CAPP 30255 Advanced Machine Learning for Public Policy
University of Chicago

Minimizing Unintended Bias in Toxicity Classification

Final Paper

June 6, 2019

Hye Yeon Chang
EVS Vishwanath
Elena Badillo Goicoechea

Abstract

Our project attempts to classify toxicity in online comments, while also reducing the "unintended bias" that gets introduced into the models. Bias here refers to the situation where models falsely classify comments as toxic because they contain identity groups, due to the training data containing toxicity directed towards certain identity groups.

We attempt bias reduction by replacing words which might correspond to identity groups with identity neutral words ("she" converts to "they") so that the classifier doesn't train on identity words.

Our basic modelling approach is as follows:

1. Perform text cleaning (cleaning special characters, mapping contractions, etc)
2. Vectorise text (using BoW, TF-IDF or word embeddings)
3. Train shallow models to establish a baseline
4. Train LSTM based model to improve on previous baseline
5. Compute the precision on data subsets containing identity groups and not containing identity groups
6. Find the difference between the above two to calculate "bias"
7. Find weighted average of bias and precision to evaluate overall model performance

We find that our bias reduction results in sight gains in the neural network based model.

1. Project Description / Motivation

In the last few years there has been a rising interest from policy, industry and academia in correctly distinguishing *toxic* comments in online social spaces, defining toxicity as "anything rude, disrespectful or otherwise likely to make someone leave a discussion." While not something that is expected to be directly implemented by governments/policy makers, we believe that systems capable of analyzing and identifying toxicity in discussion forums would be beneficial to organizations that aim to improve diversity and reduce exclusion in their workplace, or to prevent their product users from experiencing emotional distress.

Deploying such systems to monitor internal chat channels - especially ones that are anonymous, could give a sense of the degree of "toxicity" in an organization. Tangentially, tools for successfully detecting toxicity in textual data might help organizations to faster target more "intense" undesirable online behaviors such as cyberbullying, hate-speech, harassment, among others.

A natural approach for doing such classification in a systematic way is using machine learning models, trained with labeled text data available in online conversations. Even though there have been several relatively successful attempts to do so, as we detail in the next section, an important challenge faced by some of these models is that they tend to incorrectly associate the mere occurrence of names of frequently attacked identities with toxicity.

For instance, predicting a high likelihood of toxicity for comments containing those identities (e.g. "gay"), even when those comments were not actually toxic (such as "I am a gay woman"). This happens because certain identities are overwhelmingly referred to in offensive ways, and so, training a model from data with these underlying imbalances risks mirrors those biases back.

As a response for this, our project's goal is to build a machine learning model -based in neural networks and NLP techniques - that both correctly recognizes toxicity and minimizes the aforementioned unintended bias with respect to mentions of identities. In short, we want to build a "better" toxicity classifier in the sense that it does well across a diverse range of conversations.

In this sense, our project has the following objectives:

- Implement a strategy to improve upon the *bias/misclassification* baselines (as far as possible) found in the previous approach

- Develop a new score/objective function to optimize, which takes into account both predictive performance and the potential bias associated with non-hateful terms

2. Literature Review

As mentioned in the previous section, the task of correctly discriminating toxic from toxic comments has been tackled using various approaches, ranging in complexity from naive rule-based "blacklists" or regular expression matching, to state-of-the art ensemble or convolutional neural networks (CNN) methods.

Given the high complexity of the task at hand, machine learning (ML) approaches -both CNN and non CNN- have proved to be significantly more effective.

Among ML methods, successful examples of non-CNN approaches include **Nobata et al. (2016)** who perform a supervised classification using ensemble methods and NLP features of four broad classes: N-grams, Linguistic, Syntactic and Distributional Semantics. Using all features on their training set, their model yields an AUC of 0.9055, outperforming some benchmark deep-learning approaches.

Successful examples of CNN approaches include **Georgakopoulos et al. (2018)** who classify toxic comments in a large pool of documents, comparing their accuracy/specificity against traditional bag-of-words approaches along with Naive Bayes and other traditional classifiers, providing enough evidence that the former can enhance toxic comment classification performance.

Along the same lines, **Biere and Bhulai (2018)** use Twitter data to train a neural network with final accuracy, precision, recall, and F-measure values between 0.90 and 0.91, beating other approaches. Interestingly, they find that the error rate is higher for the most "hateful" comments, setting foot for further research.

Even if ML approaches are unarguably better in terms of predictive performance, several challenges remain. For instance, as we mentioned in the previous section, there is an "organic" inherent bias in the weight these classifiers assign to the presence of words identifying commonly attacked groups.

This issue has been acknowledged (and explicitly addressed in their second-round Kaggle competition call) by **Jigsaw (2018)** who explain that "the names of targeted groups appear far more often in abusive comments (...) When the training data used to train machine learning models contain these comments, ML models adopt the biases that exist in these underlying distributions, picking up negative connotations as they go."

In addition, **Hosseini et al. (2017)** identify a key vulnerability of virtually all the classifying methods we have reviewed below: an "adversary" can learn by feedback and respond by subtly modifying a highly toxic phrase in a way that the system assigns significantly lower toxicity score to it.

