Toxicity Prediction of Online Comments with Machine Learning and Deep Learning Models

Viraj Maddur*
Florida State University Young Scholars Program
vamaddur@gmail.com

Dr. Weikuan Yu†
Florida State University Computer Science Department
yuw@cs.fsu.edu

ABSTRACT

Surges of information are produced on a daily basis through Internet usage stemming from communications between users around the globe. While this connectivity has its benefits, the potential of online texts with high toxicity to harass/bully netizens is dangerous. The attempts of researchers to create an efficient model for online toxic comment prediction are still in the early stages, and require new approaches and frameworks to tackle the problem. This paper introduces and discusses refinements of logistic regression and neural network models made with Python libraries for toxicity prediction through binary classification for six different categories. An optimization for the accuracy of the logistic regression model using ternary searches on the maximum features parameter of vectorizers is implemented. The neural network model is coded with multiple layers, including a hidden convolutional component, to mirror the processing of text that occurs in the human brain and improve the accuracy of the model over time. Our results did not show a statistically significant difference between the two models in terms of accuracy, but the neural network possessed many more areas of possible improvement.

KEYWORDS

Text Classification, Convolutional Neural Networks, Binary Classification, Logistic Regression, Word Embeddings, Character n-grams, Vectorizers, Ternary Search

1 INTRODUCTION

Chat and comment systems are ubiquitous in any given online setting, but threats and harassment often prevent people from using them to the fullest extent for self-expression and seeking out new viewpoints. Platforms for these systems struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user chatting and comments. When classifying toxic (e.g. rude or disrespectful) comments, current publicly available models are not always accurate and do not allow users to select subcategories of toxicity to examine. Various organizations and data science organizations are interested in taking on the problem to the point of offering bounties up to $35,000 on sites such as Kaggle for the best solutions [3]. Recent developments in the field of natural language processing suggest that machine learning is a promising option for creating models to predict the classification of texts without using prior syntactic or semantic knowledge of a language. A logistic regression model with some transformation of the data can serve as a straightforward starting point for this toxic text classification problem. The work of Georgakopoulos et al. [2] provides evidence that convolutional neural networks (CNNs) using character-level features is an effective method, with a model that has over 90% accuracy on testing data. A multi-layer take on the CNN idea will be explored in addition to the aforementioned logistic regression model.

1.1 Data Collection

For obtaining the original data used to train the model, a dataset from a Kaggle competition was used [3]; over 150,000 English comments from the user talk and article talk namespaces/forums of Wikipedia (the two most active ones on the site) along with their revision history are included in the dataset as a CSV file. Before the dataset was finalized, administrative comments following a template and messages generated by a bot were filtered out by regular expressions to achieve a set of only human-generated texts [6]. Each comment is stated to be included or not included in each of the six following categories (they may belong to more than one of them): toxic, severe toxic, obscene, threat, insult, and identity hate.

2 LOGISTIC REGRESSION WITH SK-LEARN

SciKit-Learn (SK-Learn) is a popular machine learning library for regression models in Python [5]. Word and character vectorizers are used to process the dataset before plugging it into the actual model and a ternary search optimizes the accuracy obtained by plugging in the data transformed by the vectorizers.

2.1 Word and Character Vectorizers

The vectorizers essentially build a dictionary with "important" words and sequences of characters based on the training data texts that allows us to transform sentences of sparse data into vectors of dense information that can be processed by a model.

2.1.1 TF-IDF Component. The tf-idf (term frequency – inverse document frequency) vectorizer used from the SK-Learn library takes into account the number of times a string appears (term frequency) and the number of different texts a string appears in (document frequency) to determine the "importance" of including it in a dictionary. The weight $w_{i,j}$ of a term i in a document j is:

$$w_{i,j} = tf_{i,j} \cdot \log \frac{N}{df_i},$$

where $tf_{i,j}$ is the number of occurrences of i in j, df_i is the number of documents containing i, and N is the total number of documents. Note that a heavier weight corresponds to a higher priority for

*Main Author
†Supervising Professor

being included in the created dictionary. A model trained on non-weighted data may miss some revealing information brought by other less frequent words.

