A Machine Learning Approach to Intervening on Toxic Comments in Online Forums

Natasha Mathur
Loren Hinkson
Andrea Koch

CAPP 30255
June 2019

Abstract

The goal of this investigation is to better identify comments that are considered toxic among statements made in response to news articles online, utilizing the CivilComments dataset of 1.8 million comments from the years 2015 to 2017. Our approach utilized a series of Naive Bayes, Support Vector Machine, Logistic Regression, Neural Net, and Long Short-Term Memory models using increasingly larger training datasets, and training datasets rebalanced to address the overall low number of toxic comments within the dataset. Models were evaluated based on the recall score associated with their predicted classifications, as our goal was to correctly classify as many truly toxic comments as possible. Results indicated that a three-layer, bidirectional LSTM model yielded the best balance of recall for both toxic and nontoxic comments while maintaining a high precision score for comments labeled "toxic", i.e., a large proportion of the comments predicted "toxic" had actually been rated "toxic" by CivilComments raters.

I. Introduction

Hate speech and toxic comments not only derail productive dialogues, but also lead to mental health and quality of life problems for those who are directly targeted. Marginalized groups in particular should not be left unprotected from perpetrators of hate speech. In recent years, social accountability pressures have forced tech industry giants to evaluate their roles in the continued and evermore complex problem of dealing with hate speech and toxic comments. They have struggled with drawing the line between hate speech and protected speech both ethically and within our constitutional framework. This line is forever shifting, reactive to changes in cultural norms, yet still guided by long standing political institutions. It is in the interest of the companies that host and create online structures, spaces, and forums in which hate speech is spread, to precisely identify hate speech and toxicity so it can be removed, and offenders can be reprimanded, or censored.

Data scientists and software engineers are particularly well-positioned to assist in solving this public policy issue. With large corpuses of variable and unstructured data, determining best approaches to identify hate speech and toxic language becomes an expansive and layered problem to solve. There is such a high level of interest to utilize machine learning to solve the identification quandary that Kaggle has published a multitude of challenges to enlist the computational analysis community in finding more useful resolutions. One such challenge is the Jigsaw Unintended Bias in Toxicity Classification competition which asks competitors to create a model that both minimizes unintended bias and correctly identifies toxic comments.

In this project, machine learning is applied to unstructured data to detect hate speech in comments from the CivilComments dataset (provided in the aforementioned Jigsaw challenge), with labeling informed by the Online Hate Index Research Project at D-Lab, University of California, Berkeley. The goal is to classify comments as hateful or not hateful. Historically, attempts to do similar classifications mis-identify comments that mention identify groups that could be attacked with hate speech as hateful. We hope to develop more nuanced models that correctly categorize both hateful speech and non-hateful identity references. In our models, this is reflected as 'toxic' for hate speech and 'nontoxic' for otherwise speech without hateful sentiment.

II. Related Work

Previous attempts have been made to classify hate speech, including in other kernels of the Kaggle competition referred to above. The ability to identify discourse-derailing comments is essential to many different online platforms, particularly social media, and has therefore attracted attention from a multitude of parties in recent years. Various methods have proven useful to classify hate speech, which differs based on the attributes and nature of the text. A review of the different methods used based on different types of text (i.e. youtube comments versus tweets) helped to inform the methods used in this paper.

One example of such work was the 2018 paper by Gaydhani, Doma, Kendre, and Bhagwat, "Detecting Hate Speech and Offensive Language on Twitter using Machine Learning: An N-gram and TF-IDF based Approach". Gaydhani et alia used machine

learning methods to detect hate speech and offensive language on Twitter. Experimenting with n-grams as features and utilizing TF-IDF, they were able to classify tweets into three classes: hateful, offensive and clean. The authors begin with a data set of classified tweets, and proceeded to clean the tweets and stem them using porter stemming. It was then split into a 70%-30% train-test set, and features such as TF-IDF weights and various n-grams were created. These were then fed into logistic regression, naive bayes, and SVM models.

Another 2018 study was conducted by Saha, Matthew, Goyal, and Kukherjee, "Hateminers: Detecting Hate speech against Women". Saha et alia focused on misogynistic tweets. They represented each tweet as a feature using Sentence Embeddings, TF-IDF Vectors, and bag of word (BOW) Vectors. The combination that worked best included Sentence Embeddings, a TF-IDF vector, and a BOW vector. These features were then use to build logistic regression and gradient boosting libraries, which were in turn evaluated on the accuracy obtained.

In "Modeling the Detection of Textual Cyberbullying," Dinakar, Reichart, and Lieberman focused their 2011 investigation on cyberbullying in YouTube comments. Dinakar et alia found that binary classifiers determining whether or not a given comment fit a specific label performed significantly better than multiclass classifiers involving a set of labels. Researchers scraped 50K YouTube comments (max 1,000 comments/ video) and built features such as TF-IDF weighted unigrams, Ortony lexicon of words denoting negative connotation, a list of profane words and frequently

occurring POS bigram tags. used for all comments; topic-specific bi-grams/uni-grams were added into the feature space for specific labels. They also built SVM and Naive Bayes model, and used the data from the validation set in increments of fifty instances to avoid overfitting.