We choose to focus on tackling the first challenge in our project, since we consider that it can be mapped much more directly into a machine learning problem.

3. Data

The data for this task is supplied by Kaggle and was released in 2017 after the Civil Comment platform shut down and made ~2 million public comments from their platform available for researchers' use. Further, Jigsaw extended annotation of this data by human raters for various toxic conversational attributes.

The data consists of a **1.80m-row** training dataset and a **97.3k-row** test dataset, containing text of the individual comment (comment_text column).

Each comment in Train has a toxicity label (target column), and models should predict the target toxicity for the Test data, where the default is that examples with target >= 0.5 will be considered to be in the positive class (toxic).

The toxicity attribute represents the fraction of human raters who believed the attribute applied to the given comment was "toxic" or not. The data also include several additional toxicity subtype attributes (e.g. severely toxic, obscene) also computed as described below.

In addition, we plan to extend the original dataset in order to feed our models with more potentially informative features derived from the comments and/or metadata.

4. Implementation Details

4.1. Objective and Planned Approach

Our project had two objectives:

1. Find a way to quantify "unintended bias" in training datasets, so that future datasets can be evaluated for similar bias. Unintended bias here means the propensity to classify a comment as toxic due to the disproportionate occurrence of identity groups in toxic comments
2. Strip out the bias from the "biased" identity group features so that predictions aren't affected by the presence of an identity group

Our workflow consisted of the following steps:

1. Conduct EDA on the input text data to find out if the data-generating process itself is inherently biased
2. Conduct EDA on the input text data to identify potential toxicity-indicating features that are agnostic of the presence of bias groups
3. Formulate a quantitative measure for the bias in the dataset, which models can be trained to minimise
4. Clean and process the data in preparation for both linear and neural-network based modelling
5. Run "baseline" models which don't take any "debiasing" steps into account
6. Run "debiased" models which take into account our bias reduction strategies
7. Compare the results to establish the reduction in bias, and if possible, a bias-accuracy trade-off

4.2. Planned Bias Reduction Approaches

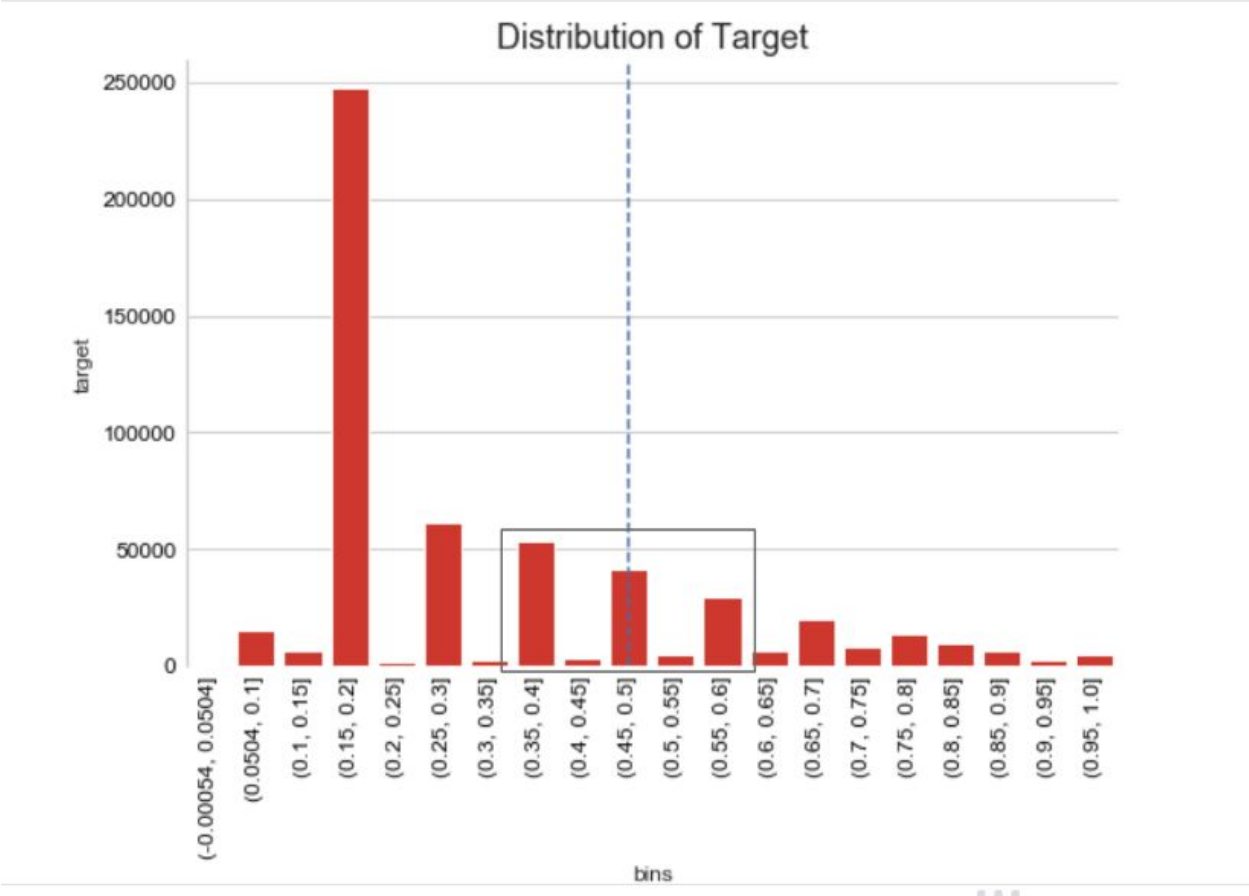
We considered the following approaches to reduce bias in the datasets

