DISCo: Detecting Insults in Social Commentary

Jayadev Bhaskaran (jayadev)
Amita Kamath (kamatha)
Suvadip Paul (suvadip)

***Abstract*—Detecting online abuse, insulting behavior or cyber-bullying has been a trending topic across all social platforms for many years. Automated systems to detect "trolls" and people who insult have failed to be completely foolproof. This is either because the basis upon which a statement is considered "insulting" or not usually depends on context, which might not be evident to a machine parsing the statement, or because traditional machine learning models fail to capture the sequential complexity of the comment. In this paper we try to solve 3 major problems: 1) How do we classify a comment as abusive and containing a personal attack, 2) Can we use newer and more stronger computational methods to classify comments using latent sequential information, 3) Does contextual information about the user help in classifying a personal attack.**

I. Introduction

Most research (including the state-of-the-art) that work to classify comments as insulting or abusive often look at comments as a whole and without user context [1,2,3]. The other major drawback from recent work is that they either use self curated data (not released to the public) or use older open data sources. In this project, we aim to better the traditional models by 1) Using newer deep learning models 2) Using the newer, well-curated and more robust Wiki-Detox dataset 3) Not just looking raw comments to gain insights, but also incorporating other contextual user characteristics (i.e. anonymity). We also determine that models can benefit from the contextual data provided by GloVe/word2vec embeddings, in comparison with n-grams.

We first establish a baseline for detection of insults in comments, then use increasingly complex algorithms with better hyper-parameters to determine how the performance changes. We finally evaluate how newer deep learning models work compared to the older traditional models which we have set as our baselines.

A. Problem Statement

Given a number of comments c_i and a set of users $u_i \in U$ who make these comments, we wish to classify each comment as being abusive or not. We aim to determine a ML algorithm that is able to classify whether a comment c_i qualifies as a personal attack, with some probability P.

B. Input-Output Specifications

The input to the algorithm is a comment and the output is a probability P which classifies a comment is abusive or not. If $P > 0.5$, it is classified as abusive.

1) Input for Baselines: Each comment c_i can be represented as a feature vector of the number of words. This is drawn from the bag of words model, where each kth entry of comment i, c_{ik}, represents the count of the word $v_i \in V$, where V is the vocabulary. We also incorporate user data using a one-hot vector of user ID scaled by some constant α by concatenating it to the comments.

2) Input for Deep Learning Models: Each comment c_i can be represented as a sequence of words after tokenization. The maximum length of each comment is restricted to 100 words for easier computation. Each word can be represented as an embedding from either word2vec or GloVe. Thus each comment c_i is a sequence of word embeddings of a particular word. This is given as input to the deep learning models.

For our baselines we use Logistic Regression and Multi-Layer Perceptron (MLP). For the Deep learning models we use MLP, Convolutional Neural Networks, LSTMs and Convolutional LSTMs.

II. Related Work

A. Detection of Offensive Language in Social Media

The first notable paper to begin abuse detection was in Yin et al, 2009 [13] wherein SVMs were used with a combination of TF-IDF to classify abuse. We have set up the same experiment as one of our baselines. This is over the bag of words model where we look at word count per comment.

Davidson et al. [19] discuss the problem of offensive language in various forms of social media, as well as the challenge in identifying hate speech in comparison with general offensive language. The dataset contained about 30,000 tweets annotated by three or more human workers each. They train a classifier for the same, and describe that using logistic regression and linear SVM did significantly better than other simple baselines. The model was biased towards classifying tweets as less hateful or offensive than the workers believed them to be.

B. Deep Learning for NLP

More recently, deep learning approaches have shown much promise when applied to traditional NLP problems of understanding sequential information. RNNs have been used to model sequence in textual data by creating long range dependencies [14, 15]. Kawthekar et al. [15] has shown that LSTMs and 1-Dimensional CNNs for understanding text are now the standard models for learning dependencies in

sequences which are not captured by traditional bag of words models. Wang, et al. [16] use LSTMs to predict tweet polarity. They show that a problem modeled this way improves on the current state-of-the-art algorithms. Although CNNs are used for image data, they have been shown to work on words as well. Similar to images [17], words closer to other words have more meaning in their context and this can be captured using the convolutional frameworks used in CNNs.

