# Exploration of Supervised Learning Methods to Identify and Classify Online Toxic Comments

**Andrew McLean**
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90007
*amclean@usc.edu*

**Nikhit Mago**
Department of Computer Science
University of Southern California
Los Angeles, CA 90007
*mago@usc.edu*

**Anmol Chawla**
Department of Computer Science
University of Southern California
Los Angeles, CA 90007
*anmolcha@usc.edu*

## Abstract

Abuse and harassment online has negatively impacted a user's right to freedom of expression. The presence of hate on the internet is partly a result of established policies inefficiently identifying and removing hateful speech. The contribution of this paper is to explore methods that combine machine learning and natural language processing to automate hate speech detection online. Given a publicly available text corpus of over 300,000 comments from Wikipedia's talk page, we extract relevant text features and allow our classifiers to predict the probability of a comment being labeled as toxic according to predefined categories. By tuning parameters, we compare performance as measured by the area under ROC curve and hamming loss to assess the significance of different experimental settings. In modeling our problem as a multi-label binary classification problem, our methodology compares the toxic labels predicted from our models with those created from human moderators to evaluate the accuracy of our classifiers. The results of our experimental models make LSTM and Naïve Bayes classifiers promising candidates for automating toxic comment detection online at scale.

## 1 Introduction

Since the rise of social media platforms, online discussion has become an essential part of people's day-to-day experience with the internet [6]. However, the anonymity of such online discussions has created an avenue for people to abuse forums and spread hate throughout the internet [7]. According to a study done by Pew in 2014, 73% of adult internet users have seen someone harassed online, and 40% have personally experienced it [4]. To combat these negative online behaviors, platforms develop policies to ensure no hate is spread on their platform. One such example would be Wikipedia's zero tolerance policy, which is clearly outlined within their talk page guidelines. However, human moderators are the ones responsible for reading and identifying toxic comments through a practice known as crowdsourcing. A sizeable number of other online video and text content based websites have similar policies, where the approach of gathering human annotated labels is not a scalable solution both in terms of time and money [8]. The scalability shortcomings and the inherent bias of human judgement make this approach not

only costly, but also difficult to effectively identify and appropriately respond to harassment online.

With the advancements of machine learning in the domain of natural language processing (NLP), researchers have developed tools to automate the task of identifying hate speech online. Recently, a subsidiary of Google released a tool called "Perspective" that can decide if an online comment is "toxic" without the aid of human moderators [11]. These efforts have given websites an affordable way to let their readers participate in online discussions in a civil and respectful manner. However, this tool is not a panacea for removing hate online as it still makes errors. The Perspective tool is just a baseline from which other researchers can build on and improve.

In a recent Kaggle competition, participants were invited to explore statistical techniques that would improve the detection abilities of the Perspective tool [2]. The given dataset consisted of numerous comments taken from Wikipedia talk page edits, that were labeled by human raters as toxic behavior based on the following types of toxicity: 'toxic', 'severe toxic', 'obscene', 'threat', 'insult' and 'identity hate'. In a supervised learning framework, participants were asked to build a model that estimated the probabilities of comments being categorized as toxic respective to each level of toxicity.

Our project aims to help improve online conversation by exploring various machine learning methods to build an accurate model that's capable of detecting diverse types of negative online comments perceived as toxic. A sub-goal of our project is to match or beat the score of the best classifier on the Kaggle leader, 0.9885, with the overall goal of building on existing models and improving the accuracy of toxic comment detection.

## 2 Background and Related Work

Detecting abusive language online is often more difficult than one expects. The noisiness of the data coming from differing sources in combination with a need for universal knowledge of toxic behavior are reasons that make automated detection a challenging task. Furthermore, the complexities of natural language constructs and spelling inconsistencies of user-generated comments make ordinary NLP approaches inefficient detectors of intentional hurtful language. Therefore, much of the recent work in automated detection builds on existing supervised learning classification methods in conjunction with NLP feature extraction.

One such approach attempted to differentiate hate speech from offensive language, using a classifier based on the Naive Bayes assumption, decision trees and SVM [10]. More recently, results from a similar work showed that simple n-gram features of text are more powerful than those of hand-engineered lexicons and word embeddings [5]. In our work, we explore the direct and indirect features of our text data to gain a better understanding of our dataset. However, we simply model our corpus using bag-of-words representation with word level n-grams and calculate the relative count features of our text to be used in our models.

