Toxic Comment Classification - An Empirical Study

Manikandan R [1]*,Sneha Mani [2] *
[1] PESIT-Bangalore South Campus
[2]Rutgers University
manikandan.ravikiran@gmail.com
sneha25@gmail.com

Abstract

We present our work on detection of abusive language in online text, where abusive language is defined as "using, containing, or characterized by harshly or coarsely insulting language". While abusive language against any group may exhibit some common characteristics, we have observed that it is typically characterized by the use of a small set of high frequency stereotypical words making our task similar to that of text classification. More specifically we address detecting abusive language into six finer categories namely toxic, severe toxic, obscene, threat, insult, identity hate. In this report we describe our definition of abusive language, the corpus used, and mechanisms for classification. We describe pilot classification experiments in which we classify abusive language reaching x%,x%,x%,x%,x%,x%, (**put results here for each of the six categories** for each of the six categories.

1 Introduction

Hate speech, offensive language, sexism, racism and other types of abusive behavior have become a common phenomenon in many online social media platforms. In recent years, such diverse abusive behaviors have been manifesting with increased frequency and levels of intensity which are yet to be addressed. Recently, the sheer volume of public data dumps released by the social media platforms and complexity of detecting language abuse have alone increased traction among machine learning and Deep Learning community. While detecting hate speech is important .In most parts of the world, hate speech is protected by the fundamental rights that are present as part of the constitution,leaving internet commentators exercise this right in online forums such as blogs, newsgroups, Twitter and Facebook. A dilemma of "block the user" or "delete the post" rises in the mind of the social media content moderators. For example if a user A posts a comment saying "Dude! what the fuck is wrong with you" the comment can be deleted while if another user B comments saying "Hail!! Klu Klax Klan" it would require action beyond deleting a post, may be the user could be reported and blocked in the social media. As such classification of hate vs non-hate suffers from its inherent limitation requiring finer granular classification. As such in this report we present our work on **"Toxic Comment Classification"**.More specifically we present the systems developed for the task of detecting obusive language in a fine graned approach i.e. Six way multi class multi label classification problem.

Concerning, abusive language detection, which is inherently formulated as classification problem multiple works are done till date with extensive usage of deep learning (**?**) , Naive Bayes (**?**), SVM (**?**) and Tree based approaches (**?**). In this work, we developed systems using Gaussian Naive Bayes, Logistic Regression, K-Nearest neighbors, Decision Trees, Multilayer perceptron and Convolutional Neural Networks(CNN) in combination with word and character embeddings by adapting insights from multiple previous works.

The rest of the report is organized as follows. In section 2, we describe the dataset and preprocessing used. In section 3 and 4, we present the feature extraction and algorithms used respectively for this work. In section 5, we present our experiments and conclude with summary with some implications of future work in section 6.

*This is a preliminary report draft, detailed paper is on works. Work performed during weekends as a practice activity. Sneha Contributed in feature engineering and empirical evaluation. Manikandan concentrated on CNN and Perceptron development.

2 Dataset and Preprocessing

There are 159,571 comments in total. Out of which 143,346 comments are clean (all toxic tags are 0). There is an imbalance in the data set. Considering, a sample of 7000 clean comments in sample. All comments converted to lower case, lemmatization of words, leaky features have been removed(user ip address), converted words with apostrophe(can't to can not), removed stop words and tokenized words in comments.

We used following opensource tools 1) Stanford Core-NLP (Manning et al., 2014) 2) Keras (Chollet et al., 2015) 3) CNTK (Seide and Agarwal, 2016) 4) Gensim (Řehůřek and Sojka, 2010) 5) NLTK for preprocessing (Loper and Bird, 2002) 6) Scikit-learn (Pedregosa et al., 2011) for grid search 7) Glove (Pennington et al., 2014).

3 Model

In this section, we explain the algorithms and hyperparameters used for system development. More specifically, in section 3.1 we explain our CNN architecture for subtask 1 and in section 3.2 we show our CRF architecture for subtask 2.

3.1 Algorithms

For subtask 1, we focused more towards deep learning. Previous works (Yin et al., 2017) suggests that both Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) architectures has been successfully applied for various instances of text classification analysis at various level. With most of recent works (Zhang and Wallace, 2017) showing success of CNN, we developed a CNN architecture based on work of Kim (2014). The architecture developed in this work is as shown in figure 1.

