CS224N: Detecting and Classifying Toxic Comments

Kevin Khieu
kkhieu@stanford.edu

Neha Narwal
nnarwal@stanford.edu

Abstract

This paper introduces various deep learning approaches applied to the task of classifying toxicity in online comments. We study the impact of Support Vector Machines (SVM), Long Short-Term Memory Networks (LSTM), Convolutional Neural Networks (CNN), and Multilayer Perceptron (MLP) methods, in combination with word and character-level embeddings, on identifying toxicity in text. We evaluated our approaches on Wikipedia comments from the Kaggle Toxic Comments Classification Challenge dataset. Our word-level assessment found that our forward LSTM model achieved the highest performance on both binary classification (toxic vs non-toxic) and multi-label classification (classifying specific kinds of toxicity) tasks. Regarding character-level classification, our best results occurred when using a CNN model. Overall, however, our word-level models significantly outperformed our character-level models. Moving forward, we seek to attain higher performance through applying richer word/character representations and using more complex deep learning models.

1 Introduction

Conversational toxicity is an issue that can lead people both to stop genuinely expressing themselves and to stop seeking others' opinions out of fear of abuse/harassment. The goal of this project will be to use deep learning to identify toxicity in text, which could be used to help deter users from posting potentially hurtful messages, craft more civil arguments when engaging in discourse with others, and to gauge the toxicity of other users' comments.

This project aims to implement various deep learning models - specifically multilayer perceptron (MLP), Long Short Term Memory Networks (LSTM), and Convolutional Neural Networks (CNN) - to tackle the above task, assessing these models' performances on both binary and multi-label classification tasks. Our project also studies the applications of these models at both word-level and character-level granularities.

2 Background/Related Work

Sentiment classification regarding toxicity has been intensively researched in the past few years, largely in the context of social media data where researchers have applied various machine learning systems to try and tackle the problem of toxicity as well as the related, more well-known, task of sentiment analysis. Comment abuse classification research initially began with Yin et al's application of combining TF-IDF with sentiment/contextual features. They compared the performance of this model with a simple TF-IDF model and reported a 6% increase in F1 score of the classifier on chat style datasets (Kongregate, MySpace)[1]. We discuss further related works specific to our approaches below.

2.1 Sentiment Label Distribution on Twitter Data

Nguyen and Nguyen[2] proposed a model for sentiment label distribution that involved a combination of using a DeepCNN for character-level embeddings (in order to increase information for word-level embeddings) with a Bidirectional LSTM which produced sentence-wide feature representations from word-level embeddings. This model attained a best prediction accuracy of 86.63% on the Stanford Twitter Sentiment Corpus. Their findings indicate the prospective advantages of utilizing LSTM and DeepCNN models on the task of toxicity classification.

2.2 Refinement of Word Embeddings for Sentiment Analysis

Yu and Wang[3] proposed a word vector refinement model that could be applied to pretrained word vectors (e.g. Word2vec and Glove) to enhance sentiment information capture. The refinement model was based on adjusting vector representations of words so that they could be closer to both semantically and sentimentally similar words and farther away from sentimentally dissimilar words. The word embeddings from the refined model (Re), Re(Word2vec) and Re(GloVe) respectively, improved Word2Vec and GloVe by 1.7% and 1.5% averaged over all classifiers for binary classification, and both improved by 1.6% for fine-grain classification. These results suggest potential further improvement of our model may result from additional enhancement of word vectors (from using this refinement model) that capture more semantic information.

2.3 Comment Abuse Classification with Deep Learning

Chu and Jue[4] compared the performance of various deep learning approaches to this problem, specifically using both word and character embeddings. They assessed the performance of recurrent neural networks with LSTM and word embeddings, a CNN with word embeddings, and a CNN with character embeddings. The best performance they achieved was a 93% accuracy using the character level CNN model.

In this paper we further extend the previous analysis to assess performance of deep learning approaches: Forward LSTM, Bi-directional LSTM, CNN, Multilayer Perceptron Networks, and Support Vector Machines, both at the word and character level and for both binary and multi-label classification.

