UCLA CS269 Natural Language Processing Final Report CodaLab Competition: Identifying Offensive Language in Social Media

Jiachen Zhong *julightzhong10@cs.ucla.edu*
Xuan Hu *x1310317@gmail.com*

June 7, 2019

1 Introduction

Offensive language is very common in the Internet word, especially in online social media. The users can easily post offensive comments without any concerns by taking the advantages of anonymity of Internet environments. Therefore, many users may behave very different from what they usually do in real life and the abuse of online offensive language may cause serious damage in real world. Therefore, many online social media platforms have been investing heavily in detecting those kinds of language and trying to prevent them from leading physical damage. Usually, the detection can be done manually but, due to the huge amount of online information, it is unlikely to spend pure human power to finish the task. Therefore, using machine learning approaches to perform automatic offensive language detection is quite valuable to study. This kinds of task has attracted significant attention in recent years and several researches project have been done on this topic (Davidson et al., 2017; Kumar et al., 2018; Malmasi and Zampieri, 2018).

In this project, we want to compare and evaluate different kinds of machine learning methods on detecting the offensive language by participate the CodaLab competition (The CodaLab Team, 2019). We majorly focus on task A in the competition, which is to identify whether a sentence is offensive or not. We want to try our best to make the test accuracy as high as possible.

2 Data processing

The dataset we use is OLIDv1.0 (Zampieri et al., 2019) obtained from CodaLab website. Before applying different machine learning algorithms, we need to process dataset first. Here we remove all punctuation from OLIDv1.0 except "!" and "?". Considering we want to detect the offensive intention from the sentence, punctuation like "!" or "?" may contain some information related to human's sentiment, thus we remain them in our corpus and deal with them as words. Besides that, there are many emojis in our dataset. We utilize third party tool to convert them to words. We find this approach is critical to improve the model performance. We will discuss the detail in the later section. Also, we apply stemming and lemmatization on our data to normalize them. Here we adopt two methods to represent word after first step data processing. One of them is one hot coding. We use a vector to represent a word and the dimension of the vector is the number of word existed in the dictionary which we build from dataset. In order to remove the influence from the bias of whole corpus, we also apply Tfidf to these vectors. However, one weakness of one-hot encoding is we will missing the order of original sentences. One way to correct it is to use n-gram, but use one-hot representation is too large and computationally inefficient. Therefore, we try word embedding, which is GloVe (Pennington et al., 2014) in this project. We cannot make sure every word in the

dataset is included in the GloVe dictionary, therefore, we use the average vector of the GloVe to represent the missing words.

3 Experiments and results

We do several experiments on both two data representations and several machine learning models including Logistic Regression(LR), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), AdaBoost (AB), Gradient Boost Tree (GB), Multi-layer Perceptron (MLP), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN).

3.1 One-hot v.s. GloVe

Since models besides RNN cannot be trained on the inputs with flexible length directly, we use two methods to handle this problem. The first one is to directly add the word vectors together for a sentence, and the second one is to fix the input length of a sentence which means we drop the rest words whenever a sentence is longer than the pre-defined input length or we append zeros whenever the sentence is not long enough. We fine-tune the prefixed length, and find 10/12 are the best for most of the models on the data without/with emojis. Since the one-hot coding is very large and sparse, we do not try to concatenate one-hot coding representation. The detail result is presented in Table 1. For one-hot coding, we also apply SVD to reduce the dimension and improve the running speed. However, in most cases(except Random Forest), we could get a better accuracy without SVD. We think the reason is that data can remain more information without using SVD in this situation. Although SVD decreases the number of calculations a lot, it may also miss some useful information. For Glove vectors, from Table 1, we can obviously observe that adding those vectors together is a better way compared with dealing with them in a pre-defined length. Also, for traditional machine learning algorithms, one-hot coding without SVD can get a better performance overall compared with Glove.

Table 1: Performances of Models without emoji (w/wo SVD means with/without using SVD, add/-fixed means adding the embedding vector together/concatenate the embedding as a prefixed length, the Glove embedding dimension is 100. N/A means we cannot got meaningful result in reasonable time.)