## 3 Analysis of the Data Set

The Wikipedia data set was already split into training and testing sets in accordance with the rules of the competition. Below are descriptions of each of the data sets.

Train Data: Consisted of 159571 rows and eight columns. The column headers were id, comment_text, toxic, severe_toxic, obscene, threat, insult, identity_hate. The total size of the train data was 23.6 MB.

Test Data: Consisted of 153164 rows and two columns. The headers were id and comment_text.

As you can see from the above description, the testing data set does not have labels associated with the comments. For this reason, we did not use the test set to assess the accuracy of supervised learning methods. However, we did use the comments of this set as part of our overall corpus to have a wider range of words available to extract features from.

### 3.1 Exploratory Data Analysis

Before cleaning and processing our data, we explore the data and comment on observations that are relevant to the identification of toxicity. Our observations are listed below:

Observations:

- Out of the 159,571 comments in training set, 143,346 were clean indicating no tags across labels.
- Total tags were 35,098, which is higher than the total comments indicating multi-tagged comments.
- The column 'comment_text' had IP addresses and other irrelevant text unique to this data set.

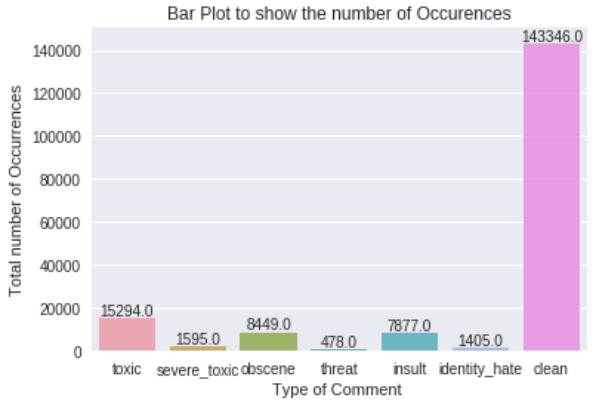


Figure 1: Plot showing number of occurrences of each label in training set.

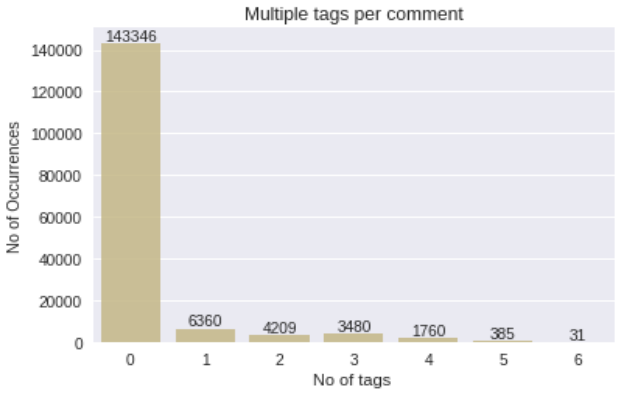


Figure 2: Plot showing number of tags per comment in training set.

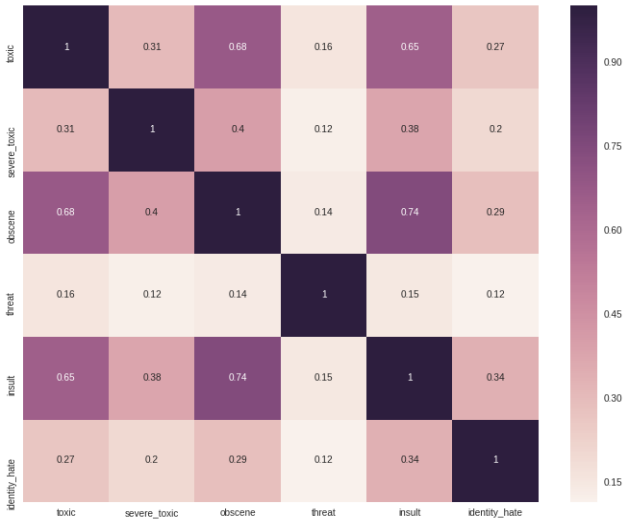


Figure 3. Correlation matrix of comments.