3 Approach

This project focuses on studying the effects of three different kinds of neural network models (MLP, LSTM, and CNN) at two levels of granularity - word-level and character-level - on both binary and multi-label classification tasks. We detail these approaches, along with our baseline, below.

3.1 Tasks

3.1.1 Binary Classification

Our binary classification task was our simple task - our aim was, given an input comment, to return whether or not this comment is *toxic* or *non-toxic*. For this, we standardized across all models to use sparse categorical cross-entropy loss.

3.1.2 Multi-Label Classification

The goal of the multi-label classification task was to determine whether or not a comment is *toxic* or *non-toxic* and, if *toxic*, to determine what kind of toxicity this comment is (*severeToxic, obscene, threat, insult,* and/or *identityHate*). For models tackling this task, we standardized across all models to use binary cross-entropy loss.

3.2 Baselines: Support Vector Machine

To start our work, we began by creating a baseline implementation for our binary classification task. The goal of our baseline was to assess the difficulty of the simpler of our two tasks and to provide

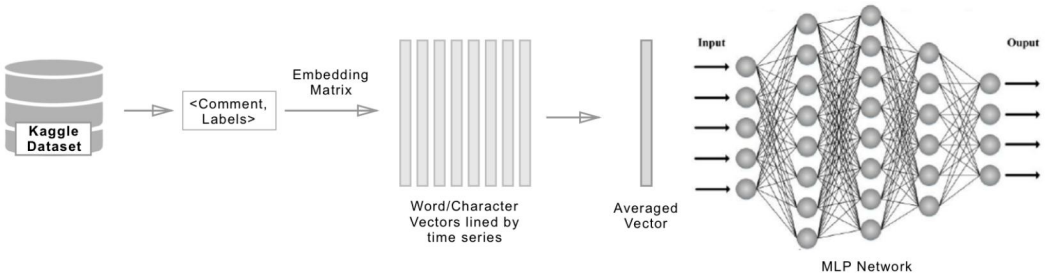
a standard from which to compare our deep learning models to. Our baseline consisted of a simple implementation of sklearn's built-in SVM package. Each training input was passed into the SVM by averaging the GloVe word vectors of all words in that input sentence, with the model returning a binary classification for each input, optimizing on hinge loss.

3.3 Neural Network Models

As stated before, our project focused on studying the effects of three different kinds of neural network models: Multilayer Perceptron networks, Long Short Term Memory Networks, and Convolutional Neural Networks. To implement these models, we used Keras, a lightweight Python wrapper written on top of TensorFlow, CNTK, and Theano that allowed us to more easily sequentially construct and tune our neural network models.

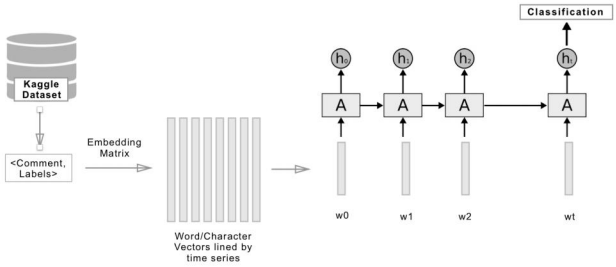
3.3.1 Multilayer Perceptron

Our first neural network was the standard multilayer perceptron network, which used stochastic gradient descent to tune the weights of all nodes in each hidden layer. The input to this network was the average vector of all GloVe representations of each word for a given comment. The network used ReLU activations between each hidden layer, and the final output layer used a softmax activation function over the number of classes (2 for binary classification, 5 for multi-label classification). Our model for binary classification used sparse categorical cross entropy loss, and for multi-label classification we used binary cross entropy loss over each label.