The aforementioned related words confirmed that SVM and Naive Bayes models were useful for classifying online comments which differs in structural characteristics from other text classification tasks, such as those involving lengthier texts like speeches and papers/novels. Models for the above research papers were evaluated using accuracy.

III. Packages and Tools Used

This analysis was done in Python and documented in Jupyter notebooks. The data was loaded, manipulated, and handled in Pandas. Initial data exploration as well as feature generation was done in large part using the Natural Language Toolkit (NLTK), which team members had used before but not in a machine learning context. When it came to model generation the Naive Bayes and Support Vector Machines (SVM) models were made using functions from sklearn, a library which was first introduced to team members in Rayid Ghani's "Machine Learning for Public Policy" course. However, this project represented a significant advancement in the team members ability to build text classification models using sklearn. The library PyTorch, as studied for the first time during this course, was also used extensively to build our Long Short Term Memory

(LSTM) model. Python packages TorchText and Stanford's GloVe: Global Word Vectors were utilized to enhance the performance and scalability of LSTM models.

Due to the large size of the training data this modelling required the use of additional resources. Amazon Web Services (AWS) tools, specifically SageMaker and S3 buckets, were used to host data, and decrease the amount of time required to process data and run each model. SageMaker uses AWS's Elastic Compute Cloud (EC2) to scale work and launch an instance of a Jupyter notebook. Instance size was determined based on a balance between cost and efficiency. These tools were adequate for building SVM and Naive Bayes models, and it was determined that an idea of model performance could be ascertained using a reasonable subset of the data.

For Long Short-Term Memory (LSTM) modelling, cost and efficiency limitations of the SageMaker tool led the team to also use the GPUs offered by the University of Chicago Research Computing Center (RCC) and Google CoLaboratory ("Google CoLab"). Code was be rewritten in order to fit the requirements of some of these resources. For example, our LSTM model had to be restructured to allow for batch processing and bash scripts had to be written to send jobs to RCC. Team members had previously used the RCC, but learned to use Sagemaker and Google CoLab specifically for this project.

IV. Data Exploration

There are approximately 1.8 million comments in the train data set provided from the Jigsaw Unintended Bias in Toxicity Classification challenge. These comments are sourced from CivilComments, which collected comments from 50 different English language news sites from 2015 to 2017. Comments were given a toxicity score as well as scores for other characteristics, such as type of toxicity or identity.[1]

An exploration of the data revealed two main findings which guided decisions made during modeling: (1) the majority of comments are not highly toxic (e.g. 'severe_toxicity', 'obscene', 'threat'); (2) nor does a majority of the data have identity markers (e.g. 'atheist', 'buddhist', 'muslim'). Toxicity levels for each identity group had relatively the same distribution, therefore it was necessary to create a new target variable which categorized comments at either a toxic or nontoxic based on a threshold of 0.5 for the raw data set's 'target' score. This further revealed that about 6% of the data consisted of toxic comments. This came up repeatedly both as an issue in training a model to recognize the toxic comments, but also when determining which metric was the best way to evaluate such an imbalanced data set.

[1] 'id', 'target', 'comment_text', 'severe_toxicity', 'obscene', 'identity_attack', 'insult', 'threat', 'asian', 'atheist', 'bisexual', 'black', 'buddhist', 'christian', 'female', 'heterosexual', 'hindu', 'homosexual_gay_or_lesbian', 'intellectual_or_learning_disability', 'jewish', 'latino', 'male', 'muslim', 'other_disability', 'other_gender', 'other_race_or_ethnicity', 'other_religion', 'other_sexual_orientation', 'physical_disability', 'psychiatric_or_mental_illness', 'transgender', 'white', 'created_date', 'publication_id', 'parent_id', 'article_id', 'rating', 'funny', 'wow', 'sad', 'likes', 'disagree', 'sexual_explicit', 'identity_annotator_count', 'toxicity_annotator_count'

V. Model Building

A. Feature Generation

Features were generated in order to enable the creation of bags of words (BOW) and TF-IDF vectors. The first step taken was to ensure all comment text was lowercase, remove punctuation marks, and remove filler words like "the" and "a" (stop words). Although traditionally stop words include phrases like "won't" those were left in as they are indicative of negative sentiment, and the typical NLTK collection of stop words is geared more towards topic modeling than sentiment analysis. The comments were also stemmed to reduce differences between the forms of a single word using both Porter and the more aggressive Lancaster methods. A copy of each comment was retained as a string, a list, and a collection of bigrams to accommodate each of the model types that was used. Finally, features were created to capture other attributes of the comment, such as the count of exclamation points or the count of uppercase letters which are generally associated with stronger sentiments. The features that were generated are listed by column name:

Name	Type	Description	Lower Case	Remove Stop Words	Remove Punctuation	Stemmed	List
Text Features							
'split'	list	Comment string split into a list of words					✓
'cleaned_w_stopwords_str'	string	Comment with punctuation removed			✓		
'cleaned_w_stopwords'	list	Comment with punctuation removed,			✓		✓

		split into list of words					
‘cleaned_no_stem_str’	string	Comment with stopwords and punctuation removed, lowercased		✓	✓		
‘cleaned_no_stem’	list	Comment with stopwords and punctuation removed, lowercased, and split into list of words		✓	✓		✓
‘cleaned_porter_str’	string	Comment with stopwords and punctuation removed, lowercased, and Porter stemmed	✓	✓	✓	Porter	
‘cleaned_porter’	list	Comment with stopwords and punctuation removed, lowercased, Porter stemmed, and split into list of words	✓	✓	✓	Porter	✓
Name	**Type**	**Description**	**Lower Case**	**Remove Stop Words**	**Remove Punctuation**	**Stemmed**	**List**
Text Features (cont’d)							
‘cleaned_lancaster_str’	string	Comment with stopwords and punctuation removed, lowercased, and Lancaster stemmed	✓	✓	✓	Lancaster	
‘cleaned_lancaster’	list	Comment with stopwords and punctuation removed, lowercased, Lancaster stemmed, and split into list of words	✓	✓	✓	Lancaster	✓
‘bigrams_unstemmed’	list	Comment with stopwords and punctuation removed, lowercased, then converted into bigrams	✓	✓	✓		
Numerical Features							
‘perc_upper’	float	Percent of uppercase letters in comment					
‘num_exclam’	integer	Count of exclamation points in comment					
‘num_words’	integer	Count of words in comment					

In addition to the text features numerical features such as the ones included in the table were initially created. However further data exploration showed that the features did not vary significantly across each class, and therefore would not be informative.

B. Metrics and Data Set Imbalance-Handling

This investigation involved building several different types of models, and therefore it was imperative to select certain key metrics to evaluate performance across models. The initial measure used was accuracy. However, pure accuracy score provides an incomplete picture. Accuracy does not give any indication of the proportion of the comments are correctly being assigned to each class: toxic or nontoxic. Further, in our sample, the vast majority (94%) of the training set consisted of nontoxic comments. A model could therefore return a high accuracy by simply labeling all comments as non-toxic. For these reasons, we had concerns with using accuracy as a measure of performance, as briefly discussed in the literature review above.

As the goal of our analysis was to better identify toxic comments, we determined that it made most sense to separately evaluate how the model was performing on toxic comments versus nontoxic comments. Further, model performance was assessed on other subgroups of interest, such as comments that represented a threat, insult, or obscene language.

Precision, and therefore the F1 score (which is calculated using precision and recall), are unsuitable metrics when looking at a data frame of solely toxic or nontoxic comments. All of the comments in each of those subgroups would be 1 or 0, respectively. As a result, the main metric we determined suitable for evaluation was recall; specifically, recall for the toxic comments in the dataset. Selecting a model based on recall supports our objective to correctly identify as many of the toxic comments as possible.

As mentioned above, the training data was comprised of 94% nontoxic and 6% toxic comments. In initial stages it was determined that such an imbalanced data set led to poor modeling because of the low proportion of toxic comments from which a model could learn. To mitigate this issue, training data was resized and rebalanced to include different portions of toxic and nontoxic comments (e.g. 25% toxic - 75% nontoxic; 50% toxic - 50% nontoxic; and 33.33% toxic - 66.67% non-toxic). A "hold-out" set was randomly sampled from the training data prior to reshaping and rebalancing to maintain an untouched set of data reflective of the actual proportions to which the model should be generalizable on unseen data. The extent of the imbalance within the data also reinforced our decision to use recall to evaluate our models with a focus on correctly classifying as many toxic comments as possible to support intervention on hate-mongering.

C. Modeling and Methodology

Based in large part on review of related literature and past researcher's conclusions and learnings, the first modeling step was to build Naive Bayes, Support Vector Machines, and Logistic Regression models. Initial testing was conducted on a small sample of the imbalanced data set. These initial models showed adequate accuracy scores in the high 80's to mid 90's percent ta ge. Upon further inspection, however, it became apparent that the models were merely assigning a '0' -- a nontoxic classification -- to nearly every comment.

This motivated two courses of action. The first idea was to build a recurrent neural network model, more specifically a Long Short-Term Memory model (LSTM), which we determined could be nimble enough to latch onto key differences between classes during text classification. Second, the training set was reweighted in favor of the toxic comments as discussed above. It is difficult for models to glean adequate information to make correct classifications with so few toxic comments, resulting in a tendency for models to assign them a lower relative weight compared to the numerous nontoxic comments.

Based on their performance and how well-suited they were for the complexities of text classification, Naive Bayes and SVM were selected from the initial models for further development. The neural net and LSTM models were built in addition for further comparison of model performance. The reweighted data sets were used for training across all models.

