sentiment analysis to see how my model is performing.

4.4 TRAINING AND TESTING THE MODEL

Here we chose to split the data into three chunks: train, development, test. I reference Andrew Ng's "deeplearning.ai" course on how to split the data.

Train set: The sample of data used for learning

Development set (Hold-out cross-validation set): The sample of data used to tune the parameters of a classifier, and provide an unbiased evaluation of a model.

Test set: The sample of data used only to assess the performance of a final model.

Another approach is splitting the data into only train and test set, and run k-fold cross validation on the training set, so that you can have an unbiased evaluation of a model. But considering the size of the data, we have decided to use the train set only to train a model, and evaluate on the dev set, so that we can quickly test different algorithms and run this process iteratively.

4.5 MODEL EVALUATION

As per model evaluation we are evaluating on the basis on TDIDF features with n-gram range so that we can cover all the accuracy and could check which can show better accuracy amongst all.

- Feature extraction on tdidf with ngram range with stopwords.

  First we are defining accuracy summary so that we could get accuracy result in which we are taking ngram range (1, 1) on TDIDF L2 Normalization through which we are getting different accuracy on different proposed features collected through features extraction. Then we'll run on ngram range (1, 2) on the same normalization. For this Evaluation we are taking all the classifier as per our midway report. During evaluating our model we are taking stopwords so that could achieve better accuracy.

- Accuracy check on different regularization parameter with solver.

  In this evaluation we are taking different regularization method and solver to achieve better accuracy.

- Accuracy check on smoothening alpha on naïve bayes.

  Now we are evaluating our model on alpha base for naïve bayes. Different alpha we are taking for different and better accuracy.

- We have analysed the different range of ngram on graph so that we can see the variation as well as we have plot a graph to check different variation of accuracy scores related with different algorithm.

- Finally we will evaluate our model on basis of final model so that we could check classification report as per test data.

5 ANALYSIS OF RESULT

At the starting phase we have analysis the sentiment through a sentiment tool "textblob" through which we get 58.51 % yield accuracy on validation set. This is a part though which we are checking our model accuracy from the processing text data. Then We have extract the features through Tf-Idf with ngram range so that we could evaluate our different algorithm. We have a comparison of different algorithm on different ngram range so that we can check accuracy with use of stopwrds.

The results of the comparative analysis of Logistic Regression (LR), Naive Bayes (NB) and Support Vector Machines (SVM) for various combinations of feature parameters is shown in TABLE I.

N-gram Range + TFIDF Norms	Accuracy		
	LR	SVM	NB
(1, 1) + L2	**94.15%**	**95.60%**	85.43%
(1, 2) + L2	93.30%	95.08%	**87.41%**
(1, 3) + L2	93.26%	95.08%	87.33%

TABLE I
COMPARISON OF THREE MODELS FOR DIFFERENT COMBINATIONS OF FEATURE PARAMETERS

TABLE I shows that all the three algorithms perform significantly better for the L2 normalization of TFIDF. However, SVM performs better as compared to Naive Bayes and Logistic Regression for L2 normalization. TABLE I shows that the best result for Support Vector Machine, **95.60%**, is obtained using ngram range up to one and TFIDF normalization L2. Similarly, Logistic Regression performs better for the same set of feature parameters achieving **91.15%** accuracy. Since both of these values are comparable, we tune both Naive Bayes and Logistic Regression, for the n-gram range up to three and TFIDF normalization L2.

TABLE II shows the results after tuning the Naive Bayes algorithm. We have considered the smoothing prior α for tuning. $\alpha \geq 0$ considers the features which are not present in the training set and in turn prevents zero probabilities. Technically, $\alpha = 1$ is called Laplace smoothing and $\alpha < 1$ is called Lidstone smoothing. Naive Bayes performs better for the α value 0.01 giving **91.04%** accuracy.

