University of Ljubljana
Faculty of Computer and
Information Science

Offensive language analysis

Mateo Kalem, Peter Horvat, Jernej Zupančič

Abstract

In this report we will present an overview of our task where we conducted an analysis into the recognition of offensive language, otherwise known as hate speech, racism, sexism, etc. We will take a look at a few datasets which contain messages and posts which contain harmful language and some different approaches which use natural language processing and machine learning to detect harmful speech.

Keywords
Offensive language, Hate speech, Natural language processing, Machine learning

Advisors: doc. dr. Slavko Žitnik

Introduction

Hate speech is a public form of speech that expresses hate or encourages violence towards a person or group based on a factor that is not common to both parties such as race, religion, sex, or sexual orientation. In the last few years we have seen its rise, especially in online forums or social media pages such as Twitter, Facebook, Reddit, etc. The detection of such speech is a critical challenge in *NLP (Natural Language Processing)*. Despite the existence of numerous *NLP* algorithms and methods dedicated to detect hate speech and offensive language, the results are sometimes very low. In our report we will take a look at some of these approaches and evaluate their effectiveness at detecting offensive language.

Related works

In this section we will take a look at some of the related works in this field of hate speech detection and NLP. The first related work is a paper [1] in which the authors combined multiple deep learning methods (such as RNN, CNN, ...) to detect hateful speech in tweets. Other works which we found are not contained in papers or articles, rather they are public Github repositories in which the authors implemented algorithms related to the problem which we are studying. In one of these projects [2] the author uses the NLTK toolkit for hate speech detection, the dataset which he uses contains tweets with harmful text within them.

In another related paper [3] the authors used Context Aware Models to detect hate speech in the comment sections of Fox News articles. With this method they reported an accuracy of about 75 % and a precision rate of 55 %.

Algorithms

A brief description of some of the projects we looked at:

- Classification of Offensive tweets, in which authors divided the algorithm in to three sub-tasks: Offensive language identification, Automatic categorization of offense types, Offense target identification
- Offensive Language Detection, authors here have divided their application in to different stages, they firstly pre-processed the text to remove any unnecessary stop words, emojis, mentions, urls and all kinds of noise, then the text is tokenized using TF-IDF vectorizer, and lastly this project used 6 different classifiers.
- Machine Learning and NLP methods for Automated Hate-Speech and Offensive Language Detection, here the authors focused mainly on tokenization. In particular, they used NLP methods to create feature spaces including weighted TF-IDF scores, TF-IDF matrix, N-grams, sentiment scores, and dependency-based features.
- HateSonar: Hate Speech Detection, is a python library which already includes pre-trained models. Just like in the other projects authors here have firstly cleaned the text, then computed the weights then classified it with scikit-learn.

Datasets

For the purpose of completing our task we must also specify some datasets which are related to our topic. One of the most widely used datasets is a set which contains Twitter hate speech annotations [4]. This dataset contains nearly 17.000 entries, which is perfect for our project. Another dataset which we intend to use is the Fox News Comments Corpus [5].We briefly mentioned this dataset when we described the article in the Related works section. Here are some of the datasets we found that we are lookin forward to use them in the testing phase.

- Fox News User Comments [5]
- Hate Speech Twitter annotations [4]
- Predicting the Type and Target of Offensive Posts in Social Media [6]
- Automated Hate Speech Detection and the Problem of Offensive Language[7]
- Hate Speech Dataset from a White Supremacy Forum [8]
- NLP and CSS 2017: Second Workshop on Natural Language Processing and Computational Social Science at ACL 2017 [9]
- Hateful Users on Twitter [10, 11]
- A Benchmark Dataset for Learning to Intervene in Online Hate Speech [12]
- MLMA Hate Speech [13]
- Hate Speech Twitter Datasets [14, 15]
- A Quality Type-aware Annotated Corpus and Lexicon for Harassment Research [16]
- Offenseval Twitter dataset [17]

Since the datasets size and quality is of grave importance we collected as many datasets as possible. This way we can select the ones that will fit our requirements the best. If we will get to a point where we will need a even larger dataset, we will attempt to merge the best datasets. By comparing the results of the individual datasets and the merged dataset we will be able to conlcude if there was any deterioration in the results.

Initial idea

The initial idea of our project is to take a few different NLP methods and compare their accuracy when it comes to detecting hate speech. We also wish to compare them on different datasets. For this purpose we must also find many more datasets which originate from different forums and social media pages. These ideas are subject to change along the course of the 2nd semester.

