EECS 495 - Project Report
Word-embedding based offensive speech detection system

Mas-ud Hussain, Alan Wojciechowski, Arindam Paul

March 22, 2018

1 Introduction and Motivation

The rise of offensive language on the internet within the past decade has increased the need for an automatic offensive speech detection system that is both reliable and accurate. Many social media websites such as Facebook, Twitter, and Reddit offer users an open platform where they can post their thoughts and provide commentary on a variety of topics. With this ability, users can also inappropriately use offensive language and speech, which may also be directed at other users. Most websites offer a report system where other users can report offensive users to website moderators, but those systems rely on human intervention and takes time to process. By providing a system that can automatically detect offensive speech as it is posted, websites can curb the impact of offensive speech within their platform. With some countries such as the United Kingdom, Canada and France having laws that prohibit the use of offensive speech toward another person, an offensive speech classification system can help websites comply with these laws and help reduce the instances of offensive speech usage.

Since the definition of offensive speech can vary among different entities, defining what is considered offensive is critical when trying to develop a classification system. According to [3], speech or conversation may be offensive for a number of reasons:

- It can be a personal attack and insults or degrades another person
- It contains terms with a recent or historical meaning relating to a particular gender, race, sexual orientation, or other characteristic of a user or group of users
- It negatively characterizes a user or group of users

Traditional schemes to detect offensive speech/text use rule-based methods or keyword counts that depend on a predefined list of profane or offensive words. The idea is to have a prefixed "vocabulary" of bad words – i.e., words that are mostly used in offensive terms, and then mark the given speech/text as an offensive one if it contains any of these words. The problems with such approaches are two-fold: (1) The list of possible bad words is always changing (also, expanding) with time, i.e., new words come out and people use different sets of spelling for a single word, and thus it is never possible to have a list including all possible words; (2) Many sentences are showed to be non-offensive even if they contain "profane" words [5]. Similarly, sentences without any "profane" words can be offensive as well. Moving away from these rule-based schemes allows us to have a more diverse model that looks at other dependencies in order to better classify.

Another important challenge for an offensive language detection scheme is to ensure that both accuracy and precision are prioritized when evaluating the system. Having a low number of false positives is important for any automatic detection system, since the repercussions of mislabeling something okay as offensive is high (e.g., bad reputation and user satisfaction). For example, if

a Facebook user's post is mislabeled as offensive, then the particular user may feel hurt/offended, stop using Facebook, and tell others about this incident. Additionally, some conversations or speech can be considered non-offensive even though they contain profane words. An example of offensive speech within our dataset was *I'm not a bigot, I just hate all Muslims* and an example of non-offensive speech was *okay but how tf does louis sit like that he must have strong a** legs to think it's comfortable.*

These examples highlight the need for a smart system that does not use keyword count or a rule-based system. Ensuring that our models can accurately classify speech within these scenarios is crucial for their success. While previous works have focused on using traditional machine learning techniques and models [5, 6], we wanted to cover more breadth by using models that took advantage of different word embeddings and neural network architectures. We also wanted to take advantage of different word embeddings and try to apply different embeddings to various models in order to get the best accuracy. These models are further discussed in depth in Section 3. We present the codebase as well as datasets for the use of the community at `https://github.com/paularindam/offensiveTweetDetection/`.

2 Related Work

Word embedding systems discussed in [8, 9, 10] lay the groundwork for our classification models. These papers introduce models that learn from vectors of words by taking into consideration their occurrence and co-occurrence information. We explore the word embeddings introduced within these papers including word2vec models skip grams and CBOW, along with GloVe model in order to find the best embedding method for our task. We compare results across different embedding spaces and attempt to find the best performing embedding model.

3 Methodology

We used the libraries Keras and Tensorflow for deep neural networks, Gensim for word embeddings, and Scikit Learn for traditional machine learning models. We split our dataset into 70% training, 10% validation, and 20% test.

3.1 Pre-Processing

We first obtained our dataset from Automated Hate Speech Detection and the Problem of Offensive Language [5] which consisted of 14,509 examples which were labeled as offensive speech, hate speech, and non-offensive speech. Since we only wanted to detect instances of offensive speech, we curated the dataset and made it into a binary classification. This resulted in a balanced data set with 7235 examples of offensive speech, and 7274 examples of non-offensive speech. We used 1 to denote all offensive and hate speech, and 0 to denote non-offensive speech.

We converted the tweets into Latin from Unicode to get rid of input errors and also removed carriage return and newline characters.