Wulczyn et al. [3] discuss the Wiki-Detox dataset that we have chosen as the focus of this project. They present a classifier that can yield a result approximately as good as the aggregate of 3 human workers, as measured by the area under the ROC curve and Spearman correlation. They considered three dimensions in terms of model design: the model architecture (Logistic Regression vs Multi Layer Perceptron), n-gram type (word vs char), and label type (one-hot vs empirical distribution). Using their classifier, they then answer questions about detecting abuse. We observed that this paper was a good starting point that allowed us to compare our approaches with existing methods.

III. Dataset and Features

We used the Wikipedia Detox dataset released in 2017 [11]. This dataset contains a corpus of all 95 million user and article talk diffs made on the Wikipedia website from 2001-2015. It also includes an annotated dataset of 100,000 comments on Wikipedia articles.

For our problem, we used the "personal attacks" corpus which focused specifically on abuse targeted at the recipient of the comment. While incorporating user data, we focused on a specific temporal subset of the personal attack dataset, joined with the overall comments corpus for that subset (using the rev_id field). This allowed us to use user-specific data while performing our analyses.

As mentioned above, each comment is reviewed by at least 10 "workers" $w_i \in W$, who independently judge if the comment could be considered a "personal attack", irrespective of severity in order to account for worker bias. A given comment is classified as a personal attack if more than 50% of the workers reviewing that comment deem it to be so. Note: Comments may also be made by anonymous users, and we choose to group all anonymous users together under one user ID, $u_i = 0$.

We divided the dataset into training data (60%), validation data (20%) and testing data (20%) with random shuffling. We experimented with representation using n-grams of different sizes as well as GloVe/word2vec embeddings.

IV. Methods: Traditional Models

We ran three traditional models on our data: Logistic Regression (our baseline), Multinomial Naive Bayes, and SVM. Logistic Regression was chosen as our baseline as it allowed us to easily model predicted probabilities of class membership.

A. Logistic Regression (Baseline)

Logistic regression is a linear model where the variable to be predicted is categorical. It finds the best parameters β that fit the equation:

$$y = \begin{cases} 1 & \beta_0 + \beta_1 \boldsymbol{x} + \varepsilon > 0 \\ 0 & \text{else} \end{cases}$$

The error term is not observed, and the parameters are found by iterative search (usually Stochastic Gradient Descent). A probability of 0 represents no abuse, and 1 means that the classifier was perfectly sure that the comment was abusive.

B. Multinomial Naive Bayes

In Multinomial Naive Bayes, samples are the frequency with which events were generated by a multinomial distribution of $p_1 ... p_n$ where p_i is the probability that event i occurs. The feature vector $x = (x_1 ... x_n)$ is a histogram with x_i being the number of times event i was observed. The equation is then given by:

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i {p_{ki}}^{x_i}$$

While the Naive Bayes assumption (that each feature is independent of the value of any other feature, given the class variable) might not be very realistic in some cases, we decided to use it as it is known to be good at classifying spam [5] (a similar problem in some ways, given the binary nature of classification and the imbalance of classes in the data).

C. SVM

SVM (Support Vector Machines) are non-probabilistic binary linear classifiers. SVMs can perform non-linear classification using the kernel trick, which allows them to map inputs into high-dimensional feature spaces. It works by finding the maximum-margin hyperplane that divides the data into the two classes, which can be done by minimizing the norm of the normal vector to the hyperplane w, i.e., $\|w\|$, subject to $y_i(w.x_i - b) \geq 1$ for $i = 1...n$, where y_i is the data label, x_i is the data point, and b helps determine the offset of the hyperplane from the origin. SVMs have also been shown to perform well in anti-spam applications [6].

D. Data Pre-processing

We took the following steps to pre-process the data from the Wiki-Detox dataset:

1) *Data Cleaning*: We removed punctuation and certain special characters, and made all data lowercase. We also used lemmatization, which groups different inflected forms of the same word together.
2) *Choice of word embeddings*: We experimented with the following techniques:
   N-grams: N-grams are a simple, yet effective method of tokenizing inputs. We looked at combinations such as (1,1), (1,2), (2,2) and (2,3) n-gram lengths. Limiting the

maximum number of features (at 10,000) and using stop words (from a list of common English stop words) were also done.