Methods

In this section we will decribe the methods with which we created our offensive language detection algorithms.

TF-IDF based method

The first method which we implemented uses the TF-IDF approach. It is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents. It's main use lies within NLP, where it is useful for text analysis and for scoring words in machine learning algorithms. TF-IDF is calculated by multiplying two metrics: the term frequency and the inverse document frequency. The formula for calculating TF-IDF looks like this:
$tfidf(t,d,D) = tf(t,d) * idf(t,D)$, where:
$tf(t,d) = log(1 + freq(t,d))$ and
$idf(t,D) = log(\frac{N}{count(d \in D; t \in d)})$. Where t is the term, d is the document and D represents the set of documents. Now we will follow up with the steps of our method. We ran this method on the Offenseval Twitter dataset. This dataset was used in a offensive language detection challend, and as such, was divided into three subtasks. These are:

- A: Offensive languge identification.
- B: Automatic categorization of offense types.
- C: Offense target identification.

Removing noise from data

The first step in this method is to remove any and all types of noise from the data. We defines noise as URL's, user tags, slashes, apostrophe's, etc. We removed such noise from our data. Then we converted all of the text to be in a lowercase format.

Tokenization

After cleaning the data we started with tokenizing the data. Tokenization common task in NLP. It is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords. In our case we tokenized our tweets into words, with the word_tokenize function from the NLTK library in Python.

Removing stop words

When tokenization is done we proceed with removing stop words from the data. Stop words are the most common words in any existing language. For the purpose of analyzing text data and building NLP models, these stopwords might not add much value to the meaning of the document, which is why they must be removed. Stop words in this case are words, such as: the, is, in, to, at, etc. To detect such stop words we used NLTK's built-in stop word corpus for the english language.

Stemming and Lemmatization

For the final step in the preprocessing phase of our program, we do stemming and lemmatization. Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Lemmatization is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form. Both processes are important in the text preprocessing stage of any NLP method. Our program uses a stemmer and lemmatizer from the NLTK library. For the stemmer we took the Lancaster stemmer, it is known as one of the most aggressive stemming algorithms. For the lemmatizer we chose the WordNet lemmatizer. We chose WordNet because it is a large, freely and publicly available lexical database for the english language.

Classification

Finally, after putting our data through all of the previous steps, we convert our data into two vectors. One vector contains the data, the other contains the appropriate labels (offensive, not offensive, targeted, untargeted, ...). We then use this data for classification, where we use a 70/30 split for train and test data. We used quite a few classifiers, these are:

- Multiple variants of Naive Bayes (Multinomial, Gaussian, Complement, Bernoulli),
- k-nearest-neighbors,
- decision tree,
- random forest,
- logistic regression.

The results for the classifiers are presented as classification accuracy, and they are:

Classificator	Task A	Task B	Task C
Multinomial NB	0.803	0.441	0.765
Gaussian NB	0.441	0.481	0.529
Complement NB	0.725	0.454	0.735
Bernoulli NB	0.756	0.441	0.765
KNN	0.769	0.441	0.765
Decision tree	0.781	0.506	0.706
Random forest	0.859	0.441	0.735
Logistic regression	0.834	0.429	0.735

Table 1. The results of our classifiers.

Result analysis

In this section we will analyse the results in Table 1 and take a look at how some classifiers performed.

First, we will take a look at our Naive Bayes classifiers. We used different variations of Naive Bayes because we wanted to see which one would be best suited for our dataset. The best one in our case was Multinomial, since this type of NB is used to predict when a event will occur in a multinomial sequence. The worst one was the Gaussian, since it expects that the data is distributed normally. For each Naive Bayes classifier we set the alpha parameter (smoothing parameter) to 0.7.

The other classifiers did very well. The highest accurary came from the Random forest classifier, where we set the max depth of the forest to 50. We believe that the Random forest classifier was the best because it is built out of multiple decision trees and thus it can perform well with datasets that contain multiple classes (in our case tasks A, B and C).

We can see that task B was the hardest to classify. The idea of task B was to recognize whether an offensive tweet was targeted or not. In only one case did we manage to get 50 % for task B, that was with our Decision tree classifier. With this in mind, we are satisfied with how our classifiers performed regarding tasks A and C.

