Toxic Comment Classification

Hewei Cao, Joshua Malmberg, Keenan Byun

1 Introduction

Toxic comment classification is the task of classifying text as abusive or derogatory. Toxic comments can be informally defined as any comment which is rude, disrespectful, or otherwise likely to alienate people from discussions. Such comments make online spaces less inclusive and impede thoughtful discourse. This definition encapsulates people's reactions to toxic comments, however it does not provide insight into the inherent qualities of toxic comments. Furthermore, what is considered offensive is subjective and may differ between individuals, making it difficult to formally define toxicity. As even formally defining comment toxicity is difficult, toxic comment classification is an ideal candidate application for data-driven natural language processing.

The task of toxic comment classification takes as input a text comment and outputs a classification of either toxic or non-toxic. Furthermore, the model should output a multi-label classification specifying the type of toxicity present in the text. Examples of toxicity classes may include 'severe_toxicity', 'insult', etc. Examples of potential toxic and non-toxic input/output pairs are given below:

Input: "People who use spaces instead of tabs are dumb."

Output: toxic; identity_attack, insult.

Input: "Using tabs instead of spaces results in smaller file sizes."

Output: non-toxic.

Ideally, powerful neural classification models could be used to automatically moderate online discussion forums. However, obtaining unbiased data with which to train such models remains a challenge. Training data for toxic comment classification can be imbalanced, with disproportionate large amount of data concentrated in a few toxicity classes and very limited data in other toxicity classes. In this project, we explore various methods to limit the impact of imbalanced data sets on model training and performance.

2 Related Work

Multiple models for toxic comment classification have been presented in the literature. Zaheri, Leath, and Stroud implemented both Naïve Bayes and LSTM classification models for toxic comment classification. They found that the LSTM classifier provided an approximately 20% improvement in the true positive rate, as compared with the widely used Naïve Bayes model. Moreover, the overall performance of the LSTM model, as measured by F1 score, also provided improvement over the Naïve Bayes model (Zaheri, Leath, and Strou; 2020). The LSTM model is an effective model for capturing long term dependencies in input data (Hochreiter and Schmidhuber; 1997). However, the LSTM model is limited in that it can only use context in one direction; conversely, the bidirectional LSTM model can capture context in both directions. In this project we set out to implement to implement both bidirectional and vanilla LSTM models, and expect that a bidirectional LSTM model will perform favorably, as compared with the vanilla version.

To address the imbalance in our dataset, we plan to implement weighted loss functions. Previous research has found that loss weighting can be an effective means of combatting data imbalance in the task of Malware Image Classification (Yue; 2017). In this project, we attempt to apply a similar approach to address the imbalance in our training dataset.

3 Approach

We implemented several classifier models to limit the impact of the imbalanced dataset on model training and performance. Initially, we implemented a variety of preprocessing methods to prepare the dataset before training. We implemented and trained a baseline mode using a Long Short-Term Memory Recurrent Neural Network Classifier. The first improvement method implemented was a bidirectional Long Short-Term Memory Recurrent Neural Network Classifier. The second improvement method implemented was weighted loss functions for the baseline and bidirectional LSTM models.

Dataset Preprocessing

We applied several preprocessing steps to prepare the dataset before using it to train our classifier models. First, we implemented a Python function to filter special characters out of the input comments. Special characters included various punctuation which does not provide semantic meaning in the comments. The complete set of characters filtered was ['`', '~', '!', '@', '#', '^', '(', ')', '[', '{', '}', ']', ';', ':', '*', '$', '%', '&', '-', '_' , '=', '+', '\", '"', '\\', '\n', '\t', '|', ', ', '<', '.', '>', '/', '?']. We then tokenized each input sentence using the “Basic English” tokenizer from the torchtext Python library and fed the resulting tokenized inputs to the torchtext build_vocab_from_iterator utility, producing a vocabulary covering all words in the dataset. The vocabulary was then used to map each tokenized sentence to a sequence of integer id’s, where each id corresponds to a word in the input sentence. Each sequence of id’s was padded to length 100 using the padding token <pad>, and sequences with length greater than 100 were dropped from the dataset.