1. Naive Bayes

The first type of model that was developed was a series of Naive Bayes models. This type of model was selected for its general prowess at language learning tasks. The training data was used to create a binary classification, where a comment was considered "toxic" if it received a target rating from comment reviewers greater than 50%, and "non-toxic" if not.

While Naive Bayes models generally perform well for the data for this task, its' probabilistic nature is a downside. When making a classification, Bayesian statistics takes into account the overall likelihood that any given observation would be in each class. In this case, the original training data was comprised of only 6% of "toxic" comments. This meant that the model tended to allocate all comments to the vastly more dominant class: nontoxic.

Therefore, rebalanced training data was used. The toxic comments in the original training data were repeated as needed to make data sets that were 25%, 50%, and 75% toxic. In addition to giving the models more examples of toxic comments, this would also reduce the models' tendency to classify all comments as nontoxic.

This also yielded the need to determine which the most useful metric to use for model evaluation. The focus in this project was to correctly identify toxic comments, and furthermore, to correctly identify toxic comments regardless of references to someone's identify. The function *get_metrics* (stored in *model_functions.py*) was written to extract and output multiple metrics.

After the test/ held out data set had been fed through the model and predictions were obtained for each test comment, the results were further subsetted into a number of categories. These included whether the comment was toxic ("Target") or not ("Nontarget"), whether it was about an identity, and whether it was an obscenity, threat, or insult. The accuracy, precision, recall, and F1 score of each of these subsets was calculated, as well as for the model overall.

2. SVM

The next set of models built were support vector machine (SVM) models. These were chosen for their aptitude for language learning tasks, as supported by our literature review. Additionally, SVM models are not probabilistic. This is an advantage because it could help reduce the models' tendency to default to the majority class, "non-toxic." Further, this could help eliminate problems which arise from having differing composition of the training and testing sets (i.e. rebalancing the training set).

Again, the training data was rebalanced to contain 25%, 50%, and 75% toxic comments. The models were evaluated by recall and with the same subcategories used for Naive Bayes.

3. Neural Nets

A preliminary Neural Net model was built to investigate how a two-layer model would perform with our data. As with previous models, a sample data set was initially used for training. This allowed us to both ensure that the data size remained manageable, and to adjust the proportions of each class. For this model, comments were classified as "super toxic" if they had a target score of greater than 80% ($target \geq 80\%$), "toxic" if it was greater than and equal to 50% and up to 80% ($50\% \leq target < 80\%$), and "not toxic" otherwise ($target < 50\%$).

Each comment was tokenized using NLTK's *word_tokenizer*. This process was slightly different from the above-described processing text used for the majority of other models (Naive Bayes, Logistic Regression, and SVM). This was used to build a list of comments and their classes, as well as build a bag of words. The list of words was then cleaned by removing stopwords, applying Porter stemming, and converting to lowercase.

A two-layer Neural Net was built and training data was passed through. . Model parameters were repeatedly tuned until the gradient converged. This very basic neural net was built and evaluated on accuracy. Due to low accuracy, it was determined that a more complex neural net was needed.

4. LSTM

To this end, we chose to apply a series of Long Short-Term Memory (LSTM) models to the hateful comments problem. Recurrent neural networks have historically

performed well for text classification tasks similar to our efforts to classify hate speech. Based on research into various types of text classification and textual analysis with this type of network, we chose to build a three-layer LSTM model including an embeddings layer, a LSTM layer, and a final linear layer. Softmax nonlinearity was applied to output from the final layer to yield a "prediction" of toxicity for each input sequence during training iterations, and while we tested both Cross Entropy and Negative Log-Linear Loss, we determined that Negative Log Linear Loss was most appropriate, and final models' loss was calculated using this measure. Dropout was also experimented with in early iterations of these models, however, we determined that little to no dropout performed best on our validation and test datasets.

As the provider of the dataset identified at time of its publishing, there is a lot of "noise" when making a determination on whether or not a comment is toxic. For example, models have a tendency to learn that the the mere mention of particular identity groups indicates toxicity due to correlation between a higher "identity" rating and a toxic label. In an attempt to mitigate some of the risks associated with that noisiness and the relative sparseness of text data, we used an Adam optimizer that would apply and adapt moving averages of per-parameter learning rates, rather than using a single learning during training. In combination with the rebalanced datasets used for training, we hoped this would allow our models to pick up on more subtle differences between comments.

An initial LSTM-based classifier was trained on a "toy" dataset of ten thousand comments for experimentation and benchmarking. At this stage, we created our own

word embeddings based on a limited corpus of 7,500 words, and allowed the model to learn from them organically, using a batch size of one. We found that these baseline models with only a few hidden layers and one epoch did not yield high performance in terms of F1 score, chosen as an early metric as it gives general measure encompassing elements of both precision and recall, for either our Target class (toxic label) or our Nontarget class (not-toxic label). For comments with a strong identity score, and comments rated highly as threats or insults, this initial classifier was almost always incorrect. Increasing the number of epochs to 20, however, allowed the model to gain a modicum of precision.

