CS 224N: Final Paper

Aaakarshan Dhakal
Stanford University
adhakal@stanford.edu

Dhruv Kedia
Stanford University
dkedia@stanford.edu

March 18, 2019

Abstract

With the rise of social media, the task of toxicity classification is increasing in importance day by day. It is important that toxicity in online platforms is reduced so that the internet is a safe place for people no matter their gender, age, race, nationality, likes, dislikes, religious beliefs, or political beliefs. Our goal for this project, is to use natural language and deep learning models to predict the toxicity class of a particular comment. We are using a dataset containing around 160,000 social media comments across 6 different toxicity classes. To address this problem we built a new deep convolutional neural network architectures that uses character to sentence information to perform toxicity classification on the short texts. The binary classifier model we built was more successfully able to distinguish between toxic and non-toxic comments, however, the multilabel classifier model that does fine grained classification performed averagely.

1 Introduction and Motivation

Social media platforms like Facebook, Twitter, Reddit, and several others have empowered people to share their views and opinions easily. As these online communities continue to grow, user generated content has started to increase. Some of this online user generated content is also "toxic" and hence the toxic content online has started to increase. Often online you can see bullying and harassment, hate speech, inappropriate comments, etc.

According to Center for Innovative Public Health Research, 47% of Americans have experienced online harassment of bullying [11]. To provide a scale of cyber-bullying and toxic comments, a report stated that there were 10 million player reports on 1.46 million toxic players in the famous game the League of Legends[10]. It is extremely important for social media and other online platforms to segregate different kinds of toxicity and thereby moderate the user generated content. Social media companies like Facebook, Twitter, and YouTube have greatly accelerated their removal of online hate speech, reviewing over two thirds of complaints within 24 hours [6]. Therefore, detecting toxic comments and speech online and acting to remove it is of paramount importance. Hence, for our project we plan to build a deep learning model that can classify different kinds of toxicity like insult, identity hate, obscene, threat, toxic, and severely toxic.

2 Related Works

2.1 Architecture

Our model architecture is based on [2] This goal of this paper is to perform sentiment analysis on short texts like single sentences, movie reviews, or Twitter messages. The authors were motivated in doing this project because of the challenge that short texts contain limited contextual information. Hence, the authors were interested in exploring a way to combine this short text content with prior knowledge, rather than performing sentiment analysis using just a bag-of-words. The authors built a new deep convolutional

neural network architectures that uses character to sentence information to perform sentiment analysis on the short texts. This was a particularly noteworthy model because it managed to achieve state of the art results in both binary classification and fine-grained multi-class classification on the Stanford Sentiment Treebank dataset that contains sentences from movie reviews. It happened to outperform the RNTN model in the paper Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank by Richard Socher and colleagues. Our task on toxicity classification is similar to this reference paper in the fact that it is doing fine grained multi-class text classification and this classification is on short texts.

2.2 Multi-Label Loss Function

Multi-label classification problems are common, they have not been widely addressed. [3], [1] focus on training a deep network that uses raw pixels and employs a multi-label ranking loss. This loss function is the biggest difference between the binary and multi-label classifier. Softmax loss proves to work extremely well for binary classification problems. However, softmax loss does not work well for multi-label classification problems, since the scores of different classes compete with each other. Using [3], we customized the loss function to be suit- able for a multi-label classification problem.

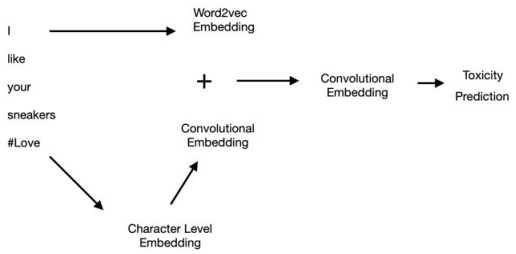
2.3 Evaluation Metrics

There is some class imbalance (across toxicity levels) present in the data, hence it was important that we look into other methods besides accuracy to evaluate our models. We used [4], to explore different evaluation metrics such as accuracy, precision, recall, and f1 scores. We also used [12] to examine how one can calculate precision and recall for multi-label classifiers differently, as mentioned in .