We also applied preprocessing to the ground truth classification data in the dataset. In the original dataset, multiple classification values were provided that are irrelevant to our projects objective. We dropped these values and only preserved what was relevant, specifically the fields “toxicity”, “severe_toxicity”, “obscene”, “sexual_explicit”, “identity_attack”, “insult”, and “threat”. In the original dataset, each of these fields contained a decimal value in the range [0, 1] which gave a likelihood that that classification applied to the comment. Since we train a classifier model in this project, we set a likelihood threshold and, for each of these fields, mapped the original value to 1 if it was greater than the threshold, and mapped it to 0 otherwise. In other words, if the likelihood score for that class was above the threshold, we classify the datapoint as belonging to the class. For the threshold value we initially tested several values, settling on 0.5. Thus, the ground truth classification for each datapoint was represented using two vectors of 0’s and 1’s: the first vector with degree 1 encodes whether the datapoint is toxic/non-toxic, and the second vector with degree 7 encodes which of the toxicity classes the datapoint belongs to.

Finally, we shuffled the dataset to ensure that the comments were randomly distributed, before applying a test/train split, with a 10:1 ratio of training data to testing data. We further applied a test/validation split to the testing set, with a 10:1 ratio of testing data to validation data. Additionally, we ensured there was an equal proportion of toxic and non-toxic comments in the test set to ensure that the model was not incentivized to produce false positives by the disproportionate ratio of toxic to non-toxic comments in the dataset.

Baseline Model

The baseline model we implemented was a long short-term memory recurrent neural network consisting of 2 main layers: a recursive and a fully connected layer. The recursive layer uses LSTM as unit cells. At each time step, the recursive layer takes as input the index of the next word in the input text, along with the hidden state from the previous step, and produces a new hidden state as output. Word embeddings of size 256 were used. In our current implementation, the word embeddings are not pretrained, but we may later attempt using pre-trained embeddings to improve performance. We used hidden layer size of 500.

The hidden state of the final iteration of the recursive layer is then fed as input to the fully connected layer. The fully connected layer converts the final hidden state into vectors with