1. Replace identity words and pronouns with identity neutral equivalents. For example, convert "she" into "they" or "christian" into "person".
2. Attempt to account for identity bias by including sentiment polarity as an additional feature in our model
3. Create an alternative loss/scoring metric which will attempt to penalise the weights associated with identity group features
4. Use word embeddings to construct identity-group vectors that are "stripped" of negativity

4.3. Initial Text EDA

Data Generation Process EDA

Our first line of EDA was to consider the distribution of proportion of reviewers who considered a comment toxic. We observed that the vast majority of comments were considered not toxic - which was in line with expectations. However, we found that the proportion of reviewers who found a comment toxic varied smoothly around the 0.5 threshold.



A bimodal distribution with clear delineation on either side of the threshold would have indicated that the comment reviewers are clear about which comments are toxic and which are not. However, the smooth distribution gives us *prima-facie* reason to suspect that the toxicity flagging process itself may not be able to distinctly separate the features which identify toxicity.

Feature Identification EDA - Sentence Characteristics

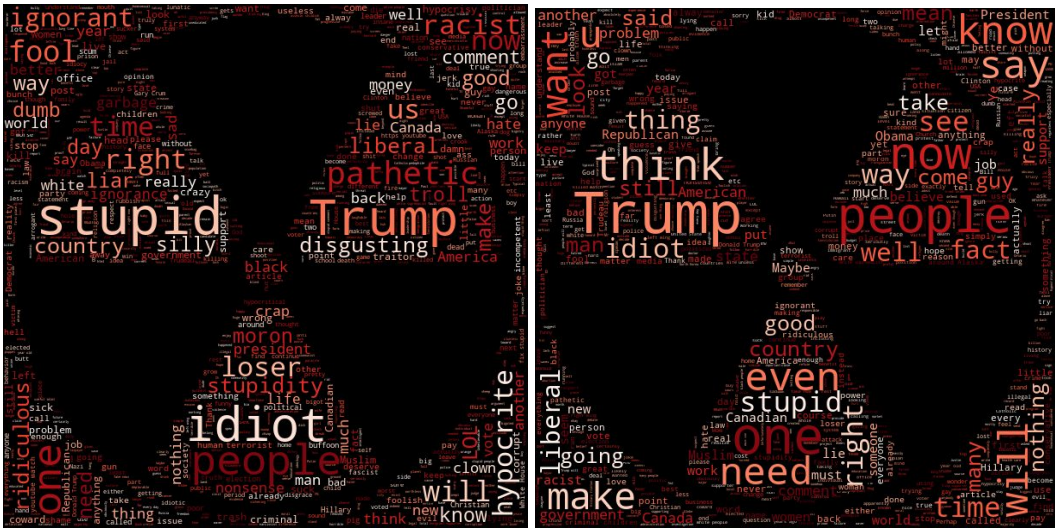
We study the pointwise biserial correlation between different sentence structure characteristics and toxicity to evaluate how the *structure* of the sentence might be an indication of toxicity. Our main hypothesis here is that better structured sentences (proper use punctuation, not using capital words throughout, etc) might be indicative of lower toxicity. The results are reported in the table below:

Sentence Structure Feature	Correlation with Toxicity
Proportion of full words that are capitalised	0.007
Proportion of text that consists of punctuation	-0.012
Proportion of characters that are capitalised	-0.002

As observed from the above table, none of the sentence structure features have any correlation to speak off with toxicity. This also allows us to clean these punctuations and lowercase the words in interest of a concise vocabulary.

Feature Identification EDA - Sentence Tail Analysis

We found a hypothesis on the Kaggle forums detailing how the "toxicity" of a sentence is confined to the last part of the sentence. We accordingly created wordclouds of the last five words in a sentence relative to remaining part of the sentence. Initial review of these two wordclouds indicated that the later part of the sentence did indeed seem to be "more toxic"



Last 5 Words Rest of Sentence

4.4. Quantifying Bias and Model Evaluation

We attempt a rough quantification of bias by finding the rate of false positives for the data subset containing identity related hate, and comparing this to the false positives for the data subset without identity related hate (holding the probability threshold constant across both the subsets). The difference between these two measures is then considered the "unintended bias" in the model - a measure of how much it overweighs the presence of identity groups.