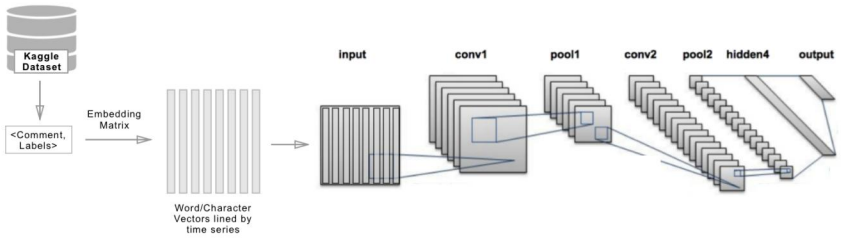
3.3.2 Long Short Term Memory

One issue with relying on the multilayer perceptron model we created was that it averaged all word vectors in a given comment, thus losing valuable word order-related information. Our usage of a Long Short Term Memory network was meant to combat that weakness by providing a model that could be unrolled across non-arbitrary comment lengths, analyzing comments word by word. For our LSTM model, we padded/clipped our input to standardize each input length. This clipping occurred at 100 words for word-level granularity and 100 characters for character-level. We experimented with various LSTM layerings and output dimensions. For our output layer, we used a softmax activation function for binary classification with sparse categorical cross entropy loss, and for multi-label classification we used a sigmoid activation with binary cross entropy loss over each label. We use the Adam optimizer for optimizing our cross entropy loss.



3.3.3 Convolutional Neural Network

Our last neural network model was an experiment on the effectiveness of applying convolutional neural networks to this task. The idea behind this approach was to treat each comment as an "image", lining each comments' words in vector form side by side and running a CNN over it. We use ReLU activations in between each convolution and pooling layer, and our final output layers use the same activation and loss functions as our previous two models. Like the LSTM, we use an Adam optimizer for optimizing cross entropy loss.



4 Experiments

4.1 Dataset

The dataset we used came from Jigsaw/Google's Toxic Comment Classification Challenge on Kaggle. The dataset contains 159,571 labeled examples of Wikipedia comments that have been labeled by human raters for toxic behavior. The data comes in the schema of <id, commentText, toxic, severeToxic, obscene, threat, insult, identityHate>, where the labels for *toxic, severeToxic, obscene, threat, insult,* and *identityHate* are all boolean labels.

4.1.1 Data Preprocessing and Augmentation

Data processing was done using R and Python.

For our binary classification task, we used the given *toxic* label as our binary indicator for whether or not a given comment was or was not toxic. Balancing the dataset using random sub-sampling led us to having a total of 30,590 examples for training/validation/testing. To separate our data, we used a 60-20-20 split.

For our multi-label classification task, we used the same balanced dataset from the binary classification task but also included the labels for each specific type of toxicity (*severeToxic, obscene, threat, insult,* and *identityHate*). For this task, to reduce the number of 0-labels, we also experimented with a version of our dataset that had 0 non-toxic sentences (15,295 examples) in an effort to further balance our labels. We talk more on this later.

4.1.2 Model-Specific Data Preprocessing

Our word-level analysis utilized GloVe word vectors pretrained on Wikipedia data as vector representations of each word in our dataset. For our character-level analysis, we trained our own character-level embeddings using Keras' Embedding Layer package.

Additionally, for our LSTM and convolutional neural network models, we implemented comment padding/clipping that would ensure that each example in our dataset was of equal word/character length. This allowed us to efficiently batch process the data by treating every input comment as being of equal lengths.

4.2 Evaluation Metrics

For the binary classification task with word level and character level granularity, we compared performance across the models through accuracy, precision, recall, specificity and F1- score. We obtained these metrics from the confusion matrix obtained for each iteration of hyper parameter tuning on the validation set.

The confusion matrix summarizes the correct and incorrect predictions through count values for each class that helps gain insight into the types of errors made by the model. Accuracy of the model is the ratio of correctly predicted observations to the total observations. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall is the ratio of correctly predicted positive observations to the all observations in actual positive class. F1 score is the harmonic mean of precision and recall. It becomes an important lever to gauge the performance of the classifier in case of imbalanced classes.