2.1.2 Character n-grams Component. A parameter of the character vectorizer is the character n-grams setting. People often attempt to obfuscate profane words with additional characters (e.g. wow becomes wooooowwww), but the use of character n-grams can potentially detect these instances by checking smaller sequences of characters for similarities as opposed to full words (the dictionary from the word vectorizer still has use, as it is better suited for analyzing broader context in comments). Character sequences of length from 2 to 6, inclusive, are analyzed by the character vectorizer for the purposes of this paper.

2.2 Ternary Search Optimization

Assume that there is an optimum number for the maximum features (size of dictionary) in each of the vectorizers. As we use smaller and smaller numbers below the optimum, the model will perform with less accuracy on the test data due to a dearth of information to make predictions (undertraining). On the other hand, using larger and larger numbers above the optimum will cause the model to perform with less accuracy on the test data due to using unnecessary features to make predictions (overfitting). These observations deem the accuracy of the model as a convex function with a maximum value, making it a candidate for a ternary search. Figure 1 displays the essence of the algorithm in context, using a mixture of Python and pseudocode to convey the code idea.

```
lo = lower_bound, hi = upper_bound
while lo < hi:
    t1 = (lo*2+hi)//3, t2 = (lo+hi*2)//3
    model_t1 = get_model(t1), model_t2 = get_model(t2)
    cv_score_t1 = cross_val(model_t1, train), cv_score_t2 = cross_val(model_t2, train)
    if cv_score_t1 > cv_score_t2:
        hi = t2-1
    else:
        lo = t1+1
```

Figure 1. Ternary Search Code Idea

2.3 Modeling Procedure

The optimum settings for the maximum feature parameter are approximately 5200 for the word vectorizer and 55000 for the character vectorizer after running the ternary search. As mentioned earlier, the data can fall under more than one category, so we are dealing with several different binary classification problems; this is handled by simply creating independent submodels for each category to decide if a comment/entry fits in it. For each of the six classes, every comment/entry of the training data is fed into a logistic regression submodel as a pair of vectors, one of words and another of character sequences, to calibrate it. The submodels output the probability of each comment in the test data falling under its corresponding class.

3 NEURAL NETWORK WITH KERAS

Tensorflow is an open-source machine learning framework for Python, and Keras is an API that runs on top of Tensorflow to provide additional prototypability for neural network models [4]. Data entries are padded before being given as inputs to the multi-layer model.

3.1 Tokenizer and Sentence Padding

The Tokenizer class in Keras utilizes character-level patterns to condense the text data while preserving the most important features (in a similar fashion to the vectorizers). Parameters are set to the default values with the exception of the maximum features being fixed to 20000. A neural network model can only accept a stream of data with consistent length, but the lengths of comments in the data is highly variable. The shorter comments must be padded with additional "ghost" features and the negligible parts of longer comments need to be truncated. The 75^{th} percentile of tokenized word counts in the training data is 435, but the padding length is set to 450 just to be safe and keep the dimensionality of the problem low while losing little critical information. Figure 2 shows a distribution of comments based on their tokenized word count; the left-skewed nature of the data caused a low padding length to be selected.

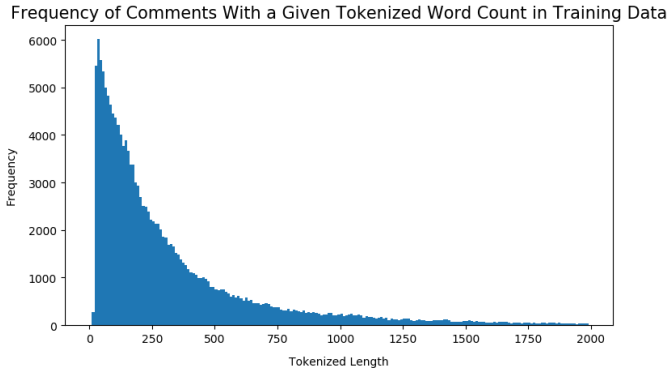

Figure 2. Frequency of Comments With a Given Tokenized Word Count in Training Data