We found very quickly that while such an imbalanced dataset could achieve a high accuracy score quite easily, it was highly likely to mislabel samples in the smaller "toxic" label group. Similar to our experimentation for Naive Bayes, logistic regression, and SVM classifiers, we decided to train the LSTM model on a more balanced data set in order to learn from a larger volume of toxic labels. We began with the original train and test sets toxic ratio of 6%. Based on iterations conducts with previous models, we opted not to train on a 25% toxic dataset for the LSTM. However, we were interested in whether LSTM models trained on datasets with a rate of toxicity between 25% and 50% toxic, or between 50% and 75% toxic, would perform very differently than a dataset with 50% "toxic" labels, which was ideal for our Naive Bayes models. To that end, we decided to train LSTM models on datasets sampled with replacement to achieve ratios of 35%, 50%, 60%, 65% and 75% toxic labels.

As RNNs may perform better when exposed to larger quantities of data during training, we set a goal to evaluate LSTM models on 20%, 40%, 60%, and 80% of the available 1.8 million comments. To this end, we began transforming our LSTM model into an executable that could be processed on a university-hosted high performance computing cluster (RCC) with available GPUs. Early attempts to scale our toy LSTM model indicated that several changes would be required, including:

- Utilizing a single binary classifier, rather than two separate classes with binary values
- Restructuring our model class for more efficient batching to make training on larger datasets more feasible
- Further adjustment to model dimensions to accommodate the above changes

Challenges with uninterrupted access for enhancement, testing, and debugging guided us to our final modeling environment, the Google CoLaboratory, which allowed us to complete our implementation of GPU-based PyTorch.cuda() compatibility for models trained on increasingly large training data sets.

As part of this transition, we implemented functionality to transform our data into batched word embedding vectors using the Python TorchText package, rather than relying on manual batching and vector transformation. We researched several workarounds for the package's typical requirement for limited data in an external document, for example a CSV or TSV, and were eventually able to bypass any external writing steps, implementing Dataset classes that enabled us to pull text data directly

from our existing rebalanced training dataframes and corresponding imbalanced validation dataframes. Our word embeddings were also augmented by the Stanford GloVe: Global Vectors for Word Representation produced by Pennington, Socher, and Manning in 2015, which improved performance across several metrics. We believe this is due to the focus of GloVe vector weights on the co-occurence of words from which the model infers meaning.

We trained a total of 25 LSTM models on each combination of data proportions mentioned above (20, 40, 60, and 80%) and toxicity rate (35%, 50%, 60%, 65%, 75%) as well as a randomly sampled training set representative of the original proportions in the data (6%), used as a control group for comparison. We found a high propensity for over-fitting after as little as two epochs, especially as training datasets grew larger. As expected, models trained on more balanced datasets tended to perform better in terms of our key metric, recall. In selecting our "best" overall LSTM model, we prioritized selecting the model that had the best recall for our Target group ("toxic" labeled comments) and performed least poorly for on Nontarget group ("nontoxic" labeled comments) recall, with a secondary emphasis on precision of "toxic" predictions. Contrary to our hypothesis that more data would yield the best recall, we found that using only 20% of the available training data (20% was reserved for testing and validation), that is, 16% of all comments, resampled for 75% toxicity, yielded the highest balance of recall for both groups.

VI. Results

Our model is extremely sensitive to the data used in training.

A. Evaluation Metrics

Evaluation metrics were informative in multiple ways. Models that obtained a high overall accuracy could in fact be performing poorly where precision or recall was low. This cemented our decision to focus on recall as an evaluation metric. In addition to comments being evaluated for the target variable (whether or not they were toxic), they were also evaluated for recall given a threshold score (> 50%) for obscenity, insult, or threat.

Several evaluation metrics were used to determine the "best" hyper-tuned models:

All Models:

1) Overall Accuracy: Accuracy over all samples
2) Overall Recall: Recall over all samples
3) Target Recall: Recall for samples where target ("toxic") > 0.5 only
4) Non-Target Recall: Recall for samples where target ("nontoxic") $\leq$0.5 only