For the multi-label classification task, additionally, we considered two versions of accuracy measurement, one which reported accuracy based on correct classification across all 5 classes on a single comment (sentence accuracy) and the other which reported label correctness across the entire dataset (label accuracy).

4.3 Models

4.3.1 SVM

Our first task was to apply our baseline approach to the binary classification task at word level granularity. To do this, we fed as input the average of the GloVe word embeddings for all words in a comment into sklearn's default SVM package. Doing this obtained a test accuracy of 0.833. This approach was naive in the sense that it uses averaged word representations and is prone to losing encoded semantic information in the sentence. However as a baseline, the approach was successful in helping us conclude that (1) this was a doable task and (2) that there is possibility for attaining high performance using advanced deep learning methods.

From here, we moved forward with assessing three different models for our task of toxic comment classification: the multilayer perceptron model, LSTMs, and CNNs, at both the word-level and character-level. As extensions, we also attempted to replicate the DeepCNN-Bidirectional LSTM model described in Nguyen and Nguyen.

4.3.2 Multilayer Perceptron

For the multilayer perceptron models, we experimented with varying the number of hidden layers, number of hidden layer nodes, and the number of epochs. All of our MLP models used a stochastic gradient descent optimizer on loss, with a learning rate of 0.01 combined with decay of 1e-6 and momentum of 0.9. Reasonable tuning of the learning rate did not have noticeable effects on our results likely due to the use of decay and momentum. The results of this tuning can be found in our supplementary materials.

Binary Classification: Our optimal model resulted from using 3 hidden layers at 64 nodes per hidden layers, converging at around 300 epochs. This model attained 0.853 test accuracy, with fairly high precision (0.862) and recall (0.842). This beat our baseline by roughly 0.02, which is a slight improvement.

Multi-Label Classification: Our optimal multi-label MLP model resulted from using 3 hidden layers at 128 nodes per hidden layers, converging at around 350 epochs. On the test set, this model attained 0.918 label accuracy and 0.729 sentence accuracy.

4.3.3 LSTM

Our LSTM models were tuned by varying the number of LSTM layers applied to our data, the output dimensions of each layer, and padding length. For the character-level analysis, we also experimented with tuning character embedding sizes. All models were tuned by validating at 5 epochs, and our final test runs for both binary and multi-label LSTM were done at 50 epochs.

Binary Classification: Our optimal binary classification LSTM used 3 LSTM layerings with 32 output units at each layer. When run on the test set, this model attained 0.889 test accuracy, 0.925 precision, and 0.850 recall. We consider this model to be our best overall approach at the binary classification task, consistently achieving high precision and recall on our validation test sets without over-fitting to training data, as shown by it also attaining our highest F1 score of 0.886.

Multi-Label Classification: Our optimal multi-label LSTM model also resulted from using 3 LSTM layers but at 64 output units per layer. On the test set, this model attained our best observed label accuracy of 0.927, and a sentence accuracy of 0.735. This model achieved precision of 0.733 and recall of 0.681. We considered this to be our best multi-label classification model as well due to it having the highest sentence test accuracy.

Character-Level Approach: Per the suggestion of our background research, we also tried a character-level forward LSTM to see how effective this model would be in tackling our binary classification task. For this, we found our model attained highest validation accuracy when using 100 character clipping and 3 LSTM layers, although our final accuracy was very low at 0.679 and F1 score of 0.669.

4.3.4 CNN

For the CNN model we varied kernel size, pool size, stride, and number of layers to observe their respective impacts on performance. Initially, we observed obvious over-fitting to the training data when our CNN was run over many iterations, at times achieving 99% training accuracy at the expense of validation accuracy. Introducing a dropout of 0.2 helped curb this over-fitting and increased our validation accuracies. For all of the above models, we looked at variation in the training and validation accuracies with regards to number of epochs to infer convergence.