The specific modelling approach involves:

- Training a classification model
- Classify comments in a validation set
- Partition the validation set into two subsets:
    - A subset where at least one of the identity group columns is marked "present" (set 1)
    - A subset where *none* of the identity group columns are marked present (set 2)
- Calculate the precision *at the same threshold* on both the subsets
- The difference precision(set2) - precision(set1) is considered as the bias in our model

Given this quantification of the bias, we then proceed to define a new model evaluation score, calculated as the weighted average of precision on the full test set and the above bias score. The weights currently are picked arbitrarily by the user, depending on how much priority needs to be given to overall precision vs bias reduction.

It should be noted, however, that our implementation does not use the above score in the cross-validation/parameter tuning process. Our current implementation only calculates the above score after the model training process as a "final indicator" of the model performance.

4.5. Data Cleaning and Preparation Process

Data Cleaning:

Our data cleaning pipeline implements the following functionalities, which are executed depending on the user input:

Lowercase all words:

Having both upper and lowercase words expands the vocabulary more than is needed ("Hello" and hello are registered as two different words as far as the model is concerned). To account for this, we convert all words into lower case.

Clean Special Characters:

Non alpha-numeric characters might be present in the word as typos, punctuation or simply as artefacts of online typing. The presence of such characters might again add unnecessary noise to the vocabulary. The clean option strips out the alpha-numeric characters.

Spell Check:

A summary reading of the comments indicates that certain alpha-numeric and non-alphanumeric sequences might have to be treated with a little bit of sophistication (f**l might be fool, written in a manner to avoid censors), and are - in theory - handled by the spell checker option. In practice, however, we find the spell checker to be intractable for large datasets due to the amount of time needed to check and correct each word of the corpus.

Stop Word Removal:

Words like "the", "of", etc are considered "stop words" in the English language - words that are extremely common, and don't really add much information to a typical classification task. We provide users with the option to remove stop words.

Stemming:

Given that the same word can take different forms (go, going, gone), our vocabulary runs the risk of expanding with multiple variations of the same words. Some tasks however, may only require the presence of the word regardless of the form. Stemming reduced words to their basic form by removing either prefixes or suffixes so that the vocabulary space is made more manageable.

Contraction Mapping:

For the sake of consistency, we'd want to treat contractions the same as their expanded form. We construct a dictionary of common contractions and their expansions, and replace the contractions with their expansions (can't with cannot).

Group ID word replacement:

One of our proposed debiasing strategy involves replacing the group identity words with group neutral equivalents. We construct a manual dictionary of identity words and their corresponding neutral equivalents and replace these occurrences in the dataset.

Vectorisation

Our text vectorisation converts the cleaned comment text into vectors, depending on the modelling strategy.

Bag-of-words:

We implement a simple bag of words vectoriser with a user defined option for the vocabulary size. The BoW vectoriser simply represents each comment as a vector consisting of the number of occurrences of each word. We use the standard sklearn implementation for this.

Term Frequency - Inverse Document Frequency:

The TF-IDF vectoriser explands on the BoW by inverse weighting each word by its frequency *across* documents. We use the standard sklearn implementation for this.

Word-Embedder:

We also implement word embeddings that learn underlying semantic relations between words. We explored both fasttext and continuous-bag-of-words word2vec. We finally decide to implement word2vec as it's faster, and the out-of-vocabulary cases which fasttext handle are handled by being replaced with 0-vectors. We use the gensim implementation, trained on ~300k words, with a 2-word context window and a dimensionality of 64.

4.6. Shallow Models

We implement "shallow" machine learning models to establish a baseline performance expectation to improve on with our Neural Network based models.

Implemented Text Preprocessing

Based on the discovery from our exploratory analysis that upper case letters and punctuations are not correlated with toxicity, we choose to safely remove these from our corpora. We also stem the words to represent words based on their semantic root and to keep the size of our vocabulary manageable. We skip the spell check out of scalability concerns and skip removing stop-words as we expect TF-IDF to downweight words that occur frequently across the whole corpora accordingly. We finally run each of the models with and without the identity word replacement in order to test the effect of replacing identity words on model performance.

Pipeline Design

Our machine learning pipeline consists of two components, one for staging the data for the model and the other for running the experiment.

Feature Staging

The part of the pipeline which stages the data contains methods to:

- **Load** the data using user-specified columns, randomly sampling a subset of the data
- Binarize the target variable
- Split the data into X and y
- Extract features and creating Bag of Words or TF-IDF vectors from the user-specified text column
- Randomly split the data into a training and testing set.

Experiment Run

We then have three options for running an experiment:

- **Build**: Experiment run trains new models using grid search, selecting best model based on criterion defined by user. The grid search method only tunes parameters on the training set and measures performance on the test set so we can obtain a set of hyperparameters.
- **Load**: Experiment run loads previously trained model from storage and calculates evaluation metrics.
- **Refresh**: Experiment run re-trains model using parameters defined by previous build run with new data.

Once a model has been trained, we provide the option of storing the model object for subsequent reuse or re-training. Retraining a model could involve two possible approaches. In the first approach, we could find a new set of hyperparameters which are able to learn patterns in the new data better. Alternatively, we could freeze our set of hyperparameters, and train the model to find patterns in the new data.