3 Approach

The model we implemented is based on the model described in the paper Deep Convolutional Neural

Figure 1: Model Overview



Networks for Sentiment Analysis of Short Texts by Cicero Nogueira dos Santos and Maira Gatti [4]. The task at hand is a multi-label problem, where a particular comment (observation) could belong to multiple toxicity classes.

The model takes a sentence as an input and computes a score for each of the sentiment labels. The model can take in sentences and words of any arbitrary size. In order to compute the score, the model passes the sentence through a sequence of layers that extract features with increasing complexity levels. The model extracts features from the character level up to the sentence level. The model architecture uses two convolutional layers. A visual representation of the model can be seen in figure 1.

A detailed description of the model in figure 1 is as follows:

3.1 Word-Level Features

The first step is to create word-level embeddings for the different words observed in the data. These word-level embeddings are created using the word2vec model explored in class. These embeddings are obtained using a neural network with a single layer. We pass a one-hot vector representing a particular input word and as the output we get the probability of that word being near the centre word. Out here the goal is not to learn the probability outputs, but to learn the weights from the hidden layer. [2]

3.2 Character-Level Features

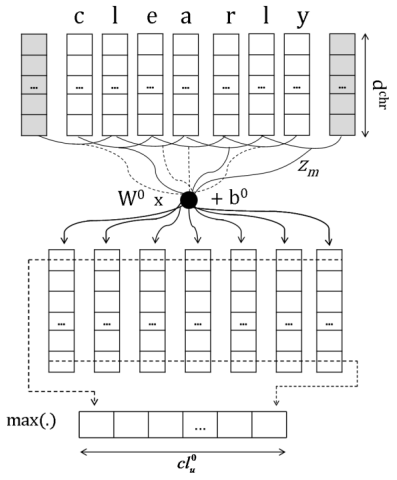
The model also makes use of character-level embeddings. These character level-embeddings are necessary because important information can appear in different parts of a word, for example in "#insta", the "#" could be crucial. The character-level embeddings are created using a convolutional approach that produces local features around each character of the word and then combines them using a max operation to create a fixed-sized character-level embedding of the word. A detailed explanation of the convolution approach is as follows-

Similar to the process of producing word-level embeddings, each character is transformed into a character-level embedding. The convolutional layer applies a matrix-vector operation to each window size k^{chr} of successive character embeddings in the sequence. Therefore, a particular window consists of the concatenation of the character embedding m, its $\frac{k^{chr}-1}{2}$ left neighbors, and its $\frac{k^{chr}-1}{2}$ right neighbors. To account for corner cases, on both the right and left side length $\frac{k^{chr}-1}{2}$ paddings are added. The output of the convolution is then reduced to a single vector of size cl_u^0 by using cl_u^0 filters and maxpooling over all character windows in each filter. This is the resulting character-level feature vector. A visual representation of the convolution approach can be seen in figure 2. This image has been taken from the paper Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts by Cicero Nogueira dos Santos and Maira Gatti. [2]

3.3 Sentence-Level Representation

The model's next step is to extract a sentence-level representation from the joint word-level and character-level embeddings. This is done so because sentences have different sizes, and critical information can appear at any position in the sentence. The model computes a sentence-wide feature vector using a convolutional layer and in a manner similar to how the character-level embeddings were created. The convolution layer produces local features around each word of the sentence and then combines them using a max operation to create a fixed-size feature vector for the sentence. Finally, this "global' feature vector is processed by two usual neural network layers that extract one more level of representation and compute a score for each label: toxic and not toxic.

Figure 2: Convolutional Approach



3.4 Baseline Binary Classifier

As an initial step we a built a simple binary classifier. A binary classifier classifies the data into 2 classes: if it is toxic or not toxic. The reason we decided to use binary classifiers is because it is a relatively simple way to get started and obtain some initial results.