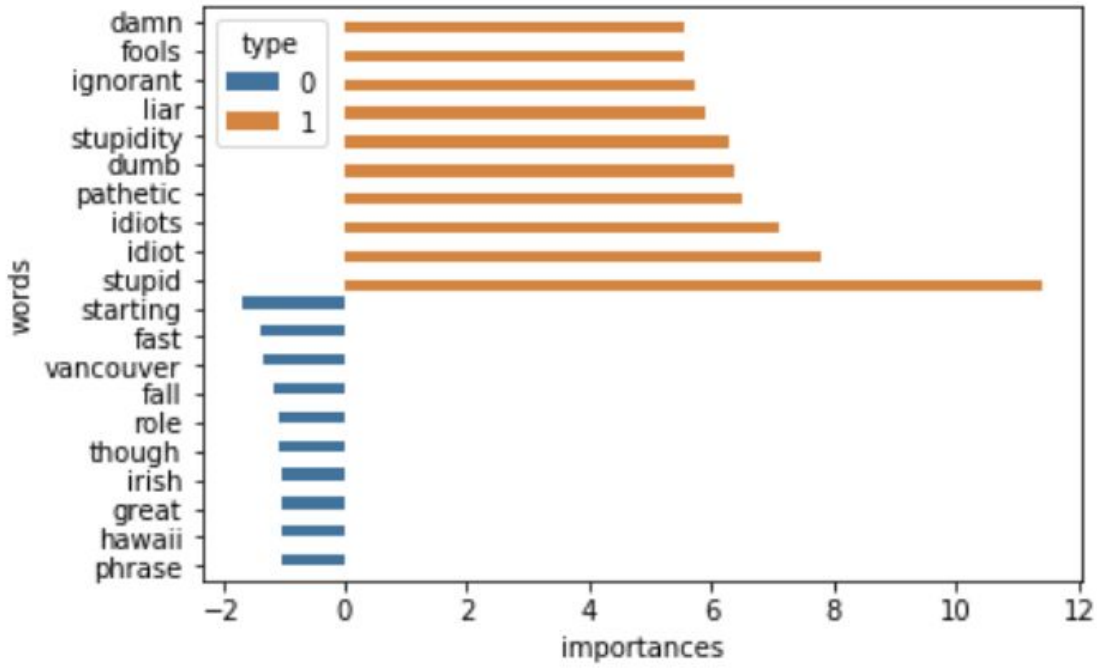
Non-Neural Net Models:

5) Strong Identity Recall: Recall for samples where identity_attack > 0.5
6) Obscenity Recall: Recall for samples where obscenity > 0.5
7) Insults Recall: Recall for samples where insult > 0.5
8) Threats Recall: Recall for samples where threat > 0.5

	Naive Bayes	SVM	LSTM
Training Dataset			
Parameter tuning	~257K observations (50% toxic, 50% nontoxic comments)	~21K observations (25% toxic, 75% nontoxic comments)	~576K observations (60% toxic, 40% nontoxic comments) after a single epoch
Model Selected			
Overall accuracy	75.951%	95.301%	88.164%
Overall Recall	83.750%	59.100%	88.140%
Target Recall	83.750%	59.100%	88.188%
Non-target Recall	75.465%	97.217%	88.000%
Strong Identity Recall	87.616%	47.945%	N/A
Obscenity Recall	83.213%	55.172%	N/A
Insults Recall	85.480%	68.073%	N/A
Threats Recall	85.472%	22.222%	N/A
Target Precision	100%	100%	99.483%

A. Feature Importance

Feature importance was gathered from the SVM final model. The top 10 and bottom 10 words for identifying toxic comments were as to be expected, with the top 10 words having strong negative sentiments and the bottom 10 having a neutral sentiment:



VII. Lessons Learned & Future Work

Steps taken to tune and enhance performance across various metrics for fundamentally different models allowed the authors to gain a firsthand understanding of the complexities inherent in text processing, and to grasp the practical realities of abstract concepts such as variability, and ambiguity in text data. Familiarity with several computing platforms and text processing packages were also valuable takeaways.

Learnings include gaining experience with the requirements and tendencies of four types of models, including recognizing modeling techniques that may lead to overfitting, for example, using too little training data, or in the case of the LSTM model, too much data. Issues rooted in our sampling methods and training data set selection led to excessive specificity in some of our SVM models, especially in combination with

imbalanced data. The authors gained valuable experience in identifying potentially useful preprocessing steps, as well as steps that were resulting in decreased effectiveness in our models, such as applying dropout on a single layer LSTM, or and not properly managing the sensitivity of a Naive Bayes model to the composition of training vs. test sets.

There were also valuable learnings in terms of the most appropriate computing resources for different part of the process. At earlier stages, we found that Amazon Web Services SageMaker/ EC2 was suitable for models that required less computing power. For models that required GPUs, it was necessary to leverage Google CoLaboratory. We believe that in the future, it may be prudent to utilize Google CoLab's available GPUs and RAM for data preprocessing as well, which would have enabled researchers to perform this step in fewer stages.

To extend this undertaking and continue to augment our ability to properly classify toxic comments, next steps would include running the Naive Bayes and LSTM model results through a second model, using these predictions as a feature. Currently, our best models correctly identify the majority of hate speech comments rated as "toxic" by CivilComments raters, however, they incorrectly label some comments that were not rated "toxic." A subsequent model could be used to identify hate speech comments from the wider pool of potentially hateful comments identified by the models developed in this exploration geared towards achieving the highest possible recall for toxic comments. We believe that the combination of focus on identifying as many truly toxic

comments as possible, and then "narrowing" vai a model focused on correctly labelling non-toxic comments, could potentially reduce misclassification due to subtleties such as references to a targeted identity group.