Evaluation Suite

Finally, we have a suite of methods that include the set of traditional evaluation metrics like precision, recall, f1 score, and AUC-ROC score to measure the model's overall performance, but also methods specifically aimed at measuring the bias across different subsets of our data. Some key methods include:

- **Word importances**: For logistic regression or random forest models, obtain the tokens used in training the model in ascending order of importance by inputting the original training or testing dataset.
- **Evaluate bias**: Obtain all evaluation scores on testing dataset for each subset of comments identified as containing mentions of particular identity groups.
- **Plot bias**: Plot user-specified evaluation metric for each subset of comments identified as containing mentions of particular identity groups.

For our modeling task, we train three types of models, logistic regression, random forest, and naive bayes, using TF-IDF features. We subsample 5% of the data, among which we use 80% for training and the remaining 20% for testing. We select the best performing model for each type of classifier by searching across a grid of hyperparameters and optimizing for AUC ROC. We use the "load" mode of our experiment pipeline to further interrogate our classifier on the bias metrics we are trying to measure on our test dataset using the evaluation suite described above.

Results

For our test set, which consists of 18,049 examples, we evaluate each model's performance by calculating the precision score on the testing dataset, or the portion of total predicted positive

observations that is actually positive, when we classify all comments whose predicted probability scores are above 0.5. We select precision because we are especially interested in making sure that the comments predicted by the model as toxic are in fact toxic, and particularly interested in misclassification of examples on account of containing mentions of frequently attacked minorities.

	No Bias Adjustment	**Bias Adjustment**
Logistic Regression	0.57	0.56
Random Forest	0.80	0.91
Naive Bayes	0.97	0.90

Logistic Regression Coefficient Analysis

In addition to the performance metrics, we look into tokens that are significant in to the logistic regression's prediction of of toxicity. To understand group specific performance of the logistic regression, we then look at how semantically similar these strong predictors are to some words which might identify one identity group or another. The full list of words and their group associations are present in the appendix, but we do identify that certain groups are strongly associated with certain "toxicity predictors". A second result that we find interesting is that certain forms of the same word maybe more associated with toxicity than others ("jew" vs "jewish")

4.7. Neural Network Sequence Based Models

We attempt to improve on the performance of our baseline shallow models by implementing neural network based classifier models. We specifically attempt to implement sequence based models which, in contrast to the featureset in the shallow models, also attempts to find toxicity in the *sequence* in which words occur.

Our NN-based pipeline broadly consists of:

- Processing text from comments (with/without bias correction)
- Tokenizing text into words
- Building a word embedding matrix from the constructed vocabulary
- Feeding the word embedding as the first layer of a bidirectional LSTM model with a 1-dimensional output (binary classification).
- Classifying comments according to the class prediction
- Evaluating performance with / without bias correction

Implemented Text Preprocessing

As part of data cleaning, we choose to lowercase all the words, remove special characters and map contractions. Given that RNNs might identify additional information from the combination of word

position and word form (tense, etc), we choose not to stem our words. For a similar reason, we choose not to remove stop words either.
After the text is cleaned, we vectorise the text by extracting the CBOW Word2Vec embeddings for each text, and pass them into the RNN model. Out-of-vocabulary words are handled by being represented with 0-vectors.
We also create two sets of data by passing our data through the group identity word replacement, to evaluate the performance of this bias reduction strategy

LSTM

Long Short-Term Memory (LSTM) are a specific type of RNN model, and as such, they capable of learning **order dependence** in prediction problems. Among RNN models, LSTM have the additional trait of their default behavior being remembering **long-term sequences** in order to use that information to predict the occurrence of new ones. They can recall past long-term sequences, future long-terms sequences (unidirectional), or both (bidirectional). Traditional RNN models need to learn such long-term dependencies, rather than recalling them by default. Further, they are not very good at learning them.
Even if computationally more expensive than vanilla NN models or even CNN, LSTM models have been found highly effective when dealing with sequence-based data, such as text. For instance, when considering the sentence *"I think those who are planning to boycott the Women's March this year are retrogrades"*, one might notice that a model that does not take sequence order into account could more easily classify as toxic against women.
Taking all of the above factors into account, we considered -in line with the consensus for this type of task- that LSTM was an appropriate choice for building our NN pipeline, even if that meant that we would have to reduce our dataset size.

Network Design

LSTM Layers

In our LSTM architecture design, we followed consensus guidelines from academia and practitioners.
First, regarding number of hidden layers, according to Stathakis (2009), a single LSTM layer works well enough to extract the sequence based features, particularly when one is not trying to optimize accuracy. Thus, we set it to that value.

Second, the number of neurons in the LSTM hidden layer is ideally to be set according to the expression:

$$Nh = \frac{Ns}{\alpha*(Ni + No)},$$

where:

Nh is no. of units
Ns is dataset size
α is a scaling factor starting at 2
Ni is no. input neurons (embedding layer dimensionality)
No is no. of output neurons

In the interest of cutting on iteration time and computation, we opted for following a common thumb rule where the layer size is set as:

512 = 128 [convention in NLP tasks] *** 2** [Accounting for bidirectionality] *** 2** [scaling factor]

Our final output layer is 1-dimensional, since our classification task is binary and so the main output is a vector of probabilities. We chose a bidirectional over a unidirectional LSTM only preserves information of the **past** (only inputs it has seen are from the past). Using bidirectional will run inputs in two ways, one from past to future and one from future to past, as shown in the following example:

U: The boys went to … (PREDICT NEXT WORD)

B: The boys went to … (PREDICT NEXT WORD)...and then back home.