In the binary classifier, the scores are transformed to a conditional probability distribution of labels by applying the softmax operation over the scores for each label: toxic and not toxic. The networks is trained to minimize the negative log likelihood. The loss function is as follows:

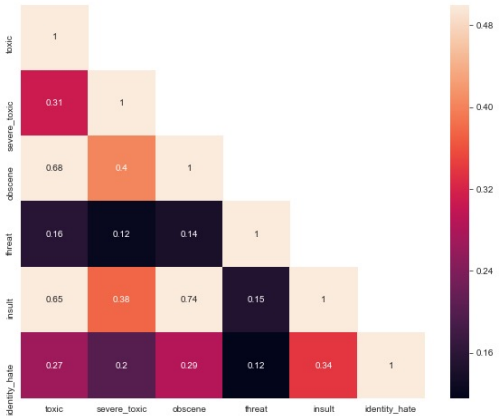
$$loss(x, class) = -x[class] + \log(\sum_{jclasses} \exp(x[j])$$

3.5 Multilabel Classifier

Since each comment can belong to multiple toxicity, it is important to also look at the correlation (co-occurrence between different toxicity class). In figure

3, we see a correlation heatmap of the toxicity classes in the dataset. It is important to transition to using a multi-label classifier in order to incorporate these correlations.

Figure 3: Correlation Heatmap of Genres



Hence, order to account for these correlations, we customized the loss function to incorporate linear logistical loss functions for multiple classes (toxicity levels), as opposed to softmax loss that improve the score of a certain class at the expense of other classes. In order to to ensure we accounted for multiple labels, we applied equation 1 from [3] where $\overline{p}_{ij}$ is the ground truth probability and p_{ij} is the posterior probability of a comment, n is the number of comments, and c is the number of toxicity classes:

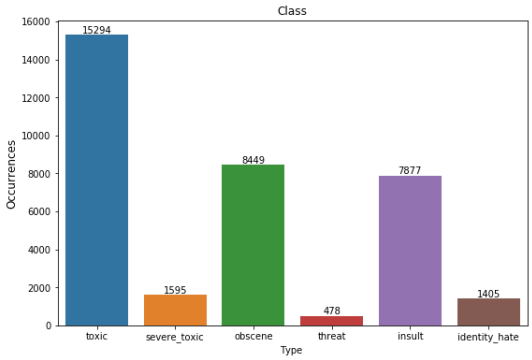
$$Loss = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c}\overline{p}_{ij}\log(p_{ij}) \tag{1}$$

4 Experiments

4.1 Data

In order to build this model, we will be using the Toxic Comment Classification Challenge dataset from a Kaggle competition [15]. The dataset contains about 159,571 Wikipedia comments that have been hand labelled by human raters for toxicity levels. The different classes of toxicity are toxic, severe toxic, obscene, threat, insult, and identity hate. The

Figure 4: Data Distribution



dataset contains the comments and binary labels for indication if the comment belongs to the class or not. The distribution of the the data can be seen in figure 4. We can see that in the data slightly over 140,000 of the comments are not toxic. Although this is a well created dataset, some preprocessing is required. Some comments are repeating in the dataset and it is important we eliminate duplicates.

In addition, we ensured that a random 80% of the data set was set aside for training, 10% for validation and 10% for test. The reason we chose to use an 8:1:1 split, rather than the conventional 6:2:2 split is because our dataset is relatively large (close to 160,000 observations) and hence 10% for validation and test is sufficient. We also ensured that the representation of each toxicity class is consistent across training, validation, and test set.