Hate-Sonar

Hate-sonar is the easiest way to implement a hate detection program. HateSonar allows you to detect hate speech and offensive language in text, without the need for training. You have only to fed text into HateSonar. It detects hate speech with the confidence score.

Hate-Sonar is using pretrained BERT (Bidirectional Encoder Representations from Transformers) based model[18]. The model was proposed in 2017 by researchers at Google AI Language. At the time it represented state-of-the-art language modelling, it was intended to be used for variety of NLP tasks, including Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and others.

BERT's main advantage over other methods is applying the bidirectional training of Transformer, a popular attention model, to language modelling. Other techniques look at the text either from left to right or right to left. The results of BERT language model, show there is an advantage to be made by bidirectionally training it. The model had a deeper sense of language context and flow than single direction language models. Bidirectionallity is enabled by using a technique named Masked LM (MLM).

Implementation

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words in a text. Transformer uses an encoder to read the text and a decoder which produces the prediction score for the task, but since BERT's main job is to generate a model, it only uses an encoder.

Difference between other directional models and a Transformer is that it doesn't read text sequentially, but instead processes entire sequence of words at once. This allows it to learn the context of words based on their surrounding text. Its input is a sequence of tokens, which are first embedded into vectors and then processed in the neural network. The challenge when using a neural network is how to define a prediction goal. Many models predict the next word in a sequence (When it rains it gets ___), this is a directional approach which inherently limits context learning. BERT uses two different techniques to mitigate this:

- **Masked LM (MLM)**
  This strategy is used to train the model by covering

	HateSonar	TF-IDF	Actual data
Hateful & offensive	4942	4100	3025
Neither	19841	20683	21758

Figure 1. Percentage of hateful speech in file: *tviter_cleaned_data.csv* as classified by Hate Sonar and second implementation of TF-IDF using POS tagging

	HateSonar	TF-IDF	Actual data
Hateful & offensive	2326	5066	4400
Neither	10914	8174	8840

Figure 2. Percentage of hateful speech in file: *tviter_dataset.csv* as classified by Hate Sonar and first implementation of TF-IDF

some percentage of the text with masked tokens. The model now has to predict the tokenised word using only the context provided by the other, non-masked, words in the sequence. Because of this the BERT model converges slower compared to other models.

- **Next Sentence Prediction (NSP)** This is the second strategy used to train the BERT language model. The model receives two sentences as input and learns to predict if the second sentence is the subsequent sentence in the original document. While training some percentage of sentences are really subsequent while others are just random sentences from the same database.

When training the BERT model, Masked LM and Next Sentence Prediction are trained together. The output is a sequence of vectors of size H, in which each vector corresponds to an input token with the same index.

Result analysis

We used the pre-trained BERT model built in to Hate sonar to compare results between different detection strategies. In two of the data sets that were used for evaluating other methods we were able to get this results:

	HateSonar	Actual data
Hateful & offensive	136	435
Neither	1392	1093

Figure 3. Percentage of hateful speech in file: *fox_cleaned_output.csv* as classified by Hate Sonar

	HateSonar	Actual data
Hateful & offensive	9356	12881
Neither	17989	14464

Figure 4. Percentage of hateful speech in file: *reddit.csv* as classified by Hate Sonar

In most of the cases the model worked pretty well but it couldn't match the performance of the other two methods. We suspect that the reason for this is that the other two strategies were trained on a specific text that slightly matched the text that we used to evaluate its performance. Although we couldn't find any duplicates, so it could very well mean that the pre-trained BERT model isn't well suited for evaluating this type of texts.

TF-IDF with POS tagging based method

In the next few paragraphs we will go through the process of our classification algorithm which uses *Part-Of-Speach Tagging*. The algorithm was initially presented on GitHub and is available on this link. To better compare it to the other algorithm that uses TF-IDF method we tried to use the same classifiers. The classifiers that we managed to adapt to make them work, and their performance analysis can be seen in the *Result Analysis* paragraph.

Reading the dataset

Our dataset is composed of around 25.000 tweets. To read them we use the function from the *sklearn package* called *test_train_split()*. Using this function we are dividing the dataset into two random sized subsets, one for training and one for testing.