Model	OneHot(w/wo SVD)	GloVe(add/fixed)
LR	76.63%/**80.35%**	77.30%/73.57%
SVM	75.81%/N/A	**77.88%**/74.04%
DT	74.88%/**77.80%**	74.38%/72.88%
RF	**75.93%**/72.09%	75.79%/74.16%
AB	75.23%/**80.15**%	77.88%/73.34%
GB	76.04%/N/A	**76.72**%/72.88%

3.2 Neural models

After initial attempt, we then try to fit those data on neural network. Since the dimension of one hot encoding is too high, we only use Glove vectors on this experiment. From the discussion in 3.1, we can know that it's better to add those vectors together rather than putting them in a fixed length window. The results in Table 2 shows that RNN can get a much better performance compared with MLP, CNN as well as other classical machine learning methods.

Table 2: Performances of neural models

MLP	CNN	RNN	GRU	LSTM
78.84%	79.88%	81.61%	84.05%	**84.63%**

3.3 Effectiveness of adding emojis

At first, when we process corpus, we delete all emoji from text since these signs are not word and they are not in our dictionary. However, since we are detecting offensive languages which may contain strong human emotion. We think emojis is an important tools to express meaning in online environment and it should be considered in the experiments. We

directly use a python package, called emoji Taehoon Kim (2019), to transfer the emojis into English words in order to keep the information of emojis. And we find the model can achieve a obviously better result by adding emojis. We firstly look at the situation on one-hot representation in Table 3. For one-hot format, the emoji does not give an obvious different on the performance. We think that because the one-hot representation is too sparse and can merely capture the sentimental meaning of the emojis. We

Table 3: Model performances between with and without emojis by using One-hot encoding, the two values in a cell refer to the result by using with-/without SVD. N/A means we cannot got meaningful result in reasonable time.

Model	with emojis	without emojis
LR	76.74%/80.23%	76.63%/80.35%
SVM	75.23%/N/A	75.81%/N/A
DT	74.53%/77.67%	74.88%/77.80%
RF	75.46%/72.34%	75.93%/72.09%
AB	76.28%/80.58%	75.23%/80.15%
GB	76.40%/79.53%	76.04%/N/A

also try to use GloVe on the dataset with emojis. The result in Table 4 shows a significant improvement by adding emojis. All models, including both classical machine learning methods and neural methods, get benefit from adding emojis. We believe there are two major reasons. Firstly, we think the offensive language is highly related to the usage of emojis. Secondly, by transferring the emojis to words and using GloVe, the models may gain more useful information to perform better on task.

3.4 Detail of RNN

After comparison between several different models, we find RNN models significantly outperform other types. Therefore, we here focus more on the RNN hyperparamters fine-tunes in order to get our best model. All the experiments are tested on the dataset with emojis.

Table 4: Model performance between with and without emojis by using 100 dimension GloVe embedding. Here all experiments besides RNN use the method of adding the embedding vector together

Model	with emojis	without emojis
LR	77.42% (+0.12%)	77.30%
SVM	78.11% (+0.23%)	77.88%
DT	74.39% (+0.01%)	74.38%
RF	76.94% (+1.15%)	75.79%
AB	78.46% (+1.74%)	76.72%
GB	79.51% (+1.63%)	77.88%
MLP	78.84% (+0.12%)	78.72%
CNN	79.88% (+0.58%)	79.30%
RNN	81.61% (+0.47%)	81.14%
GRU	84.05% (+1.74%)	82.31%
LSTM	84.63% (+2.09%)	82.54%

A sentence pass through the RNN will produce several hidden states according to the number of tokens. Each hidden state will carry useful information from previous tokens, thus, we may just use the hidden state produced by the token at the tail of the sentence to perform the classification. However, if a sentence is too long, the tail hidden states may loss important information of the beginning of the sentence. Therefore, in order to solve this problem, we may also sum up each hidden states of a sentence to perform the classification. We test which one is better by doing experiments shown in Table 5. Here, we test LSTM and classical RNN units, with 1 hidden layers, 128 hidden units, and no dropout. All models are unidirectional. The GloVe dimension is 50. In the project, all RNN experiments only use signal layer MLP to perform the final binary classification.