Binary Classification: We found that a kernel size of 3, dropout of 0.2 at 30 epochs produced the highest validation accuracy without over-fitting to the training data. It is worth noting that our CNN models at times were able to achieve upwards of 0.92 validation accuracy, metrics that are rather unexpected given that we decided to use this pure CNN model on a whim. We suspect this is a consequence of how GloVe vectors are trained and structured. Ultimately, our binary classification CNN using dropout achieved 0.872 test accuracy, 0.858 precision, and 0.884 recall.

Multi-Label Classification: Our optimal multi-label CNN model resulted from using a large Kernel size of 8. On this task, our model achieved 0.889 accuracy, 0.666 sentence accuracy, 0.664 precision, and 0.176 recall. Low recall was a common trait in all of our multi-label CNN tests, likely due to the model being overwhelmed by the high number of 0's in our multi-label dataset. Low recall was also present in the MLP task, which also ended up falling back to predicting too many 0 labels due to the heavily imbalanced dataset we had for this task. Attempts to curb this imbalance via applying label weights and trying different partitions of our dataset still led to similar low-recall results.

Character-Level Approach: Our character-level CNN used 64 output units per CNN layer with Kernel size 3 and 100 character clipping. This achieved higher performance than LSTM on our binary classification task, attaining 0.803 label accuracy, 0.805 precision, and 0.796 recall. Overall however, the character level CNN and LSTM were both less effective than the word-level models we developed.

4.4 Results

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Word Level Analysis:				
CNN (Kernel Size: 3, Drop Out: 0.2)	0.872	0.858	0.884	0.871
CNN (Kernel Size : 3)	0.873	0.878	0.868	0.873
MLP (Layers: 3, Nodes: 64)	0.853	0.862	0.842	0.852
LSTM (Layers: 3, Output Units: 32)	0.889	0.925	0.850	0.886
Character Level Analysis:				
CNN (Output Units: 64, 100 char clipping)	0.803	0.805	0.796	0.800
LSTM (Layers: 3, 100 char clipping)	0.679	0.681	0.657	0.669
Character level CNN + LSTM	0.797	0.804	0.790	0.797

Figure 1: Deep Learning Model Performance (Binary Classification)

Model	Label / Sentence Accuracy	Precision	Recall	F1 Score
CNN (Kernel Size: 4)	0.886 / 0.657	0.659	0.197	0.303
CNN (Kernel Size: 2)	0.887 / 0.658	0.644	0.188	0.292
CNN (Kernel Size: 8)	0.889 / 0.666	0.664	0.176	0.278
LSTM (Layers: 3, Output Units: 64)	0.927 / 0.735	0.733	0.681	0.706
MLP (Layers: 3, Nodes: 128)	0.918 / 0.729	0.754	0.476	0.584

Figure 2: Deep Learning Model Performance (Multi-Label Classification)

4.4.1 Overall Quantitative Results

Multi-Label Classification
For the multi-label classification task, we achieved a best performance of 0.927 label accuracy from our LSTM model. In addition to recording the best F1-score, LSTM also performed best with regards to sentence accuracy (0.735). Our multi-label LSTM model also achieved the highest F1 score of 0.706 our of all models. The MLP model performed similarly well.

For all models in our multi-label task, label accuracy was an inconsistent indicator of performance due to the overly large proportion of 0 labels in our multi-label dataset (even though half of our sentences were considered "toxic", almost 87.5% of toxicity labels were 0's). We tried to curb this issue via adding class weights and by trying different partitions of our dataset, but these changes provided little relief. From the discussions online that we read on Kaggle, this seems to be a common issue. Hence, we introduced the sentence accuracy metric to better determine which models were best - this metric combined with F1 score led us to concluding that LSTM was our best performing model.

Binary Classification
LSTM also performed highest on our binary classification task, achieving the highest accuracy, precision, and F1 scores of all models tested. For this task, F1 scores did not exhibit considerable variation across all models, with almost all models (except for our character-level LSTM) achieving better performance than our simple SVM baseline.