From the bar plot shown in Figure 1, it is evident that there is class imbalance due to uneven distribution of comments with tags and comments classified as clean. Additionally, the plot in Figure 2 indicates that the same comment can have multiple tags. Another observation is that out of the tagged comments, more than half have two or more tags. Hence, for a stable model we need to characterize the likelihood of a comment being in each label as a separate binary classification problem. Therefore, we would need to find the class conditional probabilities for each label and set a threshold to determine if a tag should be assigned to it or not. The correlation matrix shown in Figure 3, indicates the significance between certain labels occurring in a multi-tagged comment. For example, we can see that there is strong correlation between comments tagged as obscene as well as toxic. Therefore, during feature selection we can use this intuition to determine notable features of grouped tags in multi-tagged comments.

Before cleaning our data, we extracted some basic statistical features: word count, unique word count, sentence count, capital word count, special character count, percentage of unique words and percentage of capital words per label. Although these statistical features give us a better understanding of user generated text, prior work has shown that human engineered feature lists don't perform well in comparison to n-gram count features [3]. Therefore, we will not include these statistical features in our models.

### 3.2 Preprocessing

As mentioned earlier, we combined the comment text column from both the training and testing sets to build our corpus. A common first step in preprocessing is to ensure there are no missing values. Since there were no missing values, we continued with steps to clean our corpus. We want to learn meaningful features and not have our model overfit to noise. Therefore, we first removed all IP addresses, dates and other numbers because the number of comments containing them was small. The similarity of our comments to Tweets, led us to use the packaged TweetTokenizer to convert each word into a unique entity. We then applied a custom lexicon normalizer which made all letters in a word lowercase and converted slang words to proper English. For example, the text "i've" changed to "i have" and the same methodology was applied to other conjunctions. The last step was to lemmatize the words. We used the Lemmatizer from Wordnet to stem words to their roots, which reduces words with the same meaning to a constant form. For example, the words "running" and "ran" get converted to "run," which is the respective root form.

## 4 Feature Extraction

After obtaining a clean corpus, we performed feature extraction to obtain relevant information for our classifiers. One characterization of text that has proved to perform well in sentiment analysis, is TF-IDF, which stands for term frequency – inverse document frequency. This weighted term frequency metric is specifically useful for our model formulation because we did not end up removing the stop words from the text corpus. The TF-IDF metric is calculated by the following equation:

$$TF - IDF = tf_n(t,d) \times idf(t)$$

$$idf(t) = log\left(\frac{n_d}{n_d(t)}\right)$$

$n_d$: the total number of comments in the corpus

$n_d(t)$: the number of comments that contain term $t$

Our models represent the corpus either as unigrams or bigrams corresponding to sequences of one or two words as input to our classifiers. Therefore, we calculate the TF-IDF metric for both n-gram representations. For each label, we created a bar plot showing the top n-grams seen in each class based on the TF-IDF metric. Below are plots for the toxic class.

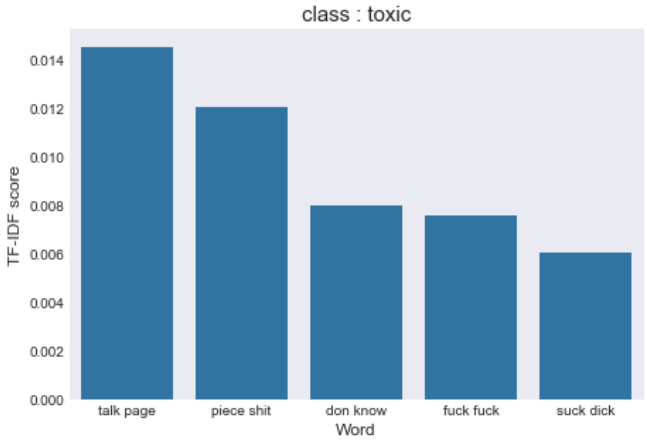


Figure 4. Top 5 TF-IDF unigrams in toxic class.