Alpha (α)	Accuracy
0.01	**91.04%**
0.1	90.60%
1	87.49%
10	83.41%

TABLE II
RESULTS AFTER TUNING NAIVE BAYES W.R.T SMOOTHING PRIOR *α* FOR THE FEATURES: N-GRAM RANGE 1-3 AND TFIDF NORMALIZATION L2

TABLE III shows the performance after tuning the Logistic Regression algorithm. Here, we have considered the regularization parameter C and the optimization algorithms (solvers) – liblinear, newton-cg and saga – for performance tuning. The model with settings C = 100 and solver liblinear gives the best accuracy 95.1%. Comparing the best accuracy for Naive Bayes and Logistic Regression, we conclude that Logistic Regression performs better. Therefore, we evaluate Logistic Regression on test data with the settings: n-gram range 1-3, TFIDF normalization L2, C = 100 and optimization algorithm liblinear.

Regularization C + Solver	Accuracy
10 + liblinear	**95.20%**
10 + newton-cg	**95.20%**
10 + saga	**95.20%**
100 + liblinear	95.04%
100 + newton-cg	95.00%
100 + saga	95.00%

TABLE III
RESULTS AFTER TUNING LOGISTIC REGRESSION W.R.T REGULARIZATION PARAMETER C AND VARIOUS OPTIMIZATION ALGORITHMS (SOLVERS) FOR THE FEATURES: N-GRAM RANGE 1-3 AND TFIDF NORMALIZATION L2

TABLE IV shows the accuracy comparison on different range of ngram as compared to different algorithm. From graph, we can see including bigram and trigram boost the model performance both in count vectorizer and TFIDF vectorizer for SVM and for every case of unigram to trigram, TFIDF yields better results than count vectorizer. At the final result we get a UNIGRAM range for SVM scores more than **95%** accuracy.

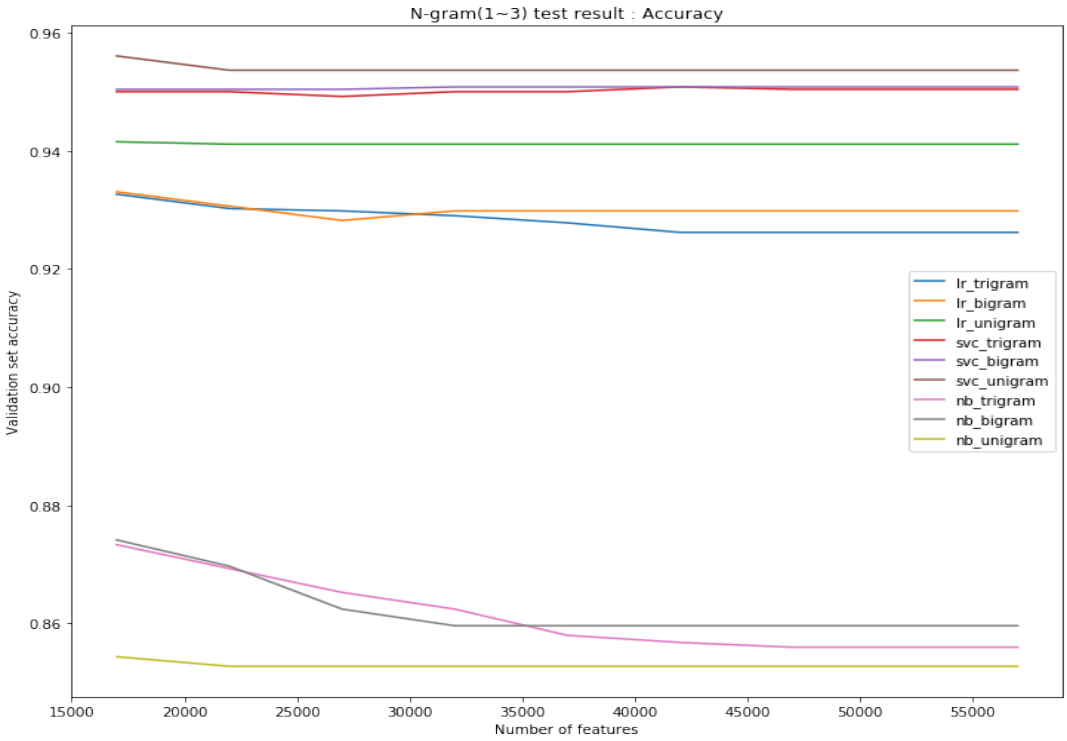

TABLE IV
RESULTS AFTER COMPARISON NGRAM RANGE ON DIFFERENT ALOGITHM

TABLE V shows a performance comparison on different model accuracy. For some range of features there are modification of accuracy but then it is showing a straight line which means accuracy showing an equal result for all different range of features. Here we observed that SVM has better performance accuracy amongst all algorithm.