3.1.1 Convolutional Neural Network

Our CNN architecture was derived from original works of Kim (2014) by using grid search over input channel size, number of convolution layers and number of filters. We use a multichannel model architecture with five input channels for processing 2-6 grams of input malware text. Each channel is comprised of the following elements:

1. Input layer that defines the length of input sequences.

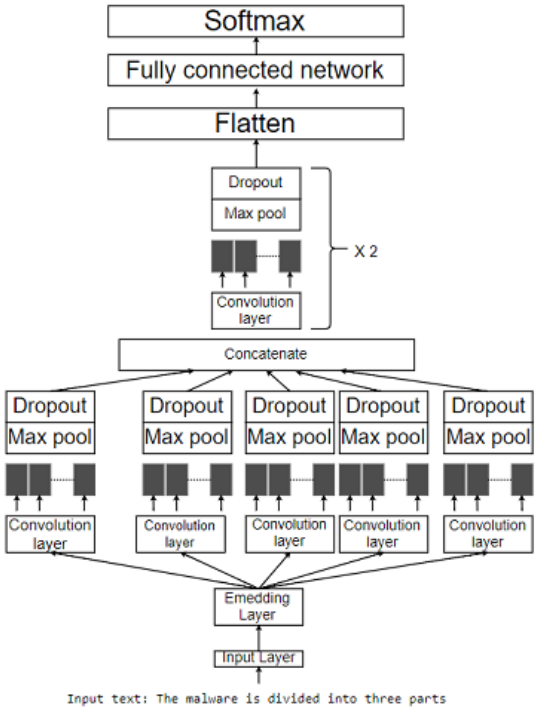

Figure 1: CNN Architecture

2. Embedding layer set to the size of the vocabulary and 100-dimensional real-valued representations.

3. One-dimensional convolutional layer with 128 filters and a kernel size set to the number of words to read at once.

4. Channel wise Pooling layer with pool size of 5 to consolidate the output from the convolutional layer.

Following CNN, we use a Fully Connected Neural Network (FCNN) to transfer the the concatenated feature map (600 dimension) to a probability distribution over the two class labels. The number of layers in FCNN is set to be 2. The first layer uses 128 units with a tanh activation function. The second layer produces the classification probability distribution over 2 units combined with a softmax activation function.

Further to handle overfitting we use regularization via dropout (Srivastava et al., 2014) with threshold of 0.25. Additionally, we also apply cost sensitive learning (Zhou and Liu, 2006) in order to balance the effect of the larger negative samples present in the training dataset. For each class, we assigned weight proportional to class frequency. We implemented the neural network model using

Feature	Value
Sentence pad length	1000
Dimensions of wordvectors	100
Number of CNN layers	8
Dimension of CNN layers	1
Number of CNN filters	128
Activation function	relu
Initialization function	Xavier
Number of FC layers	2
Dimension of 1st FC layers	128
Dimension of 2nd FC layers	2
Activation of Final layer	Softmax
Optimizer	Adam
Batch size	32
Max Epoch	10
Loss function	Cross Entropy

Table 1: Hyper parameters of CNN

Keras. We trained our networks using Adam optimizer (Kingma and Ba, 2014). All the hyper parameters are listed in table 2.

3.2 SVM

3.3 Logistic Regression

3.4 K Nearest Neighbor

3.5 MultiLayer Perceptron

4 Features

In this section, we explain various features that were used as part of model development activity.

4.1 TFIDF Feature

The algorithm builds on the following representation of sentences/documents. Each sentence d is represented as a vector d = (d(1); : : :;d(F)) so that documents with similar content have similar vectors (according to a fixed similarity metric). Each element d(i) represents a distinct word wi. d(i) for a sentence d is calculated as a combination of the statistics T F $(w_i; d)$ and $DF(w_i)$. The term frequency $TF(w_i; d)$ is the number of times word wi occurs in document d and the document frequency $DF(w_i)$ is the number of documents in which word w_i occurs at least once. The inverse document frequency IDF (w_i) can be calculated from the document frequency.

$$IDF(w_i) = \log\left(\frac{D}{DF(w(i))}\right)$$

Here, D is the total number of sentences. Intuitively, the inverse document frequency of a word is low if it occurs in many documents and is highest if the word occurs in only one. The so-called weight d(i) of word w_i in sentence d^i is then given as

$$d^{(i)} = TF(w_i; d)IDF(w_i)$$

This word weighting heuristic says that a word w_i is an important indexing term for document d if it occurs frequently in it (the term frequency is high). On the other hand, words which occur in many documents are rated less important indexing terms due to their low inverse document frequency.