GloVe embeddings: GloVe embeddings [7] are a powerful way to explore deep, hidden relationships between words and ideas. We experimented with a pre-trained set of GloVe word vectors available online [8]. To try to improve performance, we retrained the model on the corpus that we were working with, helping us generate word vectors which are more relevant to the problem statement. We found that the latter did worse, so the former was used.

3) *TF-IDF weighting*: This was used to normalize the word frequency. TF-IDF is proportional to the word occurrence but offset by the frequency of the word in the corpus, thus helping adjust for the fact that some words appear more frequently.

We incrementally incorporated each of these techniques. The observations are discussed in Section VI.

V. Methods: Deep Learning Models

We explored four deep learning models: Multi-Layer Perceptron (MLP), Convolutional 1D Neural Networks, LSTM, and Convolutional LSTM.

A. Multi-Layer Perceptron

A Multi-Layer Perceptron is the standard neural network. We used ReLU as the activation function and back-propagation for training, along with a combination of Regularization, Dropout Layers and Batch Normalization layers. We used a Multi-Layer Perceptron of size 200x200x50 and ran it on input data encoded using 1-grams as well as GloVe embeddings.

B. 1D Convolutional Neural Network

Textual sequences can be treated as 1D images where we can perform 1D Convolution [12]. This is the concept of a sentence. We used 4 Convolutional layers of output size 128 and kernel size 5, then Max Pooling, then 3 Convolutional layers of output size 128 and kernel size 5, then Average Pooling. The equations are given in subsection D.

C. LSTM

LSTM (Long Short Term Memory) networks are a type of Recurrent Neural Network (RNN) that are capable of learning long-term dependencies in the data. LSTMs have the same chain structure as RNNs, but the repeating module has four main components: a cell, an input gate, and output gate and a forget gate. The cell serves as the memory, and the gates act like neurons by computing an activation of a weighted sum, regulating the flow of values through the LSTM.

We used an LSTM layer with 128 LSTM units to classify the comments. Although we only used one layer, we obtained results much better than the baseline. The equations are similar to those given in subsection D, but with the product of terms rather than the convolution.

D. Convolutional LSTM

Convolutional LSTM is an extension of the (fully-connected) LSTM with convolutional structures in both the input-to-state and state-to-state transitions. The equations are shown below:

$$
\begin{aligned}
f_t &= \sigma_g(W_f * x_t + U_f * h_{t-1} + V_f \circ c_{t-1} + b_f) \\
i_t &= \sigma_g(W_i * x_t + U_i * h_{t-1} + V_i \circ c_{t-1} + b_i) \\
o_t &= \sigma_g(W_o * x_t + U_o * h_{t-1} + V_o \circ c_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c * x_t + U_c * h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

We used 2 layers of CNN with an output size of 128 and a kernel size of 5, with Max Pooling. We then used 1 layer of LSTM with 128 LSTM units. Our results are discussed in Section VI.

VI. Results and Analysis

Similar to existing work on this dataset [3], we use Area Under the Curve (AUC) in the ROC curve as one evaluation metric and F1 score as the other. We do not report accuracy due to the skew in the dataset having much fewer insults and accuracy not being a differentiating factor (compared to F1) while comparing models. The F1 score is calculated at a 0.5 threshold.

A. Baselines and Conventional Models

Logistic Regression was run in order to first establish a baseline for performance on this dataset. We incrementally fine-tuned the hyper-parameters, experimenting with several pre-processing and representation methods as shown in Figure 1 (left).

For the baselines, we observe that modifying the n-gram range from only 2-grams, to 2-grams and 1-grams, and then to only 1-grams caused the most significant performance increase for logistic regression. This intuitively made sense - in most insulting comments, the key insult shows up in one specific word rather than in combinations of words. Adding a list of common English stop words caused a marginal performance increase; however, the effect of restricting the number of features to top 10,000 did not cause any improvement. Reversing the order of the preceding two steps also gave us similar results, leading to the conclusion that they were performing similar functions (we decided to restrict the features rather than use stop words).

We then implemented a TF-IDF weighting scheme, which also contributed to ensuring that common words were not given undue importance. A combination of punctuation removal, case conversion, tokenization and lemmatization was the final preprocessing step; this ensured that similar ideas were treated as the same feature and not independently. These preprocessing steps were used in training the other conventional models (Multinomial Naive Bayes and SVM).

