Hate Speech Classification with Interpretable Machine Learning Models

CSCE 704 Data Analytics for Cybersecurity Final Course Project

Warning: This paper contains language that may be considered offensive to some readers

Karthik Mohan Raj
Department of Computer Science
Texas A&M University
theraj@tamu.edu

Kiet Nguyen
Department of Computer Science
Texas A&M University
kiet.nguyen@tamu.edu

Himanshu Singh
Department of Computer Science
Texas A&M University
himanshu_singh@tamu.edu

Laren Spear
Department of Computer Science
Texas A&M University
larenspear@tamu.edu

***Abstract*—Modern social media allows users to post anything they want without revealing their identity. This allows those under oppressive regimes to also have access to these sites without fear of prosecution. On the other hand, many people use their anonymity as a shield to post obscene things. We develop an interpretable machine learning model that can classify online posts as either abuse or benign. We compare this model to a pre-trained state of the art model and evaluate its performance on benchmark datasets. The new model based on Latent Dirichlet Allocation and tree-based classifiers performed similarly to state of the art models, and respectably on an unseen dataset.**

I. Introduction

Abusive language is a common feature of the internet, despite the fact that websites and content providers make every effort to remove it. Given the scale of all internet speech, a manual review of all content posted by all users is not feasible, in addition to being grossly subjective. In the landmark 1995 court case Stratton Oakmont v. Prodigy, the New York Supreme Court held that online providers could be held liable for the speech of their users if they manually moderated content. On the other hand, in the 1991 Cubby v. CompuServe case in the same venue, it was ruled that CompuServe was not liable for the speech of their users because they did not have any hand in moderating content. In response to this perverse incentive to not moderate offensive content, Congress passed the Communications Decency Act in 1996. Section 230 of this act protects the online providers from being liable for the speech of their users, as long as providers made a "good faith effort" to remove objectionable content.

Any site which allows users to post original material runs the risk of being liable for the speech of their users if their moderation scheme is insufficient, which leads to the great need for effective detection of hateful content without constant human supervision.

II. Related Work

A variety of authors have attempted to tackle abusive language detection. HateBERT [1] is a re-trained BERT model for abusive language detection in English. The authors present the results of a detailed comparison between a general pre-trained language model and the abuse-inclined version obtained by retraining with posts from the banned communities on three English datasets for offensive, abusive language and hate speech detection tasks.

Among many papers and many services, the definition of hate speech itself is not settled. Hate speech could be broadly defined, as it is with HateBERT, as abusive language that demeans at least one person, while more often it is described to mean statements disparaging an attribute or characteristic of a group of people, such as race, religion, gender, or nationality. Spertus [2] studied abusive and hostile messages, which has informed future research in the field of detecting cyberbullying, but cyberbullying is not necessarily hate speech.

According to Fortuna [3], which synthesizes many definitions, hate speech is speech which is designed to incite violence or hate and intended to attack or diminish specific targets. MacAvaney [4] gives the example of "Jews are swine" being considered hate speech while "many Jews are lawyers" is not. Hate speech analysis requiring fact checking is a difficult task. As well, MacAvaney mentions that praising the KKK can be considered hate speech, while not containing any negative language. Hate speech detection models also suffer from the inability to detect when an otherwise hateful word is being reclaimed, being intentionally used by members who would otherwise be victims, and for this purpose as well as others Rottger et al. [5] developed a set of test cases for hate speech detection designed to test the limits of any system. Existing hate speech detection models tend to be overly sensitive to group identifiers, and generalize poorly to other datasets. Hosseini et al. [6] also point out that negated statements (e.g. "I don't hate Jews") are likely to be picked out as hate speech by many classifiers, including Google's Perspective API. The Perspective API is also weak to intentional deception, with Hosseini being able to drastically reduce toxicity scores by intentionally misspelling words.

An issue often plaguing hate speech datasets, when they exist, is the great imbalance between hateful speech and benign speech, which makes manual verification untenable. Founta et al [7] estimates that between 0.1% and 3% of all tweets are abusive, with only a fraction of those being hate speech specifically. There is a considerable, but not required, overlap between abusive language such as insults or profanity and hate speech determinants. A keyword-based search can work, as found in the HateBase dataset, but simply using a derogatory word is not by itself hate speech. MacAvaney [4] discusses that the phrase "merciless Indian savages" might appear to be hate speech, except that it is a quote from the Declaration of Independence. Furthermore, individuals are often aware that hate speech will be censored and will take measures to avoid censorship.

Davidson et al. [8] tested a variety of models, including logistic regression, naïve Bayes, decision trees, random forests, and linear SVMs, and found that logistic regression and linear SVM performed better than other models on their data, which was a subset of the Hatebase dataset which was manually coded by crowdsourcing. Crowdsourcing is an important technique, but Sap et al. [9] indicate that the racial biases of the humans marking the speech can then be encoded into the hate speech model, because some words that may be considered profane to white users may not be considered profane to black users, for example. Founta [7] also notes that while humans can generally consistently tell whether a text excerpt is appropriate or inappropriate, they often have trouble distinctly classifying it into a specific subcategory of inappropriate speech. This may or may not be important for the desired implementation, such as for a social network which will elect to either remove content or not.

For the purposes of this treatment of the subject, hate speech is considered to be offensive speech directed at a group of people in a protected class, including but not limited to race, gender, religion, or national origin.

Aside from these opaque BERT variations, Smith & Wiegand [10] note that n-gram features can be predictive as well, with additional augmenting features leading to better performance. Mehdad & Tetreault [11] found that character n-grams perform better than token n-grams, but they test only on that single feature.