Character-Level Binary Classification
With regards to the binary classification task for character level granularity, we were not able to obtain superior performance compared to word level analysis. CNN (0.803 accuracy) performed better compared to LSTM on both precision and recall. In an attempt to better this performance, we also tried to replicate the model created in Nguyen and Nguyen by layering a character-level DeepCNN on top of a word-level LSTM, but our performance still did not improve (only achieving 0.797 accuracy).

4.4.2 Overall Qualitative Evaluation

To conduct further evaluation, we took our best performing model - the word level LSTM - and created a bot that rated the toxicity of user inputs. Some examples of this evaluation can be seen below in Figure 3.

Input	Judgment	Confidence	Tags
"I love you"	Non-toxic	0.946	
"You suck"	Toxic	0.995	Obscene Language, Insult
"Your face"	Toxic	0.862	
"Minnesota is a state"	Non-toxic	0.715	
"You are stopid*"	Toxic	0.591	
"I like you"	Toxic	0.504	
"I love you, but you need to shut your mouth"	Toxic	0.995	Obscene Language

Figure 3: Qualitative evaluation of LSTM word level model

Our qualitative analysis reveals that our model is able to fairly confidently identify many forms of toxicity, most notably:

- Very obvious straightforward insults are detected with very high confidence and tagged accordingly as insults.
- Racial/identity slurs are also detected with high confidence and labeled as identity hate, although we did not include these examples here.
- Perhaps most surprisingly, our model has the ability to detect toxicity even through spelling mistakes!
- Neutral statements (such as "Minnesota is a state") are labeled as non-toxic.
- Insults that are tame/immature are also correctly identified, as shown by our model's ability to classify "Your face", a popular but tame insult, as toxic.
- Our model is also able to successfully classify complex sentence structures.

From our qualitative evaluation, it does seem our model is able to confidently identify insults a majority of the time. There are some outlier mistakes (such as the "I like you" input that was identified as toxic), but these occur rarely and usually with very low confidence. Overall, we find that our model is pretty usable from a user perspective and has much potential of being useful in real-world applications, especially after further iteration.

5 Conclusion

Our best models with regards to word-level binary and multi-label classification were our LSTM and CNN models. For character-level binary classification, our CNN models yielded the best perfomance, although overall our word-level models outperformed our character-level models. Additionally, we did try layering a character-level CNN with a bidirectional LSTM as suggested by Nguyen and Nguyen, but we were unable to recreate the high accuracy metrics they attained in their work.

In the future, we aim to achieve higher performance through the use of more complex deep learning models. We would also like to test the effects of using different word/character embedding representations by making refinements on similar lines to the work done by Yu and Wang.

6 References

[1] Yin, Dawei, et al. Detection of harassment on web 2.0. Proceedings of the Content Analysis in the WEB 2 (2009): 1-7.

[2] Huy Nguyen & Minh-Le.N. A Deep Neural Architecture for Sentence-level Sentiment Classification in Twitter Social Networking(2017)

[3] Liang-Chih Yul,Jin Wang,K. Robert Lai,et al. Refining Word Embeddings for Sentiment Analysis

[4] Chu,T & Jue,K. Comment Abuse Classification with Deep Learning

[5]Kim, Yoon. Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 (2014).

[6]Dos Santos, Ccero Nogueira, and Maira Gatti. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. COLING. 2014.

[7]Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. Glove: Global Vectors for Word Representation. EMNLP. Vol. 14. 2014.

[8]A. Muhammad, N. Wiratunga, and R. Lothian, Contextual sentiment analysis for social media genres, Knowledge- Based Systems, vol. 108, pp. 92101, 2016.

[9]N.F.F.DaSilva,E.R.Hruschka,andE.R.Hruschka,Tweet sentiment analysis with classifier ensembles, Decision Support Systems, vol. 66, pp. 170179, 2014.

[10]P. Nakov, A. Ritter, S. Rosenthal, F. Sebastiani, and V. Stoy- anov, SemEval-2016 task 4: Sentiment analysis in twitter, Proceedings of SemEval, pp. 118, 2016.