As a consequence, bidirectional LSTM perform better, in general (of course, at the expense of computational cost).
Finally, we freeze these hyperparameters based on convention and computational ease.

- Optimiser: Adam
- Loss Function: Binary log cross-entropy
- Learning rate: 0.001
- #Epochs = 4
- Batch Size = 512

Results

We ran our NN pipeline on 5% of our dataset under identical conditions, one time for our baseline model (with no bias correction) and one time for our bias-corrected model. The main results are shown in the table below, indicating that overall performance improved after bias correction, with change not coming only from bias reduction but from a slight improvement in precision as well.

	No Bias Correction	Bias Correction
Accuracy	0.942195	0.942158
Precision	0.678346	0.68169
Bias	0.0623	0.049363
Overall Performance	0.75615	0.762373

Bias decreased, as expected. However, it is worth noting that the improvement in precision was not expected and -in conjunction with the marginal decrease in accuracy- it might indicate a decrease in recall, so our model might have become slightly worse at capturing true toxic cases (less powerful).

5. Conclusions and Future Work

Our preliminary results indicate, overall, that our bias correction strategy worked as expected, for NN model, improving overall its performance. Results for our non-NN models were mixed but as pointed out previously, that might have to do with the inner workings and implicit assumptions of some of these models (e.g Naive Bayes' independence)

After the insights we learned from the exercises described in this paper, we are confident that a lot of future work can be done to improve toxicity detection models. Aside from reducing hardware constraints that would allow using the whole training set (as opposed to just a small sample) and deeper hyperparameter tuning process, we can identify at least two major sources of further work and improvement on the pipelines we designed.

The first is related to **bias correction** methodology and the second one is related to **bias measurement**. Even if our preferred bias correction strategy -identity word replacement- turned out to be successful in that it increased overall performance, we acknowledge that other options we previously identified and described in section 4.2 should be explored.

In particular, adding **sentiment polarity** and identity presence scores as features could help the model in its learning to distinguish toxic+identity from identity+nontoxic cases.

Finally, we initially considered -as a potential stretch goal-implementing systems that are resilient to **adversarial modifications** to statements that allow them to bypass the toxicity filter, while continuing to remain toxic. We think that this avenue of research could be not only useful for bias

correction in terms of methodology, but interesting in itself, given the core motivation of our project.

6. Bibliography / References

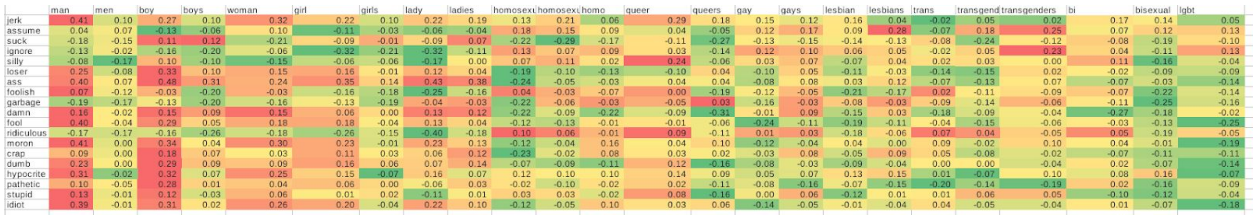
- Shanita Biere. "Hate Speech Detection Using Natural Language Processing Techniques". Vrije Universitet Amsterdam, 2018.
- C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang. Abusive Language Detection in Online User Content. In WWW, p. 145–153, 2016.
- D. Stathakis (2009). International Journal of Remote Sensing Vol. 30, No. 8, 20 April 2009, 2133–2147
- Hosseini, Sreeram Kannan, Baosen Zhang and Radha Poovendran. "Deceiving Google's Perspective API Built for Detecting Toxic Comments". Network Security Lab (NSL), Department of Electrical Engineering, University of Washington, Seattle , 2017.
- Jigsaw. "Unintended Bias in Toxicity Classification". Kaggle, 2019
- Jigsaw. "Unintended Bias and Names of Frequently Targeted Groups." Medium, 2018.
- Understanding LSTM Networks (Colah's Blog, 2015)
    - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Kaggle
    - https://www.kaggle.com/thousandvoices/simple-lstm
    - https://www.kaggle.com/bminixhofer/simple-lstm-pytorch-version
- LSTM in PyTorch (Holzner)
    - https://medium.com/@andre.holzner/lstm-cells-in-pytorch-fab924a78b1c

Appendix I - Logistic Regression Coefficients

The tokens most significantly predictive of toxicity are:

'jerk', 'of the word', 'assum that', 'suck', 'ignor', 'silli', 'loser', 'ass', 'foolish', 'garbag', 'damn', 'fool', 'ridicul', 'moron', 'crap', 'dumb', 'hypocrit', 'pathet', 'stupid', 'idiot'