4.2 Evaluation Metrics

In order to evaluate the binary classifier, we used accuracy, precision, and recall. The reason we use precision and recall is because there is class imbalance (significantly more non toxic comments).

$$Accuracy = \frac{true\ positives + true\ negatives}{All\ Examples} \tag{2}$$

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{3}$$

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \quad (4)$$

For the multi-label classifier, the issue was how to calculate precision and recall with multiple classes. To keep track of the progression of the model during training, we calculated the average precision score with scikit-learn average precision function that uses equation 5:

$$AP = \sum_n (R_n - R_{n-1}) P_n \quad (5)$$

where P_n and R_n are the precision and recall at the nth threshold [14]. To monitor training progress, we plotted the average precision over 20 epochs.

Additionally we also calculated AUC score. AUC is the area under the ROC (receiver operating characteristic curve). It is the area between the ROC curve and the x-axis. The ROC curve plots the true positive rate (recall - the probability of predicting a real positive will be a positive) against the false positive rate (the probability of predicting a real negative will be a positive) at different threshold settings. We measure AUC score because it allows us to compare results with the Kaggle competition leaderboard that uses AUC score to rank.

4.3 Experiment Details

4.3.1 Hyperparameter Tuning

Initially to begin with we choose the same hyper-parameters as the paper used for the sentiment classification task on the Stanford Twitter Sentiment (STS) corpus. The initial hyper-parameters were as follows:

Param	Param Names	Vals
d^{wrd}	Word Level Embeddings dims	30
k^{wrd}	Word Context window	5
d^{chr}	Character Embeddings dims	5
k^{chr}	Character Context window	3
cl_u^0	Character Convolution Units	50
cl_u^1	Word Convolution Units	300
h_{lu}	Hidden Units	300
λ	Learning Rate	0.01
p	Dropout Probability	0.5

We also used an Adam optimizer. Adam doesn't stuck in flat regions as it uses a decaying running average of past gradient squares. It is the recommended default algorithm to use, as it performs well on various optimization tasks and is easy to train [9].

Last, we also hyper tuned the learning rate and the dropout probability.

4.3.2 Use of Pre-trained Word Embeddings

Initially for the milestone, we self-trained the word vectors in order to obtain results. However, upon closer analysis of our training data, we realized that a lot of the words in comments were actually everyday English words. Hence, we decided to test our multi-label model using pre-trained word vectors. We experimented with GloVe vectors that had been trained on 2 billion tweets.[5] The reason we chose GloVe vectors trained on twitter data was because our dataset is obtained from user's Wikipedia comments. We expect these comments to resemble tweets similarly and many of the words found to be part of the same vocabulary.

4.4 Results

4.4.1 Binary Model

Our model managed to achieve close to 88% accuracy on the training set. Further on on the validation set, we managed to achieve the following results: Accuracy = 0.725, Precision = 0.73, Recall = 0.72. These results are better than we expected given that we had performed no hyper parameter tuning. Given, that we wanted to build a multi-label model, we did not

spend additional time hyper-tuning and improving the binary classification model.

4.4.2 Dropout

From figure 5 and 6 we can see that when there is no dropout, the model has a lower training loss and higher validation loss when compared to a model that has dropout of 0.4. Further on, we also notice that the model that does not use dropout suffers from over fitting. This can be seen because when there is no dropout, the training and validation loss diverge at epoch 3. We found that a dropout of 0.4 leads to the highest average precision.

Figure 5: Loss with No Dropout

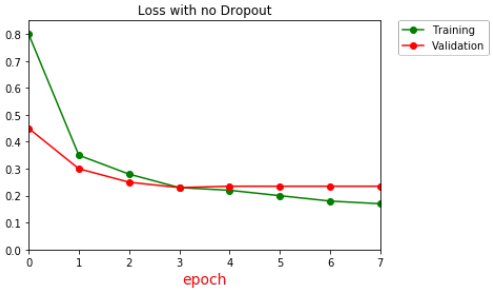
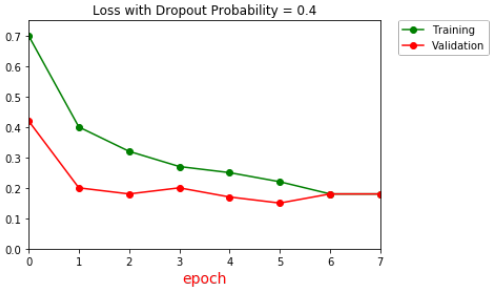