Throughout our modeling process and several iterations of each type of models models, we determined that the following steps would be useful in continuing such work in the future:

- **Data Variation:** As any analysis of language is subject to both sparsity and ambiguity of any language, a truly viable model must be able to make informed decisions about a wide variety of comments. For further generalizability, models must be trained on a wider array of both toxic and nontoxic comments that represent the full lexicon of online discourse, and patterns in toxic comments. As this dataset was limited to comments from 50 news-oriented forums, toxic verbiage, the rate of occurrence and types of "slang" utilized, the academic reading and writing level of comments, and several other such characteristics may impact the model's ability to perform as well in identifying toxicity in comments made on different platforms, for example, on social media sites such as Twitter, YouTube and Reddit.

  Additionally, the extensive imbalance between toxic and nontoxic comments in the CivilComments dataset did not allow for full realization of value from the volume of a data available. Though we were able to

"rebalance" the data by sampling with replacement, we believe that future attempts at such a classification task should include efforts to obtain a larger collection of *unique* toxic comments to enable more textured learning by models. Ideally, this would provide more robust and balanced data on which to train and test models that would yield stronger, more consistent performance.

- **Continuous Learning:** As hate speech is very tied to current events, even within the two years between the end of the data collection period and this investigation, topics and vocabulary that indicate hate speech have changed drastically. Future work should investigate the feasibility of implementing mechanisms for continuous learning to keep models from becoming obsolete.

VIII. Conclusions

Ultimately, one of the Long Short Term Memory models trained on rebalanced data performed well when both classification groups were considered on recall, as did the selected Naive Bayes model. The LSTM model also had slightly higher precision, however, making it a better candidate for future development. This model largely meets our stated criteria in terms of our goal to identify as many toxic comments as possible, and simultaneously maintains a high degree of precision for the comments labeled

toxic, that is, the proportion of truly toxic comments in relation to all comments the model predicted were "toxic."

All models described in relation to this study are first steps towards creating a robust machine learning system that can independently identify toxic comments from a variety of sources. Based on our process, we believe that layered modeling could be the key to flexible hate speech identification that allows for meaningful online conversation and idea-sharing. This is supported by the fact that models that performed well for target comments specifically did not perform well for non-target comments. A compound model strategy would allow for one model, such as the Naive Bayes or LSTM models we developed with a focus on including all toxic comments, to label potentially toxic comment from which a second model that prioritizes properly excluding nontoxic comments could extract those comments that were in fact toxic. We hope that continued research in this field leverages our learnings and results, made public on our shared Github repository, to build models that improve the way we all communicate online.

References

PyTorch Documentation and Discussion Boards:

- https://pytorch.org/docs/stable/_modules/torch/nn/modules/loss.html
- https://pytorch.org/docs/stable/cuda.html#module-torch.cuda
- https://pytorch.org/docs/stable/notes/cuda.html
- https://pytorch.org/docs/stable/nn.html#torch.nn.utils.rnn.pack_padded_sequence
- https://pytorch.org/tutorials/beginner/saving_loading_models.html
- https://discuss.pytorch.org/t/how-to-correctly-give-inputs-to-embedding-lstm-and-linear-layers/15398
- https://discuss.pytorch.org/t/understanding-lstm-input/31110
- https://discuss.pytorch.org/t/multiclass-classification-using-lstm/40944
- https://discuss.pytorch.org/t/runtimeerror-multi-target-not-supported-at-users-soumith-miniconda2-conda-bld-pytorch-1532623076075-work-aten-src-thnn-generic-classnllcriterion-c-21/31071
- https://discuss.pytorch.org/t/typeerror-can-t-convert-cuda-tensor-to-numpy-use-tensor-cpu-to-copy-the-tensor-to-host-memory-first/32850/6
- https://discuss.pytorch.org/t/argmax-with-pytorch/1528/9
- https://discuss.pytorch.org/t/how-can-we-release-gpu-memory-cache/14530
- https://github.com/pytorch/pytorch/issues/13214

Torchtext Documentation and Related Posts:

- https://torchtext.readthedocs.io/en/latest/data.html

GloVe Documentation:

- https://nlp.stanford.edu/projects/glove/

Scikit-Learn Documentation:

- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- https://scikit-learn.org/stable/modules/naive_bayes.html
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Google Cloud Storage File System Documentation

- https://gcsfs.readthedocs.io/en/latest/

Aditya Gaydhani, Vikrant Doma, Shrikant Kendre and Laxmi Bhagwat. "Detecting Hate Speech and Offensive Language on Twitter using Machine Learning: An N-gram and TFIDF based Approach". September 2018. Retrieved from: https://arxiv.org/abs/1809.08651

Anton Melnikov, "Make torchtext training examples from pandas.DataFrame on the fly," GitHub Gist, 18 Mar. 2018, gist.github.com/notnami/9d3cb9284d8d5667857baeb46b4880fa

"A PyTorch Tutorial - Deep Learning in Python." *Adventures in Machine Learning*, 26 Oct. 2017, adventuresinmachinelearning.com/pytorch-tutorial-deep-learning/.