We experimented with regularization as well, varying the weight given to the regularization term. The values of the parameter C (the inverse of the usual regularization parameter λ; increasing C implies less regularization) was varied from

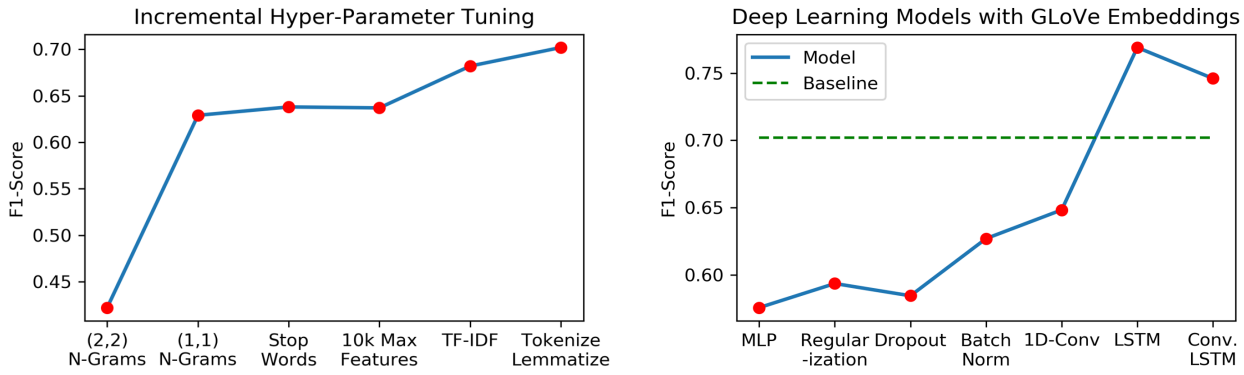

Fig. 1. Incremental tuning of hyperparameters and models

TABLE I
COMPARISON BETWEEN EMBEDDING SCHEMES

	F1 Score	ROC AUC
GloVe (pre-trained)	0.317	0.802
GloVe (trained on personal attacks data)	0.326	0.811
1-grams	0.514	0.919

TABLE II
COMPARISON BETWEEN MODELS

	F1 Score	ROC AUC
Logistic Regression	0.702	0.959
Multinomial Naive Bayes	0.598	0.933
SVM (linear kernel)	0.734	N/A
MLP (with TFIDF ngrams)	0.747	0.952
MLP (GloVe)	0.635	0.911
ConvNets (GloVe)	0.648	0.917
LSTM (GloVe)	0.771	0.962
ConvLSTM (GloVe)	0.746	0.957

0.01 to 100 (multiplying by 10 in each subsequent attempt). The best performance was seen while setting C = 10 (lower regularization than default, which was C = 1), showing that there was some inherent bias in the original baseline model.

We then substituted n-grams with GloVe embeddings [7], first with a pre-trained set of GloVe vectors available online, generated from Wikipedia data [8]. We observed that the performance with n-grams was actually higher than that with GloVe. We found this to be partially based on the reasoning presented in [9], and also from the fact that we need to find better ways of combining the embeddings. In our tests, only the mean of all the word embedding was used. We also re-trained GloVe vectors on our specific dataset (of attack-annotated comments), which improved relative performance, but still suffered similar issues as the pretrained Glove embeddings, as shown in Table I.

SVMs have been shown to perform well on text-classification tasks such as spam classification [6]. As mentioned, the problem statements are similar in that they are both binary classification on a skewed dataset. We ran SVM with different kernel types, namely sigmoid, Gaussian (RBF) and linear. As shown in [10], linear kernels resulted in the best performance. We observed that SVMs outperformed the Logistic Regression and Multinomial Naive Bayes models.

B. Deep Learning Results

As a preliminary step to judge the performance of deep learning models with the same inputs as conventional models, we used a Multi-Layer Perceptron to train on the same 1-gram data that was used for the logistic regression baseline. We observed that the MLP gave significantly better performance (as measured using F1 score) compared to all conventional models that we had trained on. This indicated to us that our model could be significantly improved using deep learning methods.