Figure 6: Average Precision (With GloVe Embeddings)



4.4.3 Word Embedding Results

In figure 7 and 8 we can see how average precision varies over epochs with and without GloVe Embeddings. We can see that with GloVe Embeddings, the model was able to achieve both a higher training average precision and a higher validation average precision. Also, the overall model with GloVe Embeddings took about half the time to train as compared with the model where the word embeddings were learned from scratch. We believe this is the case because the GloVe Words Embeddings were trained on a large dataset (2 billions tweets) and hence we are able to capture more information than training from scratch.

Figure 7: Average Precision (Without GloVe Embeddings)

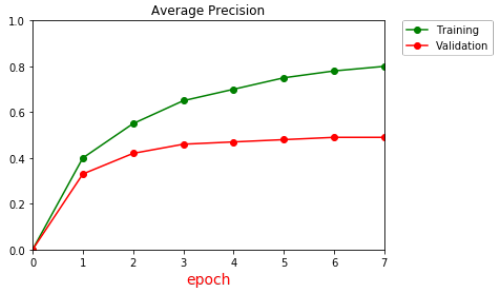
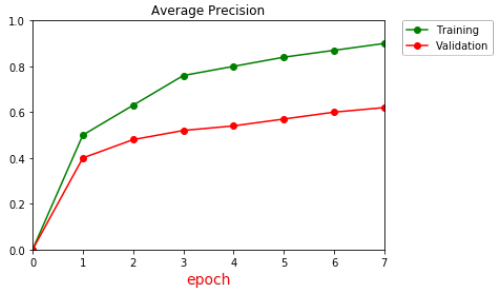


Figure 8: Average Precision (With GloVe Embeddings)



4.4.4 Test Set Results

After performing hyperparameter tuning and insights from section 4.4.3, we found that our best model would be one that uses GloVe word embeddings trained on twitter data, has a learning rate of 0.1, has a dropout probability of 0.4, and uses an Adam optimizer. The results we obtained on the validation

and test set with this best model are as follows:

Dataset	Average Precision	AUC Score
Validation	0.62	0.9782
Test	0.65	0.9740

5 Analysis

In our qualitative analysis, we found that our best model can detect toxicity in the following forms:

1) The model fails to identify a statement as non toxic when the language used is strong. For example the following sentences was tagged as toxic, even though they are not toxic:

- "Locking this page would also violate WP:NEWBIES. Whether you like it or not, conservatives are Wikipedians too." The presence of strong words such as violate, and NEWBIES confused the model.
- "Have you tried to fathom why reactions to you are harsh?" The presence of strong words such as fathom, and harsh confused the model.

2) Direct insults and comments are tagged toxic with respective labels with a very high confidence. Whenever, a sentence contains words such as "f*ck, shit, shut up, idiot".

3) Toxic comments that had spelling errors were also labelled as toxic with a fairly high confidence. Examples:

- ". Fu ck ing trollreasons"
- "if ytou think shes greek your a morooon."

Both the above sentences were correctly classified as being toxic even though they contained spelling errors such as "Fu ck ing". and "moroon".

4) Certain words were strong indicators of severe toxicity. For example, when words such as "d*ck", and "assh*le", the model always classified. This was problematic because a sentence about "Moby Dick" was classified as severely toxic when in reality it had no element of toxicity.

5) Statements that contain insults in foreign languages or unfamiliar slangs are not labelled accurately. Example:

- "Now, Tony, why do you want to be known as the typical right-wing, born-again Christian fist-fuckee of 2006?". The model was not able to identify "fuckee" as an insult, and classified this sentence as non-toxic.
- "Mr Rajchut donot bring chamars in Saini discussion. Donot degrade any caste." In this sentence, "chamars" refers to "untouchables" caste in Northern India and "Mr. Rajchut" refers to "Mr. Raj" being a "chut". Both "chamars" and "chut" are derogatory terms in the Hindi language but the model failed to classify this sentence as an insult or identity hate.