Cleaning data (Tweets)

To achieve better results using our algorithm in combination with the dataset, we had to clean up the data. To do this we iterated over all of the tweets and did all of the following processing steps for each one of them. The first step was to remove all of the URLs and tokenize each tweet. By splitting each of the tweets we got a list of tokens for each tweet which we than used for further processing. Then, for each tweet we iterated over its tokens. To normalize our tokens we first set all of its characters to lower case and expanded all of the contractions in each tweet. Then we had to remove all of the punctuation marks and all short tokens that have less than characters which have no useful information for the sake of our algorithm. The last step was to remove all non alphabetic tokens and filter out the stop words, which again present no usable information in our case.

Part-Of-Speech tagging

Our algorithm is using TF-IDF but since we already explained this method and its implementation in the previous algorithm, we will now focus on *POS tagging*. Part-Of-Speach tagging is the process of marking up a word in a corpus as corresponding to a particular part of speech, based on definition and context. An example of this method is the classification of words as nouns, verbs, adverbs, etc. By using this information we can analyse our dataset and for add another layer of feature information.

Testing the same algorithm without POS tagging decreased the performance between 5%-10%. After doing some research we found that in fact the addition of POS tagging increases the performance of most classifiers with different models up to 10%.

In the algorithm we used the existing function *pos_tag* tagger function from the NLTK library. The function takes a sequence of tokens and tags each word, returning a list of tuples where the first element is the token and the second element is its tag. For example if we have the sentence: *NLP is a great class !*, we can clean up the data and tokenize it to get *[nlp, great, class]*. Then we use the *pos_tag* function to tag the tokens and get an output like *[('nlp', 'NOUN'), ('great', 'ADJ'), ('class', 'NOUN')]*.

Classification

We ran our classification algorithm using six different models, one of which was already used in the original code of the algorithm *(Linear SVC)*. The results in the following table 2 show the general accuracy score of each of the used models using our algorithm.

Classifier	Accuracy
Bernoulli NB	0.88
Decision Tree	0.86
Gaussian NB	0.38
K-Neighbours	0.76
Linear SVC	0.80
Random Forest	0.87

Table 2. Accuracy scores of our algorithm using different classifiers.

Next we present the Recall to show how accurately our algorithm can detect the three types of hate speech.

Classifier	Hate	Offensive	Neither
Bernoulli NB	0.21	0.95	0.82
Gaussian NB	0.60	0.33	0.53
Decision tree	0.29	0.93	0.81
Random tree	0.04	0.98	0.66
K-Neighbours	0.02	0.94	0.18
Linear SVC	0.07	0.99	0.18

Table 3. The recall values of different classifiers for the three classes.

Result analysis

As we can see in the table 3, the algorithm performs the best for detecting the 'Offensive' class and then we also see huge deterioration in performance when we compare it to the 'Hate' class. We tried using different parameters when using the provided models, but there was slim to none improvement in the performance. Analysing the general accuracy and recall of the individual classes led us to the following conclusions:

- The best classifier for detecting **hate speech** is the *Gaussian Naive Bayes Classifier*
- The best approach for detecting **offensive speech** is using the *Linear Support Vector Classifier*
- The best classifier for detecting **non hateful or offensive speech** is using *Naive Bayes classifier for multivariate Bernoulli model*
- In general the best performance across all classes are produced using the *Decision Tree modeling* We can see that the results for for detecting Offensive speech is similar for most classifiers except for Gaussian Naive Bayes classifier. After some research we learnt that using *Gaussian NB* when modeling we lose a lot of information in the process. The solution for this would be to combine the usage of *Gaussian NB* and the *Bernoully NB* [19, 20]. All of the produces confusion matrices and performance analysis can be seen in the *Confusion Matrices* and *Reports* directories respectively.

Implementation of algorithm for Slovenian language

For the final task in our paper we must construct a model that will try to identify offensive language in Slovenian datasets.

Firstly we were tasked with gathering data for this task. We managed to find two datasets which we can use:

- Slovenian Twitter hate speech dataset [21]
- Dataset and baseline model of moderated content FRENK-MMC-RTV [22]

The first dataset contains only the annotations and ID's of the tweets. We will have to build a Twitter scraper if we want to use the data from this source. After succesfully collecting all of the data we will then proceed to build our model for the slovene language.

Currently, we have decided to use the Transformers [23] library for Python, which is a state-of-the-art NLP processing tool. The library is already pretrained on 100+ languages, all we have to do is construct a pipeline with which we can succesfully detect offensive language. We will try to construct it with the knowledge we have gained from building our previous NLP methods. We must also test our Transformers pipeline on some english datasets, so we can firstly verify that the pipeline works before putting Slovenian data into it.