3.2 Model Layers

After the padding process transforms the input, the data still has to go through eight layers of the designed neural network before arriving at output. The types of layers used are the following:

(1) *Embedding Layer*, is a special component of neural networks for text classification problems. Each word of a text document is transformed into a dense vector of fixed size, allowing the inputs to be processed by the next layers of the model.
(2) *Convolutional Layer*, is the core part of CNNs that is made up of "neurons" with learnable weights and biases. Each "neuron" is connected only to a small chunk of the input from the previous layer (just as in a human brain), but downsizes its piece of the data without losing its essence for use in further processing.
(3) *Bidirectional Long Short Term Memory (LSTM) Layer*, is an extension of traditional LSTMs that can improve model performance on sequence classification problems. The use of bidirectionality in speech classification is based on evidence that the context of the whole comment is used to interpret

what is being said rather than a linear interpretation [1]. The layer acts by recursively feeding information from previous iterations of it in both directions to improve the identification of important information in data.

(4) *Max Pooling, Dense, and Dropout Layers*, are essential pieces of an accurate neural network model. Max pooling reduces the dimensionality of the problem by downsampling input representations and allows for assumptions to be made about features contained in the sub-regions binned. Dense layers have every input in the previous layer of high-level features in the data connected to every output in the next layer for the model to learn non-linear combinations of these features. Dropout layers randomly deactivate neurons in the previous layer with the aim to reduce the complexity of the model and consequently prevent overfitting.

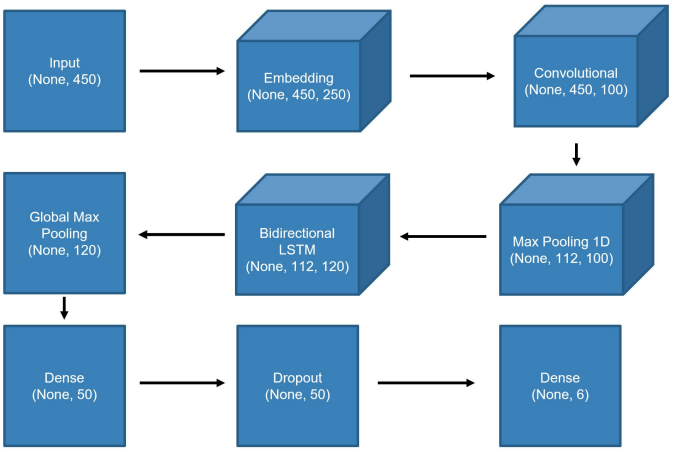

Figure 3. Neural Network Model Layers Flow Chart

4 EVALUATION AND RESULTS

The average of the ROC-AUC (Area Under the Receiver Operating Characteristic Curve) scoring method over the six classes was used to judge the performance of the models. This method takes into account the false-positive and false-negative rates to measure how well a model can discriminate between groups in a binary classification problem. More detailed results on the test data could not be included due to the opaque nature of Kaggle's scoring system and its test dataset (only a few comments/entries had their true classifications released). Timings were determined by running code on Kaggle's online kernels.

4.1 Logistic Regression Results

The model attained an average value of 0.9795 out of 1.0000 in the training data and 0.9764 out of 1.0000 in the test data over the six categories. The overall running time was approximately 35 minutes, with the majority of time (approximately 20 minutes) being spent to build the character vectorizer and then using it to transform the data. Figure 4 shows detailed results for individual categories from the training data.