Following this, we wished to understand if using sequential contextual information would be of consequence to our particular problem. Therefore, we shifted from TF-IDF based 1-gram representations to the whole sentence being represented as enbeddings. We used GloVe embeddings to represent our input data as a sequence of embeddings. To help with the training, the first layer of the neural network incorporates the embedding layer which holds the GloVe embeddings of each word. Consequently, as each one-hot word vector passes through the layer, it gets transformed into an embedding. The exact architecture was as described in the previous section.

After observing the results of the standalone MLP (with a learning rate of 0.01), we began to tune the performance in several steps. First, we incorporated L2 regularization in each layer with a constant of 0.01, this improved model performance and reduced overfitting. Next, we added a dropout layer with a dropout rate of 0.25. Noticeably, we see that the effect of the dropout layer is similar to L2 regularization - it forces the neurons to be more robust to change. Although we lost a little in terms of performance, this is mainly attributed to the dropout probability being 0.25. Lower probability values gave us similar results so we kept it at 0.25 as it was reasonably consistent with existing work. We then incorporated a batch normalization later to prevent the final (sigmoid) layer from getting saturated. We see that this further

improved performance, especially on some deeper variations of the networks. All of these results can be seen in Figure 1. Finally we used the 'Adam' gradient descent method with the parameters: Learning rate = 0.01, $\beta_1 = 0.999$, $\beta_2 = 0.99999$, $\epsilon = 1e-08$, decay = 0.00001. We noticed that the major difference of adding momentum and reduction of learning rate via Adam resulted in faster convergence but not better results. Thus we have kept Adam as the de-facto optimizer for the upcoming models.

Post this, we wanted to explore the use of CNNs and LSTMs for detecting personal attacks (Table II). We observed that performance improved significantly with a CNN, showing that there was some value in analyzing location-dependent contextual information. We saw that for smaller kernel sizes (such as 5 and 7) it performed marginally better than larger ones (such as 11). We decided to use the kernel size 5 as it combines the context of close words. Furthermore, we could compare it to the LSTM which combines the context of all the words in that sentence. When we used a LSTM network, we were able to achieve the best possible results across all models that we tried. This meant that capturing long range dependencies was definitely of significance to be able to predict whether a sentence is abusive or not. Interestingly these results were obtained using a one layer LSTM with 128 units, compared to the CNN, which had 4 Convolutional layers. Even in deeper convolutional networks, the result was not as good. This can be because CNNs only capture dependencies of words close by. Hence their performance is comparable to RNNs, but the memory unit of LSTMs give it an edge over CNNs/RNNs. Finally, we also tried using a Convolutional LSTM, in which we convolve the input before feeding it to the LSTM. This gave very good results but marginally under-performed the LSTM. Interestingly, the major advantage of CNNs over LSTMs in this case is the computational complexity, they train much faster and we could build deeper networks that could be trained on our local machines. LSTMs had a large training time.

C. Does Contextual User Data Help?

Looking into the subset of Wikipedia comments from the year 2006 (containing a total of 2,916,313 total comments) we see that only about 88,082 unique users made these comments in the dataset. Due to the repetitive nature of the data, i.e. the same users making comments, knowing contextual information about the user actually helps us in adding a prior of sorts on the users to help predict whether they will make an abusive comment or not.

We see that only 91,226 comments out of the 2.9 million were made by anonymous users in 2006. We tried to quantitatively infer if anonymous users were more likely to make abusive comments and determine if we could interpret this through our model.

We wanted to answer this question in 2 ways: 1) Whether actual user information makes a difference, which meant incorporating the user ID directly into the model as a one hot vector scaled by a parameter α. α is varied from 0.1 to 1. 2) Whether an anonymous user is more likely to make an abusive comment. Looking at the test scores for question 1, we see that we gain no extra information about users. This could be attributed to the representation (one-hot). However, it did not make a difference on subsets of data either, so we concluded that no extra information could be gained by adopting this method. This is consistent with what Cheng et al. [18] presented regarding abusive behavior. Anyone can become an abusive troll: it depends on how they have been instigated rather than on their profile features.