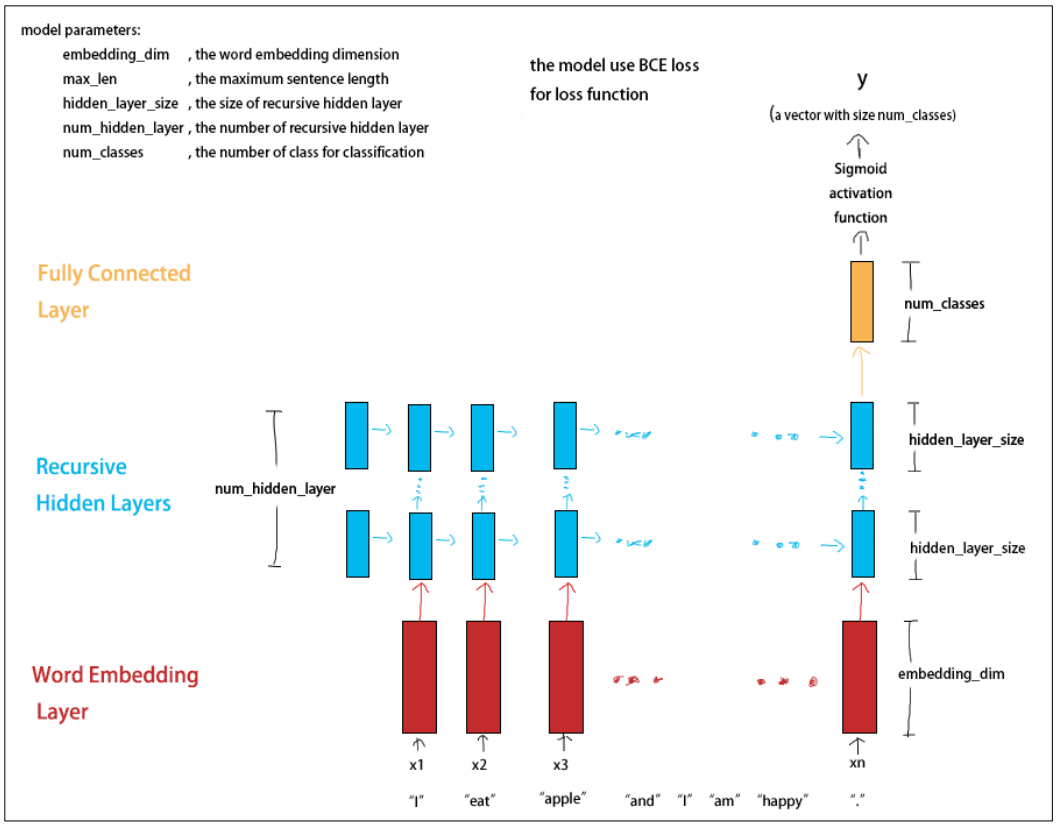

Figure 1: Baseline Vanilla LSTM Model Architecture

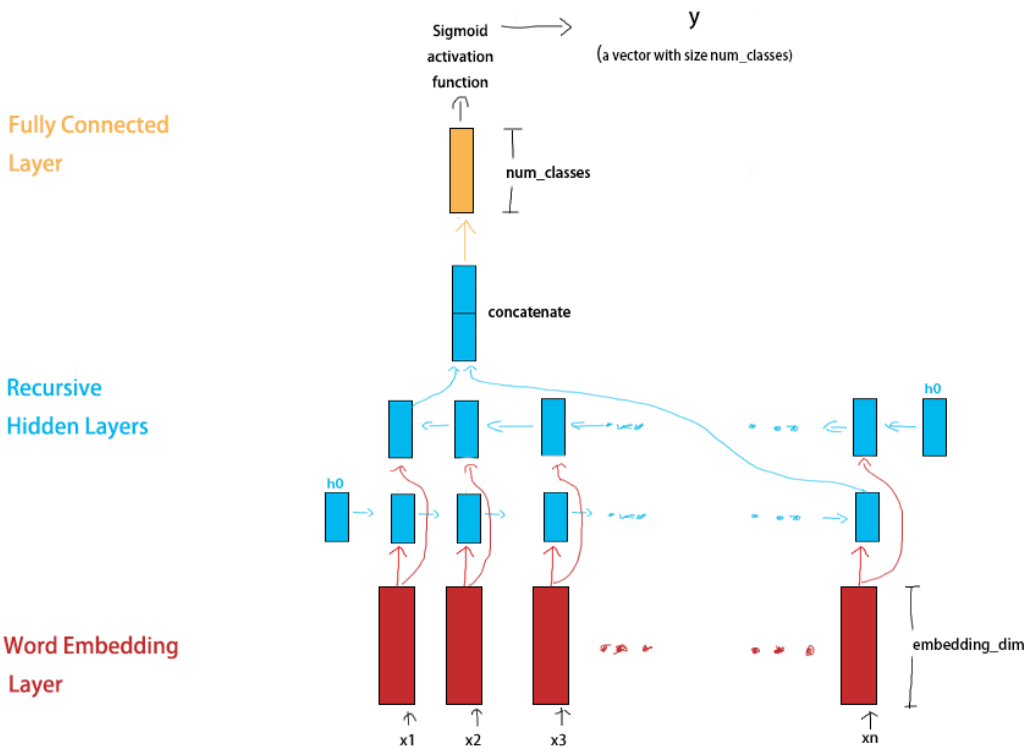

Figure 2: Bidirectional LSTM Architecture

the same form as the label vectors, as described in the preprocessing section. These vectors encode whether the comment is toxic and to which toxicity classes it belongs. Initially, the elements of these vectors provide scores for the likelihood of the input text being either or non-toxic. Thresholding is then applied to produce the classifications.