6) The model does a good job in identifying comments that belong to the toxicity class "threats". Whenever, the sentence contains action words such as "kill", "stop", "destroy", the model almost never misses classifying the comment as a threat.

6 Conclusion

Our binary model is able to perform decently well, however our multilabel model only performs averagely. Our best multilabel model used a customized loss function, pre-trained GloVe word embeddings, and a deep convolutional neural network architectures that uses character to sentence information. One of our key limitations working on this project was training time. Our model took significant amount of time to train. This did not allow us to further hyper-tune different parameters in the multilabel model such as hidden units, character context window, character embedding dimensions, etc. When using average precision as a measure to analyze performance, this model did not fair extremely well.

Nonetheless, our AUC of 0.9740 is not much lower compared to the best model in Kaggle Leaderboard that obtains an AUC score of 0.98856.

Given more time and resources, we would better hypertune our model. Additionally, we did not manage to successfully integrate ELMO or BERT contextual word embeddings. Given more time, we would like to spend more time on getting these to work. It can also be valuable to see how a completely different architecture that employs LSTMs and GRUs fairs in toxicity classification. Lastly, we would have liked to do more in depth error analysis similar, find more commonly occurring patterns, and try to identify solutions that will help make the model more robust.

7 Acknowledgements

We would like to express our gratitude to the entire CS224n teaching team, especially Sahil Chopra for mentoring us in this project.

References

[1] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T. DeCAF: a deep convolutional activation feature for generic visual recognition. *arxiv:1310.1531.*

[2] Dos Santos, Cicero & Gatti de Bayser, Maira. (2014). Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts.

[3] Gong, Yunchao, Jia, Y, Leung, T., Toshex, A., Ioffe, S. Deep Convolutional Ranking for Multilabel Image Annotation. *arXiv:1312.4894.*

[4] Hossin, M., Sulaiman, M.N. A review on evaluation metrics for data classification evaluations., International Journal of Data Mining and Knowledge Management Process. vol.5, No.2. 2015.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[6] Jubany, Olga. "Backgrounds, Experiences and Responses to Online Hate Speech: An Ethnographic Multi-sited Analysis." 2nd Annual International Conference on Social Science and Contemporary Humanity Development. Atlantis Press, 2015.

[7] Karani, Dhruvil. "Introduction to Word Embedding and Word2Vec – Towards Data Science." Towards Data Science, Towards Data Science, 1 Sept. 2018

[8] Kedia Dhruv, Schechter Amit, Pattaki Camile. Don't Judge a Movie by its Poster (2018).

[9] Kingma, Diederik, Ba, J. Adam: a method for stochastic optimization. *arxiv:1412.6980.*

[10] Kwak, Haewoon, Jeremy Blackburn, and Seungyeop Han. "Exploring cyberbullying and other toxic behavior in team competition online games." Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. ACM, 2015.

[11] Lenhart, Amanda, et al. Online harassment, digital abuse, and cyberstalking in America. Data and Society Research Institute, 2016.

[12] Marina Sokolova, Guy Lapalme, A systematic analysis of performance measures for classification tasks, Information Processing and Management. Vol 45. Issue 4. 2009.

[13] Paszke Adam, S Gross, S Chintala, G Chanan. Automatic differentiation in PyTorch. 2017.

[14] Pedregosa, Fabian, Varoquaux, G, Gramfort, A, Michel, V, Thirion, B, Grisel, O, Blondel, M, Prettenhofer, P, Weiss, R, Dubourg, V, Vanderplas, J, Passos, A, Cournapeau, D, Brucher, M, Perrot, M, Duchesnay, É. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011)

[15] Toxic Comment Classification Challenge, Kaggle, www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge.

[16] Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).

[17] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)