III. Methodology

In order to get an insight on hate speech, we did some background analysis on the most common words used, the different classes that these words belong to and if the words were extreme or moderate. Additionally, we compared the performance of various dimension reduction algorithms — LDA, PCA, LSA — to understand groups of words that are highly correlated. This allows us to choose only those features that might help in improving the performance of the classifier. Knowing that users often try to avoid censors by replacing an extreme word with a moderate word, we used Latent Dirichlet Allocation (LDA) to evaluate the most prominent words that define a particular category of hate speech.

A. Dataset

We used hate speech datasets [13] obtained from Reddit and Gab. All the datasets were split into training and validation sets. The entire dataset contains around 50,000 different posts and, each belonging to one of the two categories of hate speech or benign speech. The Reddit dataset contained 5257 instances of hate speech and 11725 instances of benign speech, while the Gab dataset contained 14614 and 17397, respectively. There is an imbalance in the size of the classes in both datasets, given the relatively low occurrence of hate speech compared to all speech. Gab has a much higher proportion of hate speech labels due to its reputation for uncensored speech. Before computing any additional features, the text in the dataset was processed to remove any special characters, hashtags, user tags and URLs and was converted to the lowercase. Moreover, before passing it to the TF-IDF vectorizer discussed in section III-B3 we removed the stop words from the text and performed lemmatization.

B. Feature Selection

1) Elementary Textual Features: We initially started our analysis by computing the most common features used for basic text classification. These common features included word count, character count and word density. On top of that we added a number of features that captured information about the parts of speech tags of text. These features included noun count, verb count, adverb count, adjective count and pronoun count. Some other commonly used features like uppercase count and title count were dropped as the text was being converted to lowercase and these features would have lost their information content.

However, on computing the distribution of data points based on the elementary features, we found out that non of the features could be used as a reliable source for distinguishing hate speech from benign speech. As illustrated in Figs.1 & 2 there was a high degree of overlap between the two classes when considering these features individually. Thus we needed additional features that could better capture the differences between the two classes.

2) Latent Dirichlet Allocation: Humans can understand language based on context and are good at understanding topics based on the content that we read. This task, however, is hard for computers to perform.

LDA is a type of topic modeling which allows computers to classify which topics a document represents among multiple possible given topics. LDA is a statistical model that discovers the abstract topics that occur in a collection of documents. Apart from this main use, LDA is also used

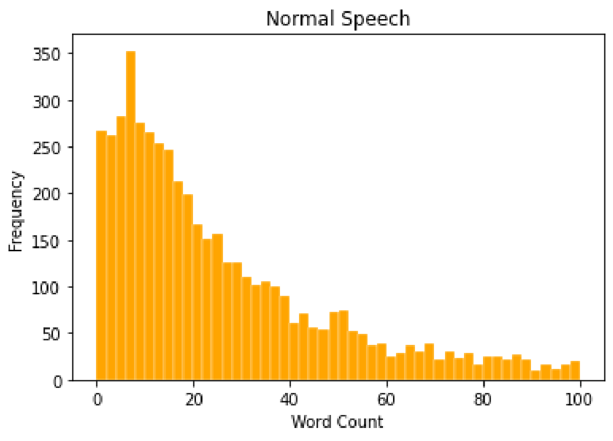

Fig. 1. Word count distribution in normal speech

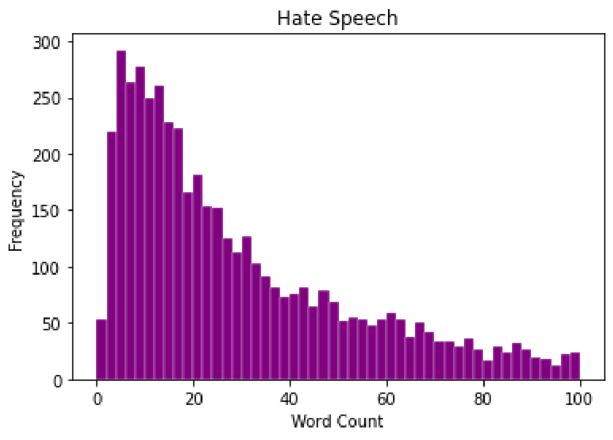

Fig. 2. Word count distribution in hate speech

for sentiment analysis, automatic harmonic analysis for music, object localization for images. The main idea behind LDA is that it suggests words carry strong semantic information and documents discussing similar topics will use a similar group of words. Latent topics are discovered by identifying groups of words in the corpus that frequently occur together within the corpus. The second assumption regards the structure of the documents themselves. LDA suggests that documents are probability distributions of related topics and topics are probability distributions of related words. Hence we will be looking at distributions of words across documents rather than frequencies of words.

TABLE I
FEATURES EXTRACTED FROM LDA

(1, '0.017*"fuck" + 0.014*"like" + 0.013*"retard" + 0.012*"women" + 0.010*"people" + 0.009*"men" + 0.008*"cunt" + 0.007*"time"')
(0, '0.014*"people" + 0.011*"think" + 0.011*"like" + 0.008*"say" + 0.006*"get" + 0.005*"want" + 0.005*"know" + 0.005*"guy"')

We divided the dataset into the two classes and performed LDA on both the subsets to get a list of the 10 most important words for each class. LDA also returns the probability of a text belonging to either of the classes. We plan to use this as a feature for our classifier. An example of the features extracted using the LDA in normal speech is given in table I.

3) TF-IDF: TF-IDF stands for Term Frequency - Inverse Document Frequency. It is a metric that gives us the relative importance of text tokens from the point of view of the individual documents and the entire corpus. It is one of the most widely used features for text classification and can produce the output vectors at different granularity like word level