To increase efficiency, we used batch training to train the model. In our initial training, we used batch size of 128, padded sentence length 100, learning rate 0.001, and trained for 9 epochs. Finally, we use Binary Cross Entropy Loss as the loss function for our LSTM model, with the Adam Optimization Algorithm. Pytorch was primarily used to implement the LSTM model.

Bidirectional LSTM Model

The bidirectional long short-term memory model we implemented was very similar to the baseline model, with the obvious exception of it being a bidirectional architecture. As with the baseline model, the bidirectional LSTM model used random embeddings of size 256, hidden layer size of 500, as well as a fully connected layer to convert the hidden state to the output. The model was trained with batch size of 128, learning rate of 0.001, and trained for 6 epochs. Pytorch was primarily used to implement the bidirectional LSTM model.

Weighted Loss Function

We implemented weighted loss functions as a method to decrease the impact of imbalanced data on model performance. The weighted loss functions were designed to penalize misclassifications of comments from underrepresented toxicity classes more harshly than comments from overrepresented toxicity classes. Two separate approaches to weighted loss were attempted. Consider classes $C_1, C_2, \dots, C_n$. The first weighted loss function is given by:

$$wLoss(x) = \frac{1}{|C_i|} loss(x), where\ x \in C_i$$

To clarify, x represents a particular datapoint for which the loss is being calculated and $|C_i|$ is the number of datapoints belonging to the class C_i. This weighting method decreases the magnitude of the loss for all datapoints, thus there is an absolute decrease in loss for all datapoints.

The second weighted loss function is given by:

$$wLoss(x) = \frac{\min(\{|C_i|\ s.t.\ i \in [1,n]\})}{|C_i|} loss(x),$$

$$where\ x \in C_i$$

Note that this weighted loss function is identical to unweighted loss for datapoints which belong to the class containing the fewest number of datapoints. For all other datapoints, the second weighted loss function decreases the loss proportionally to the size of the smallest class and the classes which the datapoint belongs to.

4 Dataset

As our data set, we will used the data set provided for the Jigsaw Unintended Bias in Toxicity Classification Competition on Kaggle. In total, this dataset provides just under 1Gb of data, stored in .csv format. The data contains text comments with associated toxicity scores, toxicity classes, and target group labels. We intend to only use the comments, toxicity scores, toxicity classes. The dataset considers the toxicity classes 'severe_toxicity', 'obscene', 'sexual_explicit', 'identity_attack', 'insult', and 'threat'. An example datapoint with the relevant fields is provided in the table below:

In the original dataset, likelihood estimates are provided instead of binary classifications. We convert these values to binary classifications before training, as described in the next section. On average, the sentences in the dataset are 65 words long, and we limit the input data for the classifier models to 100 words. In total, there are approximately 1.9 million comments in the dataset, of which approximately 7% are classified as toxic.

This competition originally tackled the problem of unintended bias in toxic comment classification, with focus in bias towards different target groups. The dataset is imbalanced in the ratio of toxic to non-toxic comments by target group. The dataset is even more severely imbalanced in the proportion of toxic comments belonging to each toxicity class, where the largest toxicity class "insult" has nearly ten times as many datapoints as

each of the other classes. Naturally, the toxicity classes which have more training data will be predicted more accurately by classification models, whereas toxicity classes with less data will be predicted poorly. Or goal in this project is to address this imbalance in the data and increase the overall model performance by better predicting the underrepresented classes.