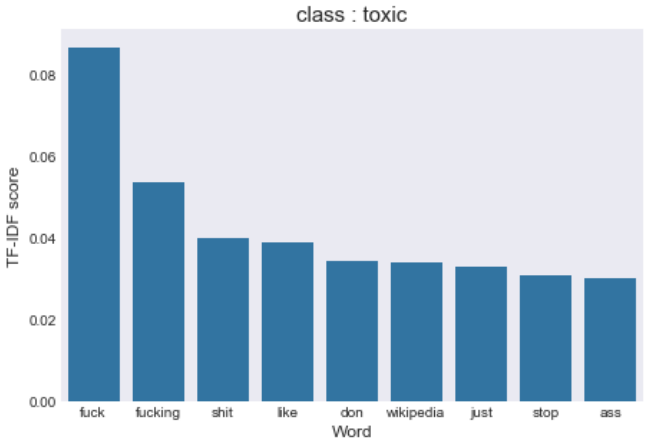


Figure 5. Top 9 TF-IDF bigrams in toxic class.

From the bar plots in Figure 4 and 5, it is evident that the words which are important in the unigram model are also important in bigram model. The text sequences with corresponding TF-IDF values will be the input into the models described in the next section.

# 5 Methods of Model Building

## 5.1 LSTM

In recent years, deep learning has emerged as a powerful machine learning technique and LSTM networks have made significant strides in the domain of sentiment classification. LSTM stands for long-short term memory and the term refers to the type of cell unit that is used to build the layers of a recurrent neural network (RNN). LSTMs are commonly used in sentiment classification because of their ability to use longer-term contexts and because the LSTM cell also prevents the problem of vanishing gradient, which is a significant drawback of certain RNNs. We chose to build a model using LSTMs because they had previously worked well on Tweets [12] and the length of the comments in our dataset are comparable to the length of tweets. The first layer of our network accepts sentences of a predefined size of 200, with comments smaller than that number being padded with zeros. The next layer is our embedding layer where we project the words to a defined vector space depending on the distance of the surrounding words in a sentence. We chose the dimensionality of the word embedding layer to be 128. Next, we feed the output into the LSTM layer, and the output of that layer is fed into a Max pooling layer for dimensionality reduction. The next few layers take advantage of dense and dropout layers to reduce overfitting in our model. Finally, the output layer is defined to have 6 cells corresponding to the 6 labels and the sigmoid activation function was used to map a probability between 0 and 1. Lastly, we defined our loss function to be binary cross entropy and used the Adam optimizer for optimization of our loss function during the training process.

## 5.2 Multinomial Naïve Bayes

Multinomial Naive Bayes is a version of the Naive Bayes algorithm that works well with textual data. Naive Bayes classifiers work on the principle of Bayes theorem with a strong assumption of independence between the features. An advantage of this classification method is that it works well with less training data, as it only calculates conditional probabilities for the feature space. In Multinomial Naive Bayes, the conditional probability distribution is a multinomial distribution, which works well for data containing word count features [9]. We used Laplace Corrections while implementing this model to avoid 'Division by zero' errors. The model took around 4-5 seconds to train the sparse input set and was much faster than the other algorithms.

## 5.3 Naïve Bayes SVM

Here we present a simple model variant where a SVM discriminative classifier is built over a generative Naive Bayes approach that models log-count ratios as feature values. Based on the results of prior work using this approach, we use bigram features for the topical classification tasks in this problem [1]. From the IDF log-count feature values, our model estimates the class-conditional probability of encountering a text vector **x** for a specific label for each comment in the corpus. Since there are six different categorizations of toxicity for this problem, we fit a model to each label where each learning task is a binary classification task. After computing the class conditional probabilities, we used the optimal probability threshold seen from the ROC curve to force values to go to either 0 or 1 and analyzed the results.

## 5.4 Random Forest

Random Forest is a state of the art ensemble classification algorithm that works on the principle of Bootstrapping and has proven to show remarkable results [13]. The algorithm combines the decision of several Decision Trees using a voting approach. Random Forests mitigate the effects of over-fitting observed in Decision Trees by using bootstrapped samples and a subset of variables in each tree. Instead of using cross validation, we used Out of Bag errors to evaluate our training

sets. A total of 1000 trees were used to build the Random Forest and approximately 18 features out of a possible 300 were randomly chosen at every decision node.