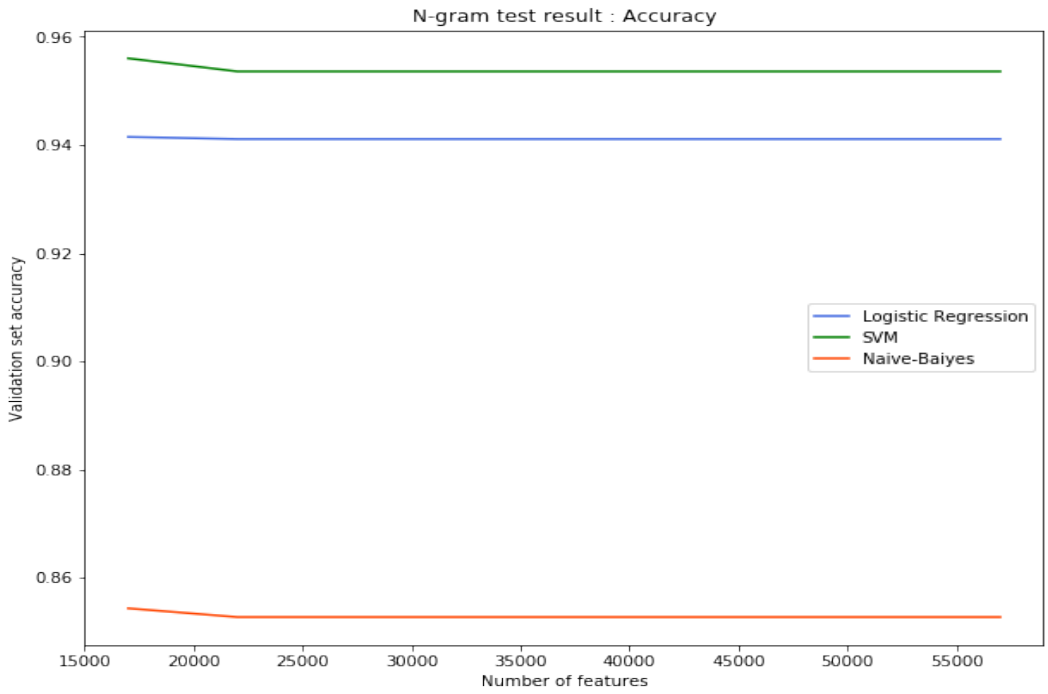

TABLE V
RESULTS AFTER COMPARISON ACCURACY ON DIFFERENT ALOGITHM

It is observed that the Precision for negative text is relatively low, 85%. This means that 15% of the tweets that are actually neutral have been misclassified by the model. Also, the precision for the Neutral class is 98%, which signifies that 2% of the tweets that are either negative or offensive have been classified as hateful. On the other hand, the recall for Neutral class is 97%, which is significantly better.

	Precision	Recall	F-Score
Negative	85%	91%	88%
Neutral	98%	97%	97%
Average	96%	96%	96%

TABLE VI
CLASSIFICATION SCORES OBTAINED AFTER EVALUATING THE FINAL LOGISTIC REGRESSION MODEL ON TEST DATA

Finally, we proposed a solution to the detection of hate speech and offensive language on Twitter through machine learning using n-gram features weighted with TFIDF values. We performed comparative analysis of Logistic Regression, Naive Bayes and Support Vector Machines on various sets of feature values and model hyperparameters. The results showed that Logistic Regression performs better with the optimal ngram range 1 to 3 for the L2 normalization of TFIDF. Upon evaluating the final model on validation data for Logistic Regression, we

achieved 96% accuracy. It was seen that 4% of the negative tweets were misclassified as hateful. This problem can be solved by obtaining more examples of offensive language which does not contain hateful words. The results can be further improved by increasing the recall and precision for the negative class.

Also, it was seen that the model does not account for neutral words present in a sentence. Improvements can be done in this area by incorporating linguistic features.