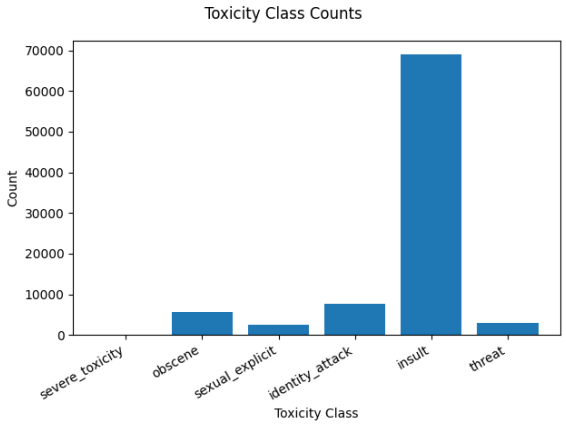

Figure 3: Data Quantity by Toxicity Class

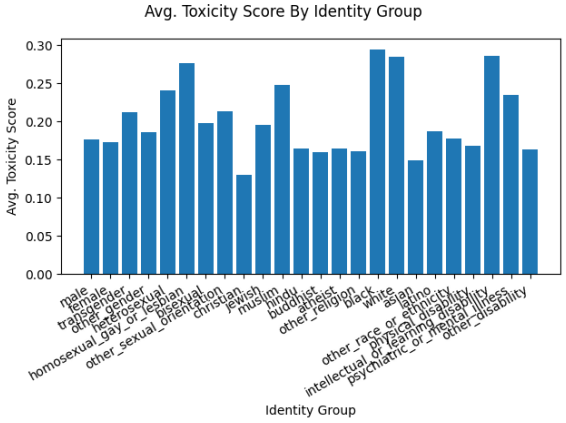

Figure 4: Avg Toxicity Score by Target Group

5 Evaluation Metrics

As our models implement a multi-label classification task, the loss function must account for the fact that the models output prediction may be partially correct, ie. it may correctly predict some of the toxicity classes that a datapoint belongs to but miss others. As such, we use recall and precision metrics defined as follows:

$$Precision = \frac{|T \cap P|}{|P|}, \quad Recall = \frac{|T \cap P|}{|T|}$$

Element a_{ij} means that comment i has label j. If $a_{ij} \in T$, then sentence i has label j in ground truth. If $a_{ij} \in P$, then the model predicted that sentence i has label j. The F1-score is obtained by taking the harmonic mean of precision and recall. Since the dataset is heavily imbalanced, we use F1-score as our main evaluation metric to obtain a representative measure of overall performance.

6 Results and Analysis

We tested model performance for the vanilla and bidirectional LSTM models with standard loss functions, the vanilla LSTM model with both two weighted loss functions, and the bidirectional LSTM with the second weighted loss function. We benchmarked both the binary classification performance (toxic vs. non-toxic) and the multilabel classification performance (toxicity class). The results are provided in the table below.

Model	Precision	Recall	F1
Vanilla LSTM	0.948	0.674	0.788
Bidirectional LSTM	0.910	0.770	0.834

Table 1: Binary Classification Performance

Model	Precision	Recall	F1
Vanilla LSTM	0.906	0.870	0.906
Bidirectional LSTM	0.940	0.917	0.930
Weighted 1 w/ Vanilla LSTM	0.908	0.823	0.863
Weighted 2 w/ Vanilla LSTM	0.929	0.892	0.910
Weighted 2 w/ Vanilla LSTM	0.940	0.892	0.916

Table 2: Multilabel Classification Performance

Considering that the testing set had equal proportions of toxic and non-toxic comments, achieved binary classification performance is significant. The model we implemented compares favorably with the LSTM model implemented by Zaheri, Leath, and Strou, which achieved F1 score of 0.73, albeit on a different data set (Zaheri, Leath, and Strou; 2020). Conversely, the testing set was not designed to have equal representation for all toxicity classes and a single toxicity class accounted for most of the data. Even using the F1

score, it is difficult to assess how effectively the multi-label classification models performed, given the imbalanced data set.

We observed that the bidirectional LSTM model with normal loss function performed best in both the binary and multi-lable classification tasks. Although the second weighted loss function did provide slight improvement when applied to the vanilla LSTM model, the weighted loss functions required significantly longer training times and overall did not provide a significant performance advantage over the unweighted loss function models.