### 5.5 Logistic Regression

Logistic Regression, a linear classifier was trained on the data with L2 regularization and 3-fold cross validation was used to choose the appropriate L2 penalty. We used L2 penalization in this task to overcome overfitting as the feature space was huge for the classification. Logistic Regression is a simple linear classifier that uses the sigmoid function to predict probabilities for each observation. We used ROC curves, explained in the next section, to use these probabilities for thresholding and coming up with accurate predictions. This classification algorithm best works with linearly separable data and fails to classify for data that have intricate non-linearities.

### 5.6 XGBoost

XGBoost is short for "Extreme Gradient Boosting", where the term "Gradient Boosting" is proposed in a work by Friedman [15]. One of the popular decision tree algorithms, this model was used to derive a split directly from the loss function. Another reason to try XGBoost was the historical significance of the model in past Kaggle competitions. More than fifty percent of the competitions which did not use deep learning were won by XGBoost. The feature list fed into this model was a combination of unigrams and bigrams and the experimental parameters used as well as the results are explained in the next section.

## 6 Model Evaluation

### 6.1 ROC Curves

ROC curves provide an excellent measure of performance for binary classification tasks with imbalanced classes and use the concept of Area Under the Curve. The True Positive Rate (TPR) and False Positive Rate (FPR) are plotted on the curve for varied probability thresholds. For an imbalanced binary classification problem like ours, it is not wise to use a 0.5 probability threshold for the segregation. So instead, for each class, we chose the probability threshold from the ROC curve that maximized the difference between the TPR and FPR and this threshold was observed to be as low as 0.01 for some of the classes due to large imbalance. Since this was a multi-label classification problem, we chose to report the mean AUC for all classes in our results section.

### 6.2 Hamming Score

Although the competition did not require us to calculate this metric, we thought of including it in our project because of the multi-label classification task. Hamming score is a metric that calculates a percentage of correct classification for all classes per observations, averaged over all observations. A higher score is desirable for multilabel classification tasks and it summarizes the results well [14].

## 7 Experiments, Analysis and Results

Our experiments for machine learning algorithms were divided into two broad settings. The parameters chosen for the two settings are tabulated in the table below in Table 1:

Table 1. Experimental Settings

| Tuning Parameters | Set 1 | Set 2 |
|---|---|---|
| Min Word Len | 3 | 3 |
| Ngram Rng | (1,2) | (1,2) |
| Max Doc Freq | 50,000 | 50,000 |
| Max Feat | 300 | 1,000 |

As observed from the table above, the first 3 parameters were common for both the settings. We chose both unigrams and bigrams in our feature list, and used a maximum document frequency of 50,000, which means that the selected tokens were in a maximum of 50,000 documents. The results for both the settings are tabulated below in Table 2:

Table 2. Experimental Results

| Model | Set 1 Mean AUC | Set 2 Mean AUC | Set 1 Hamming Score | Set 2 Hamming Score |
|---|---|---|---|---|
| Log Reg | 0.92 | 0.96 | 0.87 | 0.92 |
| MNB | 0.92 | 0.96 | 0.86 | 0.89 |
| Random Forest | 0.89 | 0.9 | 0.86 | 0.87 |
| XGB | 0.91 | 0.92 | 0.87 | 0.88 |

Analyzing the results from Table 2, it is quite evident that increasing the feature space from 300 to 1,000 leads to much better results. Logistic Regression and Multinomial Naïve Bayes give a significant increase in both mean AUC and Hamming score. However, increasing the feature space does not affect the performance of the tree based classifiers. From this, we concluded that the data are linearly separable and the tree based models (non-linear classifiers) are unable to perform better than the two rudimentary techniques. Further experiments were performed by increasing the number of features to 2000 and 5000, and by including more Ngrams, but the mean AUC remained the same for the top 2 classifiers. The experiments can be visualized in Table 3 below:

Table 3. Extra Experimental Results for MNB

| Tuning Parameter | Values | Mean AUC |
|---|---|---|
| Ngram Rng | (1,3), (1,4), (1,5), (1,6) | ~0.96 for all |
| Max Feat | 2000, 5000 | ~0.96 for both |