Bakharia, Aneesha. "Visualising Top Features in Linear SVM with Scikit Learn and Matplotlib." *Medium*, Medium, 1 Feb. 2016, medium.com/@aneesha/visualising-top-features-in-linear-svm-with-scikit-learn-and-matplotlib-3454ab18a14d.

bentrevett. "Pytorch Sentiment Analysis." *GitHub Repository*, 10 Apr. 2019, github.com/bentrevett/pytorch-sentiment-analysis/blob/master/2%20-%20Upgraded%20Sentiment%20Analysis.ipynb.

Brownlee, Jason. "A Gentle Introduction to Long Short-Term Memory Networks by the Experts." *Machine Learning Mastery*, Machine Learning Mastery, 19 July 2017, machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/.

Brownlee, Jason. "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning." *Machine Learning Mastery*, 26 May 2019, machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

Brownlee, Jason. "How to Prepare Text Data for Machine Learning with Scikit-Learn." *Machine Learning Mastery*, 21 Nov. 2017, machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/.

Bushaev, Vitaly. "Adam — latest trends in deep learning optimization." *Towards Data Science*, Towards Data Science. October 2018. Retrieved from: towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

Deeplearningathome. “Pytorch Language Model.” GitHub Repository, 11 June 2017, github.com/deeplearningathome/pytorch-language-model/blob/master/lm.py.

Dinakar, Karthik, et al. “Modeling the Detection of Textual Cyberbullying .” *Association for the Advancement of Artificial Intelligence*, Association for the Advancement of Artificial Intelligence, 6 July 2011, www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/view/3841/4384.

Fangwater. “Medical-Named-Entity-Recognition-for-ccks2017.” *GitHub*, 21 July 2017, github.com/fangwater/Medical-named-entity-recognition-for-ccks2017/blob/master/LSTM-CRF%20NER/LstmModel.py.

geoffn91. “Dataframe as Datasource in Torchtext.” *Stack Overflow*, Stack Overflow, 4 Oct. 2018, stackoverflow.com/questions/52602071/dataframe-as-datasource-in-torchtext.

_gk. “Text Classification Using Algorithms.” *Chatbots Life*, Chatbots Life, 11 Jan. 2017, chatbotslife.com/text-classification-using-algorithms-e4d50dcba45.

gk_. “Text Classification Using Neural Networks.” *Machine Learnings*, Machine Learnings, 26 Jan. 2017, machinelearnings.co/text-classification-using-neural-networks-f5cd7b8765c6.

keitakurita. “Language Modeling Tutorial in Torchtext (Practical Torchtext Part 2).” *Machine Learning Explained*, 2 Mar. 2018, mlexplained.com/2018/02/15/language-modeling-tutorial-in-torchtext-practical-torchtext-part-2/.

keitakurita. “A Comprehensive Introduction to Torchtext (Practical Torchtext part 1).” *Machine Learning Explained*, Machine Learning Explained, 8 Feb. 2018, mlexplained.com/2018/02/08/a-comprehensive-tutorial-to-torchtext/

Koehrsen, Will. “Beyond Accuracy: Precision and Recall.” *Towards Data Science*, Towards Data Science, 3 Mar. 2018, towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c.

Lynn-Evans, Samuel. “How to Use TorchText for Neural Machine Translation, plus Hack to Make It 5x Faster.” Towards Data Science, Towards Data Science, 27 Sept. 2018,

towardsdatascience.com/how-to-use-torchtext-for-neural-machine-translation-plus-hack-to-make-it-5x-faster-77f3884d95.

Mukherjee, Animesh, et al. “Hateminers : Detecting Hate Speech against Women.” ArXiv.org, 17 Dec. 2018, arxiv.org/abs/1812.06700.

“Precision and Recall.” *Wikipedia*, Wikimedia Foundation, 3 June 2019, en.wikipedia.org/wiki/Precision_and_recall.

Ren, Yitong. “Get Started with Using CNN+LSTM for Forecasting.” *Towards Data Science*, Towards Data Science, 11 Mar. 2019, towardsdatascience.com/get-started-with-using-cnn-lstm-for-forecasting-6f0f4dde5826.

Stamenković, Florijan. “RNN Language Modelling with PyTorch - Packed Batching and Tied Weights.” Medium, Medium, 3 June 2018, medium.com/@florijan.stamenkovic_99541/rnn-language-modelling-with-pytorch-packed-batching-and-tied-weights-9d8952db35a9.

Trevett, Ben. “Bentrevett/Pytorch-Sentiment-Analysis.” *GitHub*, 10 Apr. 2019, github.com/bentrevett/pytorch-sentiment-analysis/blob/master/2%20-%20Upgraded%20Sentiment%20Analysis.ipynb.