Qualitatively, we expected that the bidirectional LSTM would provide improved integration of bidirectional context when making binary toxicity classifications. However, we observed that the bidirectional LSTM model tended to focus more on keywords and ignore context in some instance. For example, in the table below the first two comments are not toxic but were nonetheless classified by the bidirectional LSTM model as toxic with high certainty because they contain a certain derogatory keyword. In this instance, the vanilla LSTM model surprisingly seems to perform better.

Comment	Toxicity Score, Vanilla	Toxicity Score, Bidirectional
"The man is f****** cool."	0.790	0.982
"The taste is f****** great."	0.719	0.921
"You are a f****** idiot."	0.983	0.997
"You f****** dirty sh**"	0.982	0.993

Table 3: Vanilla and Bidirectional LSTM Performance on Toxic and Non-Toxic Comments with Derogatory Keyword

7 Limitations

The model design, evaluation, and training approach all had limitations in this project. Most significantly, our model design was limited by the fact that we used randomized word embeddings. In the case of toxicity classification, the semantics of the words in a sentence are pivotal to correctly classifying the sentence. We attempted to use the pretrained GLoVe embeddings but resolved to use randomized embeddings after finding that a significant portion of the words in our data set were missing in the GLoVe vocabulary. Finding a way to effectively use pretrained embeddings or using a transfer learning model such as BERT would be a promising avenue for future research. Another interesting change we could make to our model in future research is designing and training for toxic comment regression, rather than classification. As the original dataset includes data ideally suited for training a regression model, we may be able to obtain better results by training our model for regression and then applying the threshold to its output only when the model is used in application. Dataset augmentation may also be an effective means to better utilize the training data.

Our evaluation was limited in that we did not perform separate evaluations of the model performance for each toxicity class. As a result, we cannot directly compare the relative performance achieved for each toxicity class by each model. Rather, in our findings we compare the overall performance achieved for all toxicity classes by each model. As such, we were not able to obtain detailed insights into the specific performance obtained for each toxicity class. In future research, we will alter our evaluation model to evaluate the performance on each toxicity class separately.

Finally, our training was limited in that we only trained the weighted loss models for a limited number of epochs, specifically 12. As discussed in the approach section, weighting the loss function results in lower loss magnitudes on average, which would naturally tend to reduce the training rate. It is possible that the weighted loss models underperform compared to the bidirectional LSTM because of under-training. In future research we intend to increase the training rate and epochs for the weighted loss models to explore this issue.

8 Conclusion

We found that the bidirectional LSTM outperforms the vanilla LSTM model at both the binary and multi-label classification tasks. However, our expectation that the bidirectional LSTM would derive performance improvement from improved use of context was contradicted by several qualitative examples we analyzed. We were unable to conclude that loss function weighting effectively addresses the issue of imbalanced data for the multi-label classification problem; however, we did observe that loss function weighting results in longer training time. Further exploration is required to better evaluate the effectiveness of loss function weighting, including implementing more effective evaluation metrics. Finally, future exploration should include attempts to incorporate transfer learning or pretrained word embeddings in the classification model, as these tools seem to be ideally suited for the task of toxic comment classification.

9 Contributions

Keenan Byun: Implemented data pre-processing methods, loss function weighting, and debugged LSTM models.
Hewei Cao: Implemented LSTM models and evaluation metrics.
Joshua Malmberg: Wrote and typeset final report, project milestone, project proposal, and project abstract. Wrote presentation slides. Delivered final presentation.

References

Zaheri, Sara; Leath, Jeff; and Stroud, David (2020) "Toxic Comment Classification," SMU Data Science Review: Vol. 3 : No. 1 , Article 13.

Mozafari, Marzieh; Farahbakhsh, Reza; and Crespi, Noel (2019) " A BERT-Based Transfer Learning Approach for Hate Speech Detection in Online Social Media," Complex Networks and Their Applications: Vol. 8 : No. 1, pp. 928-940.

Yue, S. (2017). Imbalanced malware images classification: a cnn based approach. *arXiv preprint arXiv:1708.08042*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.