Appendix II - Semantic Similarity between Predictors and Identity Groups



	man	men	boy	boys	woman	girl	girls	lady	ladies	homosexu	homosexu	homo	queer	queers	gay	gays	lesbian	lesbians	trans	transgend	transgenders	bi	bisexual	lgbt
jerk	0.41	0.10	0.27	0.10	0.32	0.22	0.10	0.22	0.19	0.13	0.21	0.06	0.29	0.18	0.15	0.12	0.16	0.04	-0.02	0.05	0.02	0.17	0.14	0.05
assume	0.04	0.07	-0.13	-0.06	0.10	-0.11	-0.03	-0.06	-0.04	0.18	0.15	0.09	0.04	-0.05	0.12	0.17	0.09	0.28	-0.07	0.18	0.25	0.07	0.12	0.13
suck	-0.18	-0.15	0.11	0.12	-0.21	-0.09	-0.01	-0.09	0.07	-0.22	-0.29	-0.17	-0.11	-0.27	-0.13	-0.15	-0.14	-0.13	-0.08	-0.24	-0.12	-0.08	-0.19	-0.10
ignore	-0.13	-0.02	-0.16	-0.20	-0.06	-0.32	-0.21	-0.32	-0.11	0.13	0.07	0.09	0.03	-0.14	0.12	0.10	0.06	0.05	-0.02	0.05	0.23	0.04	-0.11	0.13
silly	-0.08	-0.17	0.10	-0.10	-0.15	-0.06	-0.06	-0.17	0.00	0.07	0.11	0.02	0.24	-0.06	0.03	0.07	-0.07	0.04	0.02	0.03	0.00	0.11	-0.16	-0.04
loser	0.25	-0.08	0.33	0.10	0.15	0.16	-0.01	0.12	0.04	-0.19	-0.10	-0.13	-0.10	0.04	-0.10	0.05	-0.11	-0.03	-0.14	-0.15	0.02	-0.02	-0.09	-0.09
ass	0.40	0.07	0.48	0.31	0.24	0.35	0.14	0.43	0.38	-0.24	-0.05	-0.03	0.04	0.04	-0.08	0.08	0.03	0.12	-0.07	-0.13	0.07	-0.07	-0.03	-0.14
foolish	0.07	-0.12	-0.03	-0.20	-0.03	-0.16	-0.18	-0.25	-0.16	0.04	-0.03	-0.07	0.00	-0.19	-0.12	-0.05	-0.21	-0.17	-0.02	-0.11	-0.09	-0.07	-0.22	-0.14
garbage	-0.19	-0.17	-0.13	-0.20	-0.16	-0.13	-0.19	-0.04	-0.03	-0.22	-0.06	-0.03	-0.05	0.03	-0.16	-0.03	-0.08	-0.03	-0.09	-0.14	-0.06	-0.11	-0.25	-0.16
damn	0.16	-0.02	0.15	0.09	0.15	0.06	0.00	0.13	0.12	-0.22	-0.09	-0.22	-0.09	-0.31	-0.01	0.09	-0.15	-0.03	-0.18	-0.09	-0.04	-0.27	-0.18	-0.02
fool	0.40	-0.04	0.29	0.05	0.18	0.18	-0.04	0.13	0.04	-0.12	-0.13	-0.01	-0.01	-0.06	-0.24	-0.11	-0.19	-0.11	-0.04	-0.15	-0.06	-0.03	-0.13	-0.25
ridiculous	-0.17	-0.17	-0.16	-0.26	-0.18	-0.26	-0.15	-0.40	-0.18	0.10	0.06	-0.01	0.09	-0.11	0.01	0.03	-0.18	-0.06	0.07	0.04	-0.05	0.05	-0.19	-0.05
moron	0.41	0.00	0.34	0.04	0.30	0.23	-0.01	0.23	0.13	-0.12	-0.04	0.16	0.04	0.10	-0.12	-0.04	0.04	0.00	0.09	-0.02	0.10	0.04	-0.01	-0.19
crap	0.09	0.00	0.18	0.07	0.03	0.11	0.03	0.06	0.12	-0.23	-0.02	0.08	0.03	0.02	-0.03	0.08	-0.05	0.09	0.05	-0.08	-0.02	-0.07	-0.11	-0.11
dumb	0.23	0.00	0.29	0.09	0.09	0.16	0.06	0.07	0.14	-0.07	-0.09	-0.11	0.12	-0.16	-0.08	-0.03	-0.09	-0.04	0.00	0.00	-0.04	0.02	-0.07	-0.14
hypocrite	0.31	-0.02	0.32	0.07	0.25	0.15	-0.07	0.16	0.07	0.12	0.10	0.10	0.14	0.09	0.05	0.07	0.13	0.15	0.01	-0.07	0.10	0.08	0.16	-0.07
pathetic	0.10	-0.05	0.28	0.01	0.04	0.06	0.00	-0.06	0.03	-0.02	-0.10	-0.02	0.02	-0.11	-0.08	-0.18	-0.07	-0.15	-0.20	-0.14	-0.18	0.02	-0.16	-0.09
stupid	0.13	-0.01	0.12	-0.03	0.06	0.01	0.02	-0.11	0.01	0.03	0.03	-0.02	0.08	-0.16	0.00	0.06	-0.12	0.01	0.01	0.06	0.05	-0.10	-0.12	-0.04
idiot	0.39	-0.01	0.31	0.02	0.26	0.20	-0.04	0.22	0.10	-0.12	-0.05	0.10	0.03	0.06	-0.14	-0.05	-0.01	-0.04	0.04	-0.05	-0.04	0.01	-0.07	-0.18