4.2 Glove Feature

GloVe (Pennington et al., 2014) is an unsupervised learning algorithm for obtaining vector representations of words. It creates word vectors based on the distributional statistics of words, in particular how frequently words co-occur within a certain window in a large text corpus such as the Gigaword corpus (Parker et al., 2011). The resulting word vectors can be used to measure similarity between word sentences directly or through a classifier, following the hypothesis that similar words tend to have similar distributions. The Euclidean/Cosine or a softmax classifier between two sentence vectors can thus be used as a measure of their semantic similarity.

4.3 Word2Vec Feature

The work in (Mikolov et al., 2013) is a popular choice for pre-training the projection matrix $R^{d\times V}$ where d is the embedding dimension with the vocabulary V . As an unsupervised task that is trained on raw text, it builds word embeddings by maximizing the likelihood that words are predicted from their context or vice versa. The skip-gram models objective function is to maximize the likelihood of the prediction of contextual words given the center word. More formally, given a document of T words, we wish to maximize

$$L = \frac{1}{T}\sum_{T}^{t=1}\sum logp(w_{t+j}/w_t)$$

Where c is a hyperparameter defining the window of context words. To obtain the output probability $p(w_o|w_i)$, the model estimates a matrix $O\epsilon R^{d\times V}$, which maps the embeddings w_i into V dimensional vector o_{w_i}

Table 2: My caption

Feature Identifier	Feature Name
x1	Unigram
x2	Bigram
x3	Char Ngram
x4	Preselected features
x5	Word Embeddings
x6	x1 to x4
x7	x1 and x2
x8	x1,x3
x9	x1,x2,x3

4.4 N-Grams

An N-gram is an N-character slice of a longer string. Although in the literature the term can include the notion of any co-occurring set of characters in a string (e.g., an N-gram made up of the first and third character of a word), in this paper we use the term for contiguous slices only. Typically, one slices the string into a set of overlapping N-grams. In our system, we use N-grams of several different lengths simultaneously. We also append blanks to the beginning and ending of the string in order to help with matching beginning-of-word and ending-of-word situations. (We will use the underscore character (_) to represent blanks.) Thus, the word TEXT would be composed of the following N-grams: bi-grams: _T, TE, EX, XT, T_ tri-grams: _TE, TEX, EXT, XT_, T_ _ quad-grams: _TEX, TEXT, EXT_, XT_ _, T___ In general, a string of length k, padded with blanks, will have k+1 bi-grams, k+1tri-grams, k+1 quad-grams, and so on.

With previously defined features we use following 8 feature combinations shown in table 2.

5 Experiments and Results

In this section, we present results for each of the developed systems. The original dataset was split into **train17, test-17**[1] released at the start of competition and **dev-18, test-18** released during the competition pre-evaluation period for tuning of parameters and final evaluation respectively . We submitted CNN system for subtask 1 and CRF system for subtask 2. Tables 3-5 show the results of subtasks 1 and 2 respectively across the datasets. Our systems achieve F-score of 0.5 for subtask 1

[1](train/dev/test)-(17/18) is not an official naming convention , instead used here for ease of understanding

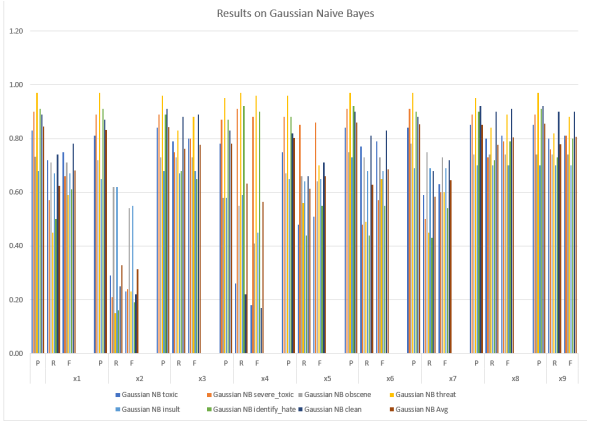

Figure 2: Gaussian Naive Bayes Results

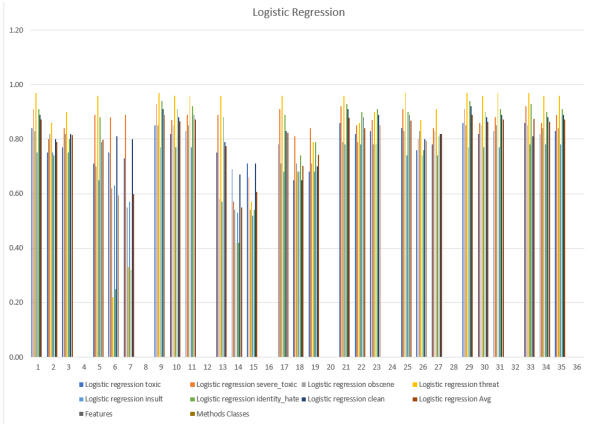

Figure 3: Logistic Regression Results

and 0.25, 0.36 for subtask 2 over strict, relaxed runs of subtask 2.