In the experimentation of LSTM models, we used 10% of the training data as a validation set for each epoch of the training. To reduce the effects of overfitting, the number of training epochs was set to 2 and the best dropout layer configuration was set to 10% of the nodes. The deep learning model was executed on a CPU, which took around 40 minutes to complete. The comparison of results between the best machine learning algorithm and LSTM can be seen in the Table 4 below:

Table 4. Comparison between ML and DL

| Model | Mean AUC | Hamming Score |
|---|---|---|
| Log Reg | 0.96 | 0.92 |
| LSTM | 0.98 | 0.94 |
| Kaggle | 0.9885 | NA |

## 8 Discussion and Conclusion

After observing the performances of the machine learning algorithms, we can conclude that linear classifiers like Logistic Regression and Multinomial Naive Bayes perform slightly better as compared to non-linear state of the art classifiers like Random Forest and XGBoost. There are two

possible explanations to this. First, the data may be linearly separable and second, tree based algorithms do not work as well with highly dimensional sparse data sets in comparison to Naïve Bayes. Another point to note is, Naive Bayes classifier vastly outperforms all the other algorithms when it comes to time complexity. The classifier gave one of the best results and took hardly 6-7 seconds to train on all 6 labels. Furthermore, increasing the features using TF-IDF gave better results and it can be said that text classification tasks require a lot of data to train.

As expected, the LSTM model gave us the best results. This is most likely because LSTM gives a larger context window for the model to remember past words fed into the network between iterations. Incorporating our own word embeddings, enabled the classifier to capture the complex linguistic features of the text within our dataset. However, this could be a potential overfitting problem and may not be a viable solution for building a tool to be used across multiple online platforms. Some ideas for future work could be to effectively incorporate pretrained embeddings such as Glove or FastText into the LSTM model to make it more generalizable and reduce the train time.

The automated classifiers explored in this paper may be a valuable tool, not only for researchers, but also for platforms seeking to scale the task of detecting negative behavior online. We hope that our efforts motivate others to improve online conversations and make the internet a safer place.

## References


[1] E. Wulczyn, N. Thain, and L. Dixon. (2016). Ex Machina: Personal Attacks Seen at Scale. arXiv:1610.08914 [cs.CL].

[2] Toxic Comment Classification Challenge. https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

[3] S. Wang and C. D. Manning. (2012) Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. In ACL pages 90-94, 2012.

[4] M. Duggan. (2014) Online harassment. Pew Research Center.

[5] C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang. (2016). Abusive language detection in online user content. WWW.

[6] C. Shirky. (2011). The political power of social media: Technology, the public sphere, and political change. Foreign affairs, 28-41.

[7] I. Awan. (2014). Islamophobia and Twitter: A typology of online hate against Muslims on social media. *Policy & Internet*, 6(2), 133-150.

[8] M.W. Hughey, J. Daniels. (2013). Racist comments at online news sites: a methodological dilemma for discourse analysis. *Media, Culture & Society*, 35(3), 332-347.

[9] A. McCallum, K. Nigam, et al. (1998). A comparison of event models for naive bayes text classification. AAAI-98 workshop on learning for text categorization, vol. 752, pp. 41–48, Citeseer.

[10] T. Davidson et. al. (2017). Automated Hate Speech Detection and the Problem of Offensive Language. ICWSM.

[11] J. Roberts. (2017). Check Out Alphabet's New Tool to Weed Out the 'Toxic' Abuse of Online Comments. Fortune.

[12] M. Huang, C. Yujie, D. Chao. (2016). Modeling Rich Contexts for Sentiment Classification with LSTM. arXiv preprint arXiv:1605.01478.

[13] K. Fawagreh, M. M. Gaber, E. Elyan (2014) Random forests: from early developments to recent advancements, Systems Science & Control Engineering, 2:1, 602-609, DOI: 10.1080/21642583.2014.956265

[14] M. L. Zhang, Z.H. Zhou (2007). ML-KNN: A lazy learning approach to multi-label learning, Pattern Recognition

[15] J.H. Friedman (1999). Greedy Function Approximation: A Gradient Boosting Machine.