	white	whites	mexican	mexicans	immigrant	immigrants	black	blacks	african	africans
jerk	0.12	0.12	-0.04	-0.04	0.01	-0.13	0.11	0.02	-0.07	-0.12
assume	0.09	0.09	-0.14	0.04	0.21	0.11	0.06	0.07	-0.06	-0.05
suck	-0.11	-0.21	0.01	-0.08	-0.22	-0.20	-0.04	-0.20	-0.08	0.01
ignore	-0.03	0.03	-0.03	0.07	0.06	0.07	-0.08	0.05	-0.02	-0.05
silly	0.02	0.02	-0.03	0.08	-0.04	-0.05	-0.05	-0.05	-0.16	0.03
loser	-0.03	0.14	0.09	0.20	0.09	0.10	0.01	0.11	-0.08	-0.01
ass	0.15	-0.06	0.01	0.06	-0.08	-0.14	0.12	-0.10	-0.16	0.03
foolish	-0.06	-0.01	-0.02	0.10	-0.03	0.06	-0.09	-0.11	-0.05	0.09
garbage	0.01	-0.09	0.13	0.11	-0.11	-0.02	-0.05	-0.05	-0.17	0.07
damn	0.06	-0.06	-0.01	0.03	-0.17	-0.17	0.12	0.00	-0.13	0.07
fool	0.04	0.03	-0.07	0.11	-0.10	-0.05	-0.03	-0.06	-0.05	0.09
ridiculous	-0.07	0.04	0.05	0.09	0.00	0.05	-0.09	-0.01	-0.14	0.01
moron	0.11	0.02	0.05	0.14	0.02	0.00	0.00	-0.01	-0.05	-0.02
crap	0.12	-0.10	0.15	0.14	-0.10	-0.03	0.06	-0.04	-0.18	0.17
dumb	0.24	0.11	0.22	0.23	0.10	0.11	0.21	0.04	-0.03	0.09
hypocrite	0.08	-0.05	-0.11	-0.04	-0.12	-0.22	0.00	-0.11	-0.07	-0.10
pathetic	0.07	-0.03	-0.01	-0.05	0.00	-0.07	-0.03	-0.04	-0.07	-0.06
stupid	0.10	0.09	0.11	0.20	0.01	0.08	0.07	0.03	-0.10	0.07
idiot	0.04	-0.07	-0.01	0.06	0.02	-0.04	-0.04	-0.10	-0.07	-0.09

	jews	jew	jewish	muslim	muslims	islam	christian	christians
jerk	-0.04	0.16	-0.06	0.13	0.11	0.06	0.08	0.10
assume	0.07	0.11	0.03	0.01	0.09	-0.01	0.16	0.17
suck	-0.32	-0.24	-0.33	-0.27	-0.31	-0.32	-0.30	-0.30
ignore	-0.03	-0.11	0.00	0.11	0.12	0.17	0.06	0.08
silly	-0.03	-0.04	-0.02	0.04	0.10	0.14	0.13	0.08
loser	-0.08	0.08	-0.25	0.00	0.04	-0.03	-0.22	-0.15
ass	-0.16	0.06	-0.26	-0.20	-0.20	-0.26	-0.23	-0.21
foolish	-0.04	-0.13	-0.16	-0.02	0.10	0.06	0.01	0.03
garbage	-0.05	-0.01	-0.27	-0.22	-0.10	-0.15	-0.21	-0.11
damn	-0.12	0.14	-0.23	-0.17	-0.16	-0.20	-0.13	-0.06
fool	0.00	0.14	-0.28	-0.11	0.07	0.01	-0.05	0.03
ridiculous	-0.05	-0.13	-0.08	0.03	0.07	0.04	0.04	0.00
moron	-0.07	0.15	-0.29	-0.08	-0.01	-0.09	-0.20	-0.10
crap	-0.05	0.07	-0.21	-0.12	-0.06	-0.07	-0.17	-0.06
dumb	-0.20	-0.11	-0.26	-0.02	0.01	-0.11	-0.10	-0.09
hypocrite	-0.10	0.24	-0.19	-0.06	-0.05	-0.03	0.07	0.04
pathetic	-0.12	-0.03	-0.11	0.03	-0.04	0.03	0.04	-0.05
stupid	-0.18	-0.11	-0.27	-0.02	0.04	-0.07	-0.10	-0.05
idiot	-0.10	0.16	-0.27	-0.07	-0.03	-0.08	-0.13	-0.07

Appendix III - Workflow Diagram