5.1 Discussion

6 Conclusion

In this work

Acknowledgments

We thank the task organizers

References

François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Edward Loper and Steven B Bird. 2002. Nltk: The natural language toolkit. *CoRR*, cs.CL/0205028.

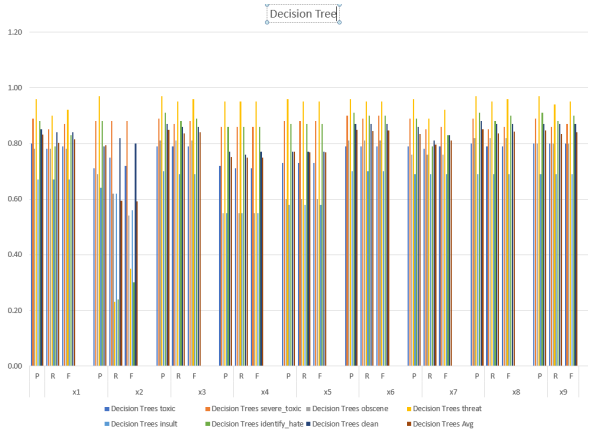

Figure 4: Decision Tree Results

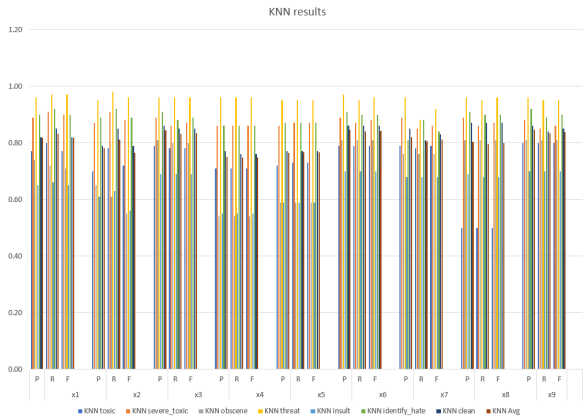

Figure 5: KNN results

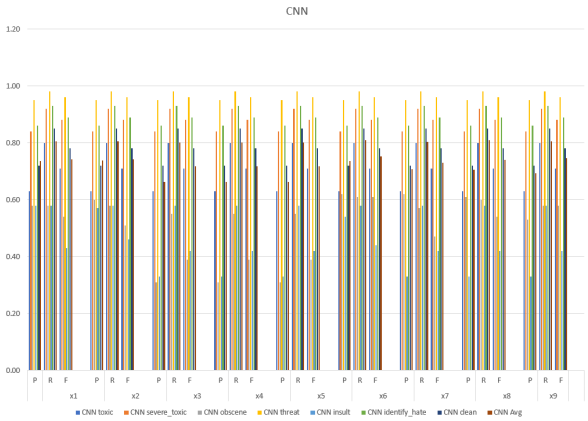

Figure 6: CNN Results

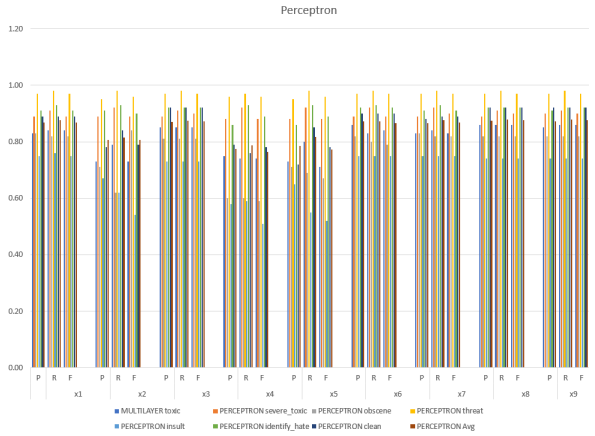

Figure 7: Perceptron Results

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *ACL*.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jacob VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

Frank Seide and Amit Agarwal. 2016. Cntk: Microsoft's open-source deep-learning toolkit. In *KDD*.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative study of cnn and rnn for natural language processing. *CoRR*, abs/1702.01923.

Ye Zhang and Byron C. Wallace. 2017. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *IJCNLP*.

Zhi-Hua Zhou and Xu-Ying Liu. 2006. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on Knowledge and Data Engineering*, 18:63–77.