3.2 Embeddings

Due to the small nature of our dataset, we wanted to test out pre-trained embeddings along with trained embeddings on our dataset. We gathered 27 B Twitter GloVe model across multiple dimensions varying from 25 [1] to 300 [2] along with a pre-trained Google word2vec dataset. We also trained word2vec on our 15,000 offensive speech dataset using skip grams and CBOW.

We also trained a word2vec model over a 60K twitter corpus which contained insulting words. The original data set contain 1.6M tweets which we curated to remove any tweet that did not

contain an insulting word. We used common swear words in order to obtain a filter for the corpus. We hypothesized that using a word embedding model that was more closely related to the domain of offensive language might perform better, thus would be useful to try out for our models.

3.3 Model Specifics

We used binary cross entropy loss for our loss function, along with Adam optimization. We used sigmoid activation for our neural networks, with relu for our Convolutional layers. We altered batch size to be 32, 64, and 128, and varied using dropout values from 0 to 0.4. Within our models, we varied the number of units in LSTMs 30 to 200, and used early stopping with patient 0 to 2. We also experimented with reversing word order and comparing it to the original word order accuracy. For our skip-gram models with negative sampling, we varied the amount of negative words from 0 to 20.

3.4 Traditional Models

Although deep neural network based models like Convolutional Neural Networks(CNNs) and Long Short Term Memory Networks(LSTMs) have achieved state-of-the-art accuracies on many standard text classification tasks, we wanted to explore traditional machine learning algorithms because our dataset was relatively small. We explored Naive Bayes classifiers (both Multinomial and Bernoulli), support vector classifiers, logistic regression, k-nearest neighbors, perceptrons, boosted trees, random forest and extremely randomized trees. The best accuracy was achieved for extremely randomized trees and random forests.

Although TFIDF vectorization performed best, we also explored different kinds of vectorizers such as Hashing Vectorizer as well as applying Latent Semantic Analysis reduction on the sparse matrices to reduce the dimensionality. As mentioned in Subsection 3.2, we also explored the use of dense word embeddings for traditional algorithms but they did not perform satisfactorily.

3.5 Neural Network Models

We explored several neural network algorithms during our experiments. This included 1-D CNNs with maxpooling, LSTMs (1 and 2 layer), CNNs with LSTMs and FastText. As it was a binary classification problem, we used binary cross-entropy as the loss function. We present the illustrations of some of these models in the Appendix.

3.6 Performance Comparison

Figure 1 presents the comparative analysis of the accuracy across the best performing models. Figures 2, 3, 4 and5 present the train-validation curves for accuracy and loss for best performing LSTM, 1-D CNN, CNN-LSTM and FastText models respectively.

4 Insights

Overall, all of our models using adam achieved an accuracy of between 88% and 92% using adam optimization. We explored other optimizers such as traditional sgd, rmsprop, adagrad or momentum but they usually performed worse. Using skip grams with and without negative sampling did not do better than CBOW, and our best model actually used CBOW. We observed no perceivable difference between using pre-trained embeddings and embedding that were training. Also, we observed no significant different between using batch sizes of 32, 64, and 128 in terms of accuracy.

Overall, embeddings only worked well with neural network models. Using word embeddings such as Word2Vec and GloVe with Ensemble classifiers such as Boosted Decision Trees, Random Forests

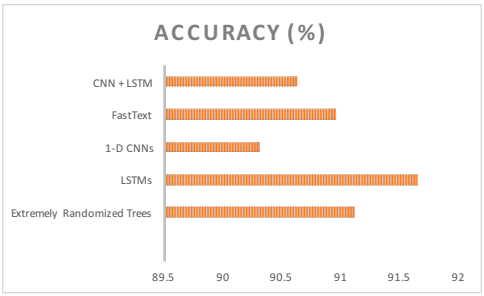
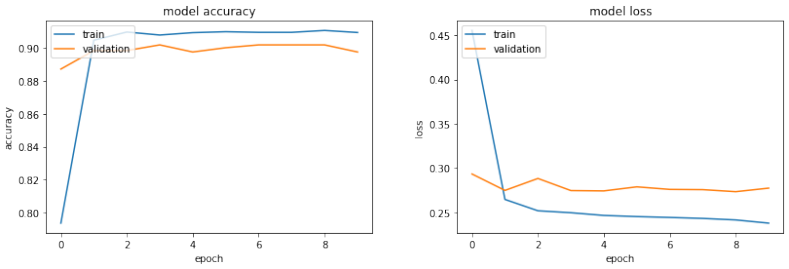
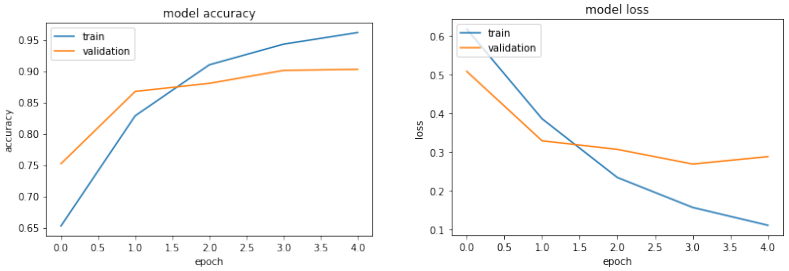