Table 5: "All" means using all hidden states, and "Tail" means use only the last hidden states.

RNN type	All	Tail
LSTM	**82.53%**	81.84%
RNN	**80.68**%	79.51%

As what shows in Table 5, the way of using all hidden states could achieve a better result. We also try different types of RNN units to find out the best one. Here, we focus on LSTM, GRU and classical RNN 3 types of RNN units. We present the result in Table 6 which indicates that LSTM beats GRU and classical RNN units. Another important factor for

Table 6: Comparison of different RNN units. Experiments setting:3 hidden layers, 256 hidden units, 0.5 dropout rate, bidirectional, use all hidden states, and 300 dimension Glove embedding

RNN types	LSTM	GRU	RNN
Result	**84.63%**	84.05%	81.61%

the performance of RNN models is the model capacity including hidden units size, number of hidden layers, bi/uni-directional, and the Glove embedding size. We firstly present the comparison between different hidden units sizes in Table 7. The result indicates that using suitable(neither too big nor small) hidden size may get a better result. We also try dif-

Table 7: Comparison of different hidden layers size and depth. Expriments setting: bidirectional LSTM, using all hidden states, 0.5 dropout rate, 300 dimension GloVe. For size experiments, the depth is 2. For depth experiments, the layer size is 128.

Size	128	256	512
Result	84.05%	**84.28**%	83.93%
Depth	1	2	3
Result	83.70%	**84.05**%	**84.05**%

ferent depths of the models shown in Table 7. Basically, it follow the intuition that a deeper model may get a better performance. Moreover, we compare the effect of bi/uni-directional structure to the result. The result is presented in the Table 8 which shows that the bidirectional structure dose give positive improvement to the result. Finally, we find models may achieve a better result if the word embedding layer is set to be trainable during the training, The result is shown in Table 8.

Table 8: Comparison of bi/uni-directional LSTM and trainable/untrainable GloVe. Experiments setting: 2 layers, 128 hidden units LSTM, using all hidden states, 0.5 dropout rate, 50 dimension GloVe)

LSTM	Unidirectional	Bidirectional
Result	82.42%	**82.65**%
GloVe	Trainable	Untrainable
Result	**82.77**%	82.65%

4 Conclusion

Finally, our best test accuracy is 84.63% which is achieved by using dataset with emojis and 3 layers bidirectional LSTM with 256 hidden units, 300 dimension trainable GloVe embedding, 0.5 dropout, and using all hidden states. Because of the limitation of computational resources and time, we did not preform too many fine-tunes, and we may get better accuracy by doing more hyperparameters searches. Overall, RNN is better than all other algorithms whose input is only a fixed length. We believe the reason is that RNN can capture more relational information than other algorithms whose input vectors are added up as a single vector, and it can learn more global information when the input vectors of other algorithms are concatenated as a pre-defined length. Moreover, we find that we may get a big improvements on all machine learning models we used in the projects by utilizing the emojis information in the dataset.

In this project, we have not tried the state-of-art language algorithms like transformer due to the time limitation. However, we do not believe that the key is on the capacity of model, since most of our algorithms can achieve over 95% accuracy on train dataset. The capacity of models are obvious enough for this task, thus, we believe the important factor is how to better process data(add more useful information) and perform suitable regularization methods.

References

Davidson, T., Warmsley, D., Macy, M., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. In *Eleventh International AAAI Conference on Web and Social Media*.

Kumar, R., Ojha, A. K., Malmasi, S., and Zampieri, M. (2018). Benchmarking aggression identification in social media. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*, pages 1–11.

Malmasi, S. and Zampieri, M. (2018). Challenges in discriminating profanity from hate speech. *Journal of Experimental & Theoretical Artificial Intelligence*, 30(2):187–202.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Taehoon Kim, K. W. (2019). Emoji package for python. `https://github.com/carpedm20/emoji/`, Last accessed on 2019-06-01.

The CodaLab Team (2019). Offenseval: Identifying and categorizing offensive language in social media. `https://competitions.codalab.org/competitions/20011`, Last accessed on 2019-04-20.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N., and Kumar, R. (2019). Predicting the Type and Target of Offensive Posts in Social Media. In *Proceedings of NAACL*.