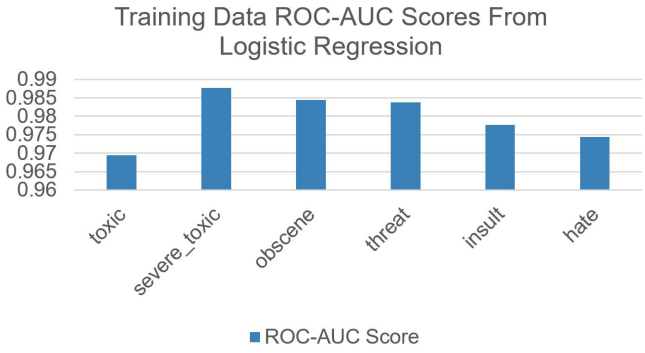

Figure 4. Training Data ROC-AUC Scores From Logistic Regression

4.2 Neural Network Results

The model attained an average value of 0.9823 out of 1.0000 in the training data after the sixth epoch and 0.9723 out of 1.0000 in the test data over the six categories. The overall running time was approximately 276 minutes, more than 6 times that of the logistic regression. The accuracy of the training data overtaking that of the validation data in the 3rd epoch indicates that the network is learning and changing its weights to better its performance in the following epochs as seen in Figure 5.

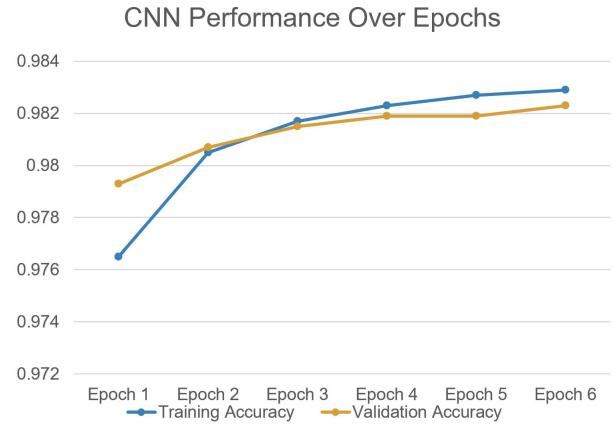

Figure 5. CNN Performance Over Epochs

5 DISCUSSION AND CONCLUSIONS

The logistic regression and CNN models both have the potential for improvement. The accuracy of the logistic regression model could be improved by increasing the range of character n-grams to consider. Although the dimensionality of the model is increased, it is still less computationally intensive than its neural network counterpart. Adding more layers to the neural network could improve the accuracy of the model, but may risk overfitting the training data, while removing layers could improve the running time on a dataset at the cost of accuracy. The multi-layer nature of the CNN allows for many possible combinations of fine-tuning measures (e.g. tweaking dimensionality) to improve accuracy that could be

explored in future works. The simpler logistic regression model did not outperform the neural network by a statistically significant margin in the tests, but there exist many more refinements to the latter model that could allow it to surpass the former.

A GITHUB

The code written/used in this paper can be found under the username of "vmaddur" in the "Comment-Toxicity-Prediction" repository at https://github.com/vmaddur/Comment-Toxicity-Prediction.

ACKNOWLEDGMENTS

The authors are thankful for the assistance of Amit Nath and Harsh Kundnani from the computer science department in writing this paper and for having this opportunity provided by the Young Scholars Program through Dr. Erica Staehling and Barbara Shoplock.

REFERENCES

[1] Jason Brownlee. 2017. How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras. (Jul 2017). https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

[2] Spiros V. Georgakopoulos, Sotiris K. Tasoulis, Aristidis G. Vrahatis, and Vassilis P. Plagianakos. 2018. Convolutional Neural Networks for Toxic Comment Classification. *CoRR* abs/1802.09957 (2018). arXiv:1802.09957 http://arxiv.org/abs/1802.09957

[3] Kaggle Division of Google. 2018. Toxic Comment Classification Challenge. (2018). https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

[4] Google Brain Team. 2018. Tensorflow API Documentation. (2018). https://www.tensorflow.org/api_docs/

[5] Scikit-Learn Team. 2017. Documentation of scikit-learn 0.19.2Âű. (2017). http://scikit-learn.org/stable/documentation.html

[6] Ellery Wulczyn, Nithum Thain, and Lucas Dixon. 2016. Ex Machina: Personal Attacks Seen at Scale. *CoRR* abs/1610.08914 (2016). arXiv:1610.08914 http://arxiv.org/abs/1610.08914