Figure 1: Comparison of accuracy across the best performing models



(a) (b)

Figure 2: Train and Validation Curves for model accuracy and loss for best LSTM model using Adam with Learning Rate=0.0005 using Continuous Bag of Words Embedding

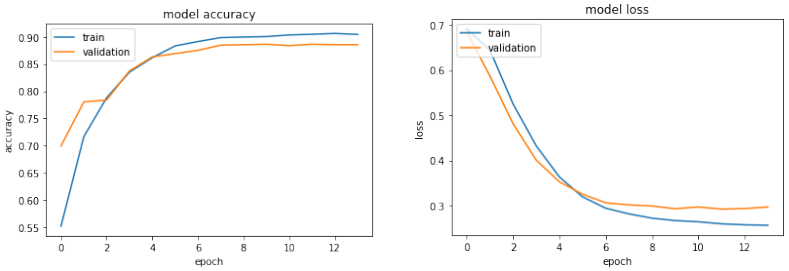


(a) (b)

Figure 3: Train and Validation Curves for model accuracy and loss for best 1-dimensional CNN model using Adam with Learning Rate=0.001 using Continuous Bag of Words Embedding

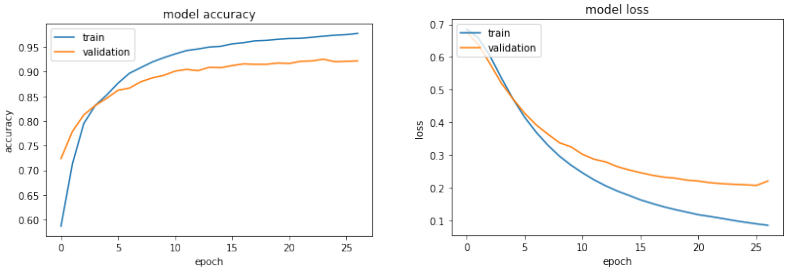
or Extremely Randomized Trees resulted in low accuracy. A reason why some of our neural network models may not have performed even better could be because of the size of our dataset. We only had about 15,000 examples from our dataset, which is small compared to other datasets used in tweet classification. Also the tweets themselves were sometimes very short, with some being only a few words long.

For most of the models, we used Adam which uses an adaptive learning rate. However, we modified the initial learning rates(α) from 10^{-2} to 10^{-4} with most models achieving best accuracy between 10^{-3} to 10^{-4}. For LSTMs, we found $5\text{x}10^{-4}$ to perform best. For FastText and CNNs, we achieved the best performance when using initial learning rate of 10^{-3}. For CNN and LSTM combined model, we achieved best performance at learning rate of $2\text{x}10^{-4}$.



(a) (b)

Figure 4: Train and Validation Curves for model accuracy and loss for best CNN+LSTM combination model using Adam with Learning Rate=0.0002 using Skipgram with no negative words.



(a) (b)

Figure 5: Train and Validation Curves for model accuracy and loss for best FastText model using Adam with Learning Rate=0.001 using Skipgram Embedding with Negative Sampling (number of negative words=20)

5 Future Work

We also tried to extend our work by using similar techniques as [6]. Although the work in [6] was for a different context, it is worth investigating the effects of the techniques used there. Noun, verb, and adjective "offensiveness" can be computed for different words using similar functions as [6] – where the authors computed "sexiness" instead. We prepared a list of offensive words (count=458 words) and used the retrieved 60k profane tweets do some initial experiments. One of the challenges we were facing is fewer training data. We plan to continue to work on it, and add the results in future versions. Given additional time, gathering more data and/or labeling data to add to our dataset would be another priority for our system. Davidov et al. [4] had found that using hashtags and smileys led to higher accuracy compared to just using words in the text. Exploring other user comments from other social media websites besides Twitter would also help make our models applicable to other platforms. We also want to analyze the performance of other models such as Hierarchical Attention Networks [11], Bidirectional LSTMs [12] and Recurrent CNNs [7] that have yielded state-of-the-art results on text classification problems. Extending our methodologies to classifying sub categories of offensive speech such as hate speech or defamation might also be worth examining. There are different implications for different kinds of speech, so we can learn different lessons when evaluating across different categories of offensive speech.