TABLE III
INCORPORATING CONTEXTUAL USER DATA

Anonymous Info (With/Without)	F1 Score	ROC AUC	Recall
Logistic Regression Without	0.488	0.913	0.32
Logistic Regression With	0.544	0.907	0.38
MLP Without	0.616	0.970	0.52
MLP With	0.632	0.876	0.54

For question 2 we look at the test scores for logistic regression and MLP over the baselines where we add a new parameter determining if the comment was made by an anonymous user or not. The values reported are for the best α and we see that knowing if a user is anonymous or not does indeed affect the model performance. It is observed that there is clear improvement (in terms of F1 score) after adding this extra dimension to the data, depicted in Table III. Notably what this model improves the most is the recall, although precision and accuracy remain more or less the same. We infer that knowing anonymity can actually help in increasing relevant comments retrieved by the model, thus behaving as a prior.

VII. CONCLUSION AND FUTURE WORK

This project tackled the issue of offensive language in social media, focusing on personal attacks, i.e. abusive comments targeted at the recipient. We established and thoroughly evaluated three traditional models: Logistic Regression, Multinomial Naive Bayes and SVM. We then explored four deep learning models: MLP, CNNs, LSTM and Convolutional LSTM. Each of these models was incrementally tuned to obtain the set of hyperparameters yielding the best results. We then investigated the question of whether contextual user data can aid classification. We determined that indeed, anonymous users are more likely to post abusive comments than non-anonymous users.

This project has great scope in terms of being able to answer open-ended questions plaguing social media platforms. Some areas that we hope to address in the future are:

1) Are some article topics more likely to incite abusive comments than others?
2) Temporal characteristics: is there an observable trend in the quantity/nature of abusive comments over time?
3) The effect of incorporating worker demographics (age, gender, education) on classification to determine and handle worker bias to improve the dataset quality.

Answering these questions would be very helpful to social media platforms aiming to curb hate speech and personal attacks on their websites [4].

VIII. Contribution

All three members sat together, studied related literature, and worked on running different models with several variations (in terms of hyper-parameter and architectural modifications) in order to tackle our problem statement. We then spent time reasoning through the results obtained from these models, and determining how to present our findings.

References

[1] O. Erdur-Baker. Cyberbullying and its correlation to traditional bullying, gender and frequent and risky usage of internet-mediated communication tools. In New Media and Society, 2009.

[2] Google Perspective. *https://www.perspectiveapi.com/*

[3] E. Wulczyn, et al. Ex Machina: Personal Attacks Seen at Scale. In IW3C2, 2017

[4] Impermium. Detecting insults in social commentary dataset, 2012. *https://www.kaggle.com/c/ detecting-insults-in-social-commentary*

[5] I. Androutsopoulos et al. Learning to Filter Spam E-Mail: A Comparison of a Naive Bayesian and a Memory-Based Approach. In PKDD, 2000.

[6] H. Drucker et al. Support Vector Machines for Spam Categorization. In IEEE Transactions on Neural Networks, 1999.

[7] J.Pennington et al. GloVe: Global Vectors for Word Representation. In EMNLP, 2014.

[8] GloVe. *https://nlp.stanford.edu/projects/glove/*

[9] C. Nobata et al. Abusive language detection in online user content. In WWW, 2016.

[10] T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In Technical report, LS VIII Number 23, Univ of Dortmund, 1997.

[11] E. Wulczyn et al. Wikipedia Detox. figshare. 2016 *doi.org/10.6084/m9.figshare.4054689*

[12] Y. Kim. Convolutional Neural Networks for Sentence Classification. In EMNLP, 2014.

[13] Y. Dawei et al. Detection of harassment on web 2.0. In CAW2.0 Workshop at WWW, 2009.

[14] D. Yogatama et al. Generative and Discriminative Text Classification with Recurrent Neural Networks. *arXiv:1703.01898*

[15] P. Kawthekar et al. Evaluating Generative Models for Text Generation. *https://web.stanford.edu/class/cs224n/reports/2737434.pdf*

[16] X. Wan et al. Predicting Polarities of Tweets by Composing Word Embeddings with Long Short-Term Memory. In ACL, 2015.

[17] Y. Kim, Convolutional Neural Networks for Sentence Classification. In EMNLP, 2014.

[18] J. Cheng et al. Anyone Can Become a Troll: Causes of Trolling Behavior in Online Discussions. In CSCW, 2017.

[19] T. Davidson et al. Automated Hate Speech Detection and the Problem of Offensive Language. In ICWSM, 2017.