Conclusion

TODO: Write a conclusion for the report after finishing the final task – TBC

References

[1] Nicolò Frisiani, Alexis Laignelet, and Batuhan Güler. Combination of multiple deep learning architectures for offensive language detection in tweets. *arXiv preprint arXiv:1903.08734*, 2019.

[2] Aman Saha. Hate speech detection. https://github.com/aman-saha/hate-speech-detection, 2018.

[3] Lei Gao and Ruihong Huang. Detecting online hate speech using context aware models. *arXiv preprint arXiv:1710.07395*, 2017.

[4] Zeerak Waseem and Dirk Hovy. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL Student Research Workshop*, pages 88–93, San Diego, California, June 2016. Association for Computational Linguistics.

[5] Lei Gao and Ruihong Huang. Fox news comments. https://github.com/sjtuprog/fox-news-comments, 2017.

[6] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. Predicting the Type and Target of Offensive Posts in Social Media. In *Proceedings of NAACL*, 2019.

[7] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pages 512–515, 2017.

[8] Ona de Gibert, Naiara Perez, Aitor García-Pablos, and Montse Cuadros. Hate Speech Dataset from a White Supremacy Forum. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 11–20, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[9] Akshita Jha and Radhika Mamidi. When does a compliment become sexist? analysis and classification of ambivalent sexism using twitter data. In *Proceedings of the Second Workshop on NLP and Computational Social Science*, pages 7–16, 2017.

[10] Manoel Ribeiro, Pedro Calais, Yuri dos Santos, Virgilio Almeida, and Wagner Meira Jr. ”like sheep among wolves”: Characterizing hateful users on twitter. 12 2017.

[11] Manoel Ribeiro, Pedro Calais, Yuri dos Santos, Virgilio Almeida, and Wagner Meira Jr. Characterizing and detecting hateful users on twitter. 03 2018.

[12] Jing Qian. A benchmark dataset for learning to intervene in online hate speech. https://github.com/jing-qian/A-Benchmark-Dataset-for-Learning-to-Intervene-in-Online-Hate-Speech, 2017.

[13] Nedjma Ousidhoum, Zizheng Lin, Hongming Zhang, Yangqiu Song, and Dit-Yan Yeung. Multilingual and multi-aspect hate speech analysis. In *Proceedings of EMNLP*. Association for Computational Linguistics, 2019.

[14] Mai ElSherief, Shirin Nilizadeh, Dana Nguyen, Giovanni Vigna, and Elizabeth Belding. Peer to peer hate: Hate instigators and their targets. In *Proceedings of the 12th International AAAI Conference on Web and Social Media*, ICWSM '18, 2018.

[15] Mai ElSherief, Vivek Kulkarni, Dana Nguyen, William Y. Wang, and Elizabeth Belding. Hate lingo: A target-based linguistic analysis of hate speech in social media. In *Proceedings of the 12th International AAAI Conference on Web and Social Media*, ICWSM '18, 2018.

[16] M. Rezvan, S. Shekarpour, L. Balasuriya, K. Thirunarayan, V. Shalin, and A. Sheth. A quality type-aware annotated corpus and lexicon for harassment research. arxiv. https://github.com/Mrezvan94/Harassment-Corpus, 2018.

[17] Marcos Zampieri, Preslav Nakov, Sara Rosenthal, Pepa Atanasova, Georgi Karadzhov, Hamdy Mubarak, Leon Derczynski, Zeses Pitenis, and Çağrı Çöltekin. SemEval-2020 Task 12: Multilingual Offensive Language Identification in Social Media (OffensEval 2020). In *Proceedings of SemEval*, 2020.

[18] Rani Horev. Bert explained: State of the art language model for nlp, 2018.

[19] Fred Foo. sklearn gaussiannb bad results [nan] probabilities, 2013. StackOverflow Q&A.

[20] Einar. how to explain low performance of naive bayes on a dataset, 2017. StackExchange Q&A.

[21] Petra Kralj Novak, Igor Mozetič, and Nikola Ljubešić. Slovenian twitter hate speech dataset IMSyPP-sl, 2021. Slovenian language resource repository CLARIN.SI.

[22] Nikola Ljubešić, Tomaž Erjavec, and Darja Fišer. Dataset and baseline model of moderated content FRENK-MMC-RTV 1.0, 2018. Slovenian language resource repository CLARIN.SI.

[23] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.