References

[1] glove.twitter.27b.25d — kaggle. https://www.kaggle.com/richardfourd/glovetwitter27b25d.

[2] stanfordnlp/glove: Glove model for distributed word representation. https://github.com/stanfordnlp/GloVe.

[3] Wikipedia:offensive speech. https://en.wikipedia.org/wiki/Wikipedia:Offensive_speech. Accessed: 2018-03-20.

[4] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd international conference on computational linguistics: posters*, pages 241–249. Association for Computational Linguistics, 2010.

[5] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. *arXiv preprint arXiv:1703.04009*, 2017.

[6] Chloe Kiddon and Yuriy Brun. That's what she said: double entendre identification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 89–94. Association for Computational Linguistics, 2011.

[7] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, volume 333, pages 2267–2273, 2015.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[9] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

[10] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[11] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[12] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. *arXiv preprint arXiv:1611.06639*, 2016.

Appendix

We present the illustrations for some of the neural network models we developed as part of this project.

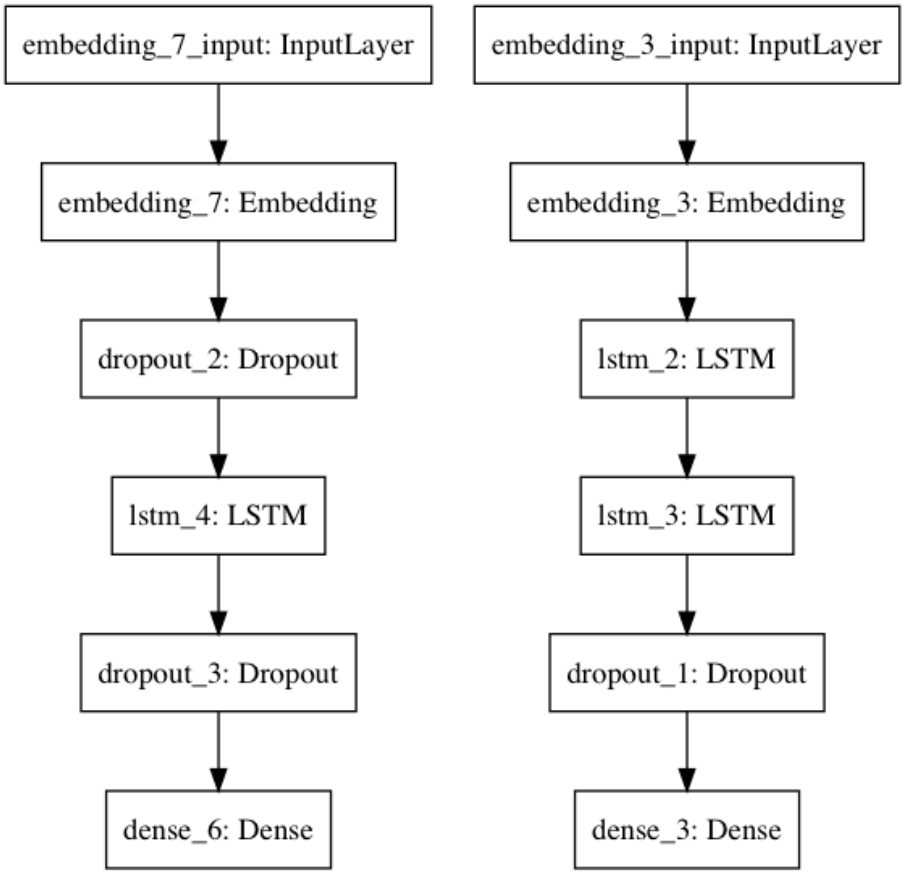


Figure 6: Illustration of the 1 and 2-layer LSTM models

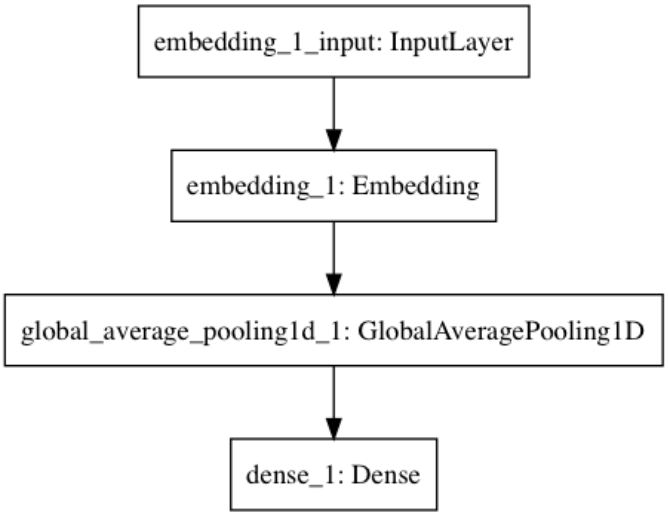


Figure 7: Illustration of the FastText model

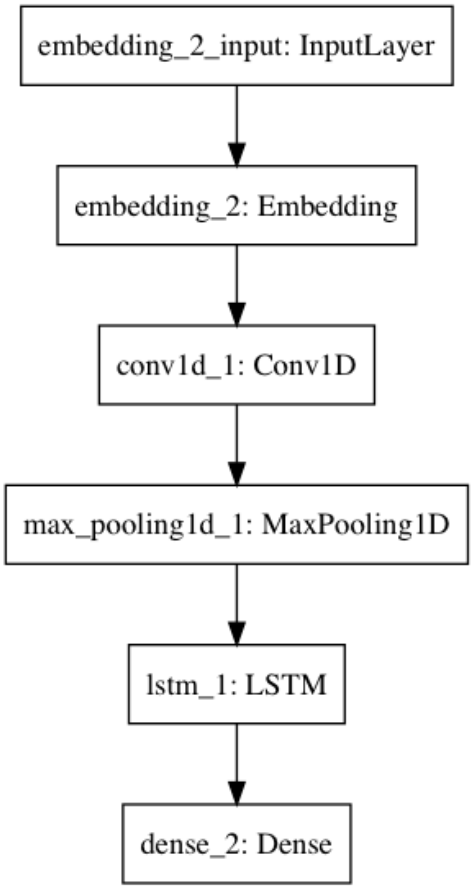


Figure 8: Illustration of LSTMs with CNN

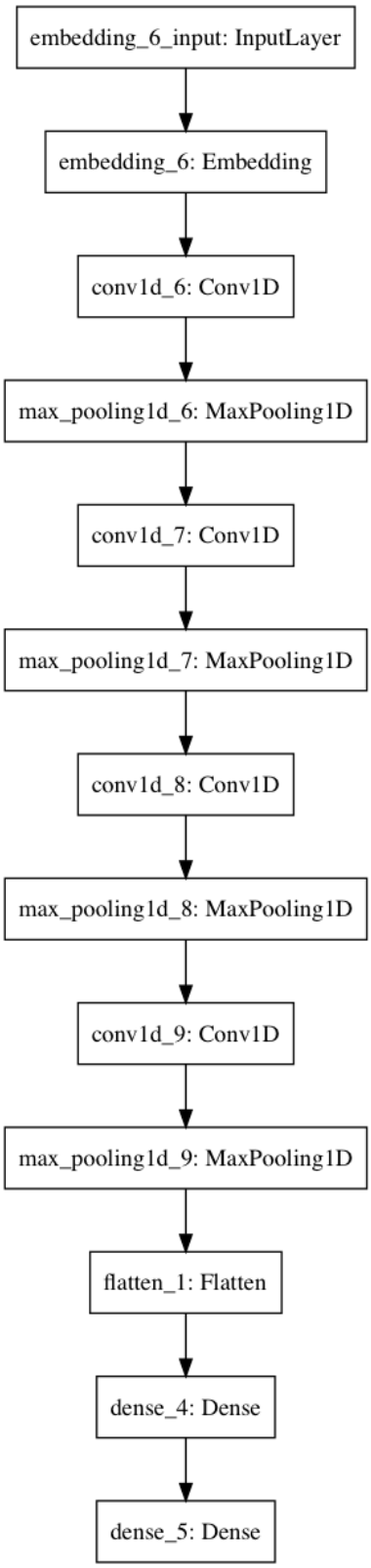


Figure 9: Illustration of the CNN with max pooling model