Offensive Tweet Classification

June 23, 2019

陳君彥, b04703091 b04703091@ntu.edu.tw
陳柔安, b04701232 b04701232@ntu.edu.tw
蕭法宣, b04705007 b04705007@ntu.edu.tw

Division of Work

陳君彥, b04703091:

- Generation and testing of TFIDF-Vectorized models.
- Generation and testing of domain knowledge based features.
- Creation of written report.

陳柔安, b04701232:

- Generation and testing of bi-LSTM.
- Generation and testing of CNN models.

蕭法宣, b04705007:

- Generation and testing of BERT model.
- Generation and testing of string based features.

1 Introduction

The goal of OffensEval competition on codalab by SemEval 2019 [1] is to tag a series of tweets with regards to their offensive nature. The competition is separated into 3 subtasks.

- Subtask a: Tag a tweet based on whether the tweet is considered "offensive" or not. Tags include "OFF" and "NOT"
- Subtask b: If a tweet is tagged as offensive in subtask a, further tag the tweet on whether it is offensive in a targetted entity, or not targetted. Tags include "TIN" and "UNT".
- Subtask c: If a tweet is tagged as targetted in subtask b, further tag the tweet on whether it is targetted towards an individual, towards a group, or towards something else. Tags include "IND", "GRP", and "OTH".

1.1 Dataset Analysis

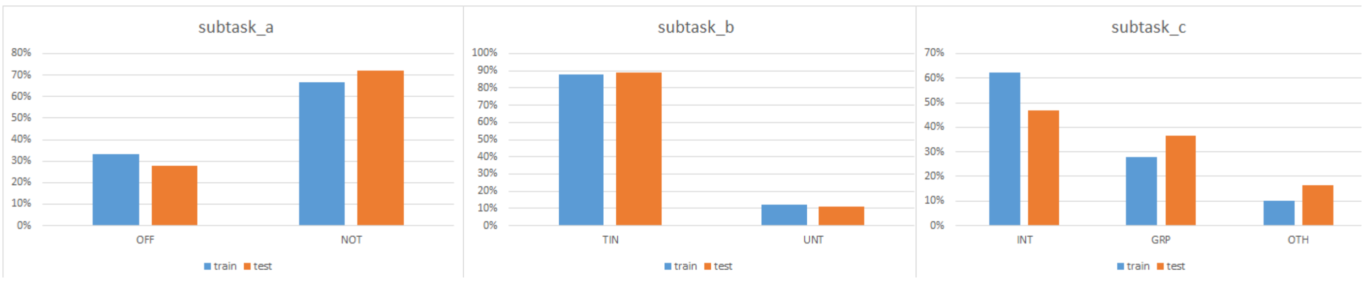

Figure 1: The percentage of each tag in the data set

Our training dataset consists of 13240, 4400, 3876 tweets for subtask a, b, c respectively, and the testing set has 620, 240, 213 respectively. We suspect the small size in the training data, coupled with the large disparity between the tag distribution as seen in Figure 1, contributes greatly to the difficulty in creating an accurate prediction model.

2 Preprocessing

2.1 Basic Text Preprocessing

We implemented the most common methods of text preprocessing, including the removal of stopwords, word tokenization, stemming, and lemmatizing. This was all done with the nltk package [2], using the punkt corpus for stopwords, Lancaster model for stemming, and WordNet model for lemmatizing.

2.2 Additional processing

We also experimented with different additional processing that is more tailored to our tweet based data, including testing between cased and uncased text, the inclusion or removal of ”tweet tags” (@USER) and twitter hashtags.

3 Methods

3.1 Domain based features

3.1.1 Subtask a

We considered what constituted as an ”offensive” tweet to the general public, and decided that the main criteria would likely be the inclusion of the use of profanity. We downloaded a list of common profanity words that Google uses as a filter list [3], and matched it in our tweets.

This method provides a very good baseline and close to our top performing models, all while being almost instanteous. We surmise that the use of a filter list is likely one of the most cost efficient ways to detect offensive languages.

3.1.2 Subtask c

We considered what it meant be be targetted towards a group, person, or issue, and arrived at the conclusion that if a tweet considered a noun that referenced a group or person, it is likely that it is targetted towards said entity, and if no such noun was found, it was non towards an entity in particular.

This idea was implemented through the use of ”Named Entity Relationship taggers” created by Stanford [4], as well as the SpaCy [5] tagger. This performed much less poorly than our more advanced models, but still provided a baseline that can be trained and predicted on the fly.

3.2 General Classification Models

We also tried to take a general approach to this classfication problem, as it relies somewhat less on specific natural language characteristics, and more on just pattern recognition. After preoprocessing, we transformed the tweets into vectors using TFIDF-Vectorizers, and ran the 3 subtasks through Naive Bayes, K-Nearest Neighbours, Decision Tree, Random Forest, and Linear Regression, all using the provided models found in the Sci-kit Learn [6] packages.

Of the tested models, Decision Tree and Linear Regression performed the best, though still somewhat short of our BERT method, while Random Forest performed slightly poorer, while also requiring much longer to train.

3.3 BERT Models

We also elected to use Google’s BERT [7] designed for NLP tasks. This model is the best performing method out of all of the ones we tried, but required large amounts of computation power, and a very long training time for each subtask. This leads to this method being viable for our rather small dataset, but could possibly be not worth it for larger, more complete data.

3.4 Neural Network Models

3.4.1 CNN

Based on a paper from WildML [8], we decided to take a non traditional approach and tried out the use of CNN in our prediction. The use of CNN with natural language leads to a behaviour similar to an optimized version of n-gram models, segmenting our tweets into semantical parts. Surprisingly, the model performed quite well, similar to general machine learning models, while requiring extremely low training time.

3.4.2 BiLSTM

We tried out the BiLSTM from the Keras package [9], which is often used in NLP modelling. However, our model performed poorly, sometimes worse than our general models, while also taking a long time to train, and we neglected to investigate further.

4 Experiments

The following are the experiments we did for each model used in our testing. Our computer ran a Intel 9400f CPU clocked at 4.1GHz x1/ 3.9GHz x6, running Windows 10 Home Edition. For all time related metrics mentioned, it can be assumed that the testing conditions were all roughly equivalent.

4.1 Domain Knowledge Models

Method	Training	Testing
Profanity Filter (a)	0.6887	0.7043
SpaCy NER (c)	0.3114	0.3082
Stanford NER (c)	0.2516	0.2992

Table 1: Performance of domain knowledge based models. Brackets denote subtasks.

As seen in Table 1, we achieved acceptable figures using a profanity filter on subtask a. This was done by downloading a list of profanity filter words used by Google, as well as adding in some extra words manually by looking at the training dataset and noting words with a high tfidf disparity between offensive and non-offensive tweets.

Subtask C performed rather poorly however, only beating a baseline of tagging all as ”IND” (0.210). As the two NER taggers we used are quite known for being rather performant, we suspect the issue is likely to be from the difficulty in labelling proper nouns, as well as the ”unclean” data found by mistypes, hashtags, or slangs often found in tweets.

4.2 General Classification Models

Method	Subtask a	Subtask b	Subtask c
MNB	0.4851	0.4703	0.2819
KNN	0.3900	0.4742	0.3507
DT	0.7281	0.5530	0.4396
RF	0.7213	0.5045	0.4379
LR	0.6982	0.5282	0.4346

Table 2: Performance of general machine learning models.

As seen in Table 2, our decision tree, random forest, and linear regression model performed much better in subtask c, while Naive-Bayes and K-Nearest Neighbours fell far short in both tasks.

While training, we noticed a large disparity between the validation accuracy for subtask b and c, somtimes upwards of 0.90 even with proper loss functions. We took a look at the dataset, and found that the training set and testing set differ substantially with regards to class imbalance as well as overall content, enough to cause our models to overfit greatly. We can see this trend continue to occur with other more sophisticated models.

Each model took around 30 minutes to train, and didn’t scale greatly with an increase in dataset size.

4.3 BERT

Method	Subtask a	Subtask b	Subtask c
No Preprocess Uncased	0.7774	0.6874	0.5731
No Preprocess Cased	0.7734	0.6870	0.5766
Preprocess Uncased	0.7875	0.6868	0.5726
Preprocess Cased	0.8023	0.6827	0.5986

Table 3: Performance of BERT models

As seen in Table 3, our BERT model returned the best results out of all the methods. We tested with using cased and uncased data, which gave us much better results with cased data on subtask a and c.

As most tweets are made through a phone app these days, which automatically capitalize the first letters of proper nouns, we suspect this leads a better recognition of named entities in our model, leading to a higher accuracy on subtask c.

A major problem with BERT however, is the massive amount of time required to train the model. With our relatively small data set size, it still took 4.5 hours to train on on subtask a, and 1.5 hours on subtasks b and c, with the time proportional to data set size. With some additional tuning, it could run slightly faster, though we realize that this method may not be very useful as of now, beyond the scope of a competition with no time constraints.

4.4 Neural Network Models

Dataset	Subtask a	Subtask b	Subtask c
CNN	0.613	0.928	0.442
BiLSTM	0.630	0.916	0.452
BiGRU	0.602	0.913	0.436

Table 4: Validation Set accuracy of NN model.

Dataset	Subtask a	Subtask b	Subtask c
CNN	0.738	0.655	0.472
BiLSTM	0.702	0.638	0.435
BiGRU	0.686	0.612	0.471

Table 5: Testing Set accuracy of NN model.

As seen in Table 4 and 5, our neural network based models also perform quite well, being on par for subtask a and c, and outperforming on subtask b. Of note is the large difference between the validation and testing accuracy on subtask b, which indicates a very high degree of overfitting on our model. As mentioned before, we speculate that this is due to the large tag and data discrepency between the two data sets. The creator of the OffensEval competition mentioned in their final report that they wish to tackle these two areas in future iterations, and we hope to see a better outcome should it happen.

Dataset	Subtask a	Subtask b	Subtask c
CNN	31s	10s	41s
BiLSTM	61s	22s	81s
BiGRU	56s	17s	81s

Table 6: Training time for each NN model.

Table 6 shows the training time for each of our NN models, which provides a very clear insight on the advantages that such models can provide. Compared to the 30+ minutes on general machine learning models, or hours long training on BERT, our neural networks can achieve an very acceptable rate of accuracy all while being incredibly fast to train. Coupled with the ease of performing "warm start" training on such models means that this method can be very cost effective, particularly in a online training method.

5 Conclusion

In this paper, we trained NLP models to recognize offensive language in tweets, as well as additional analysis on targetation of entities. We tested a wide range of models, including basic count and filter based models, general machine learning methods, and deep learning NLP models. Our best performing model, BERT, achieve a testing accuracy of 0.8023, 0.6827, and 0.5986 on the three subtasks, which would have attained ranks 9, 11, and 10 in the original competition. However, the long training time required for BERT means that we find it hard to recommend using it in a real world scenario, and would consider the use of neural network based models like CNN should such circumstances arise.

For possible future works, we would like to try this problem set again on a more complete data set, and perhaps do some additional tuning or leverage the power of GPUCompute to increase the performances of our models to acceptable rates. We would also like to further analyze the reason behind the overfitting issue on subtask b, and possibly come up with a solution to remedy such a problem.

References

[1] SemEval. (2019) SemEval - OffensEval: Identifying and Categorizing Offensive Language in Social Media. [Online]. Available: competitions.codalab.org/competitions/20011

[2] E. Loper and S. Bird, “NLTK: The Natural Language Toolkit,” in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ser. ETMTNLP ’02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 63–70. [Online]. Available: https://doi.org/10.3115/1118108.1118117

[3] Google. (2019). [Online]. Available: https://www.freewebheaders.com/full-list-of-bad-words-banned-by-google/

[4] J. R. Finkel, T. Grenager, and C. Manning, “Incorporating non-local information into information extraction systems by gibbs sampling.” in *Proceedings of the 43nd Annual Meeting of the Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling.* Association for Computational Linguistics (ACL 2005), 2005, pp. 363–370. [Online]. Available: http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf

[5] M. Honnibal and I. Montani, “spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing,” *To appear*, 2017.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[8] D. Brizt. (2015) Understanding convolutional neural networks for nlp. [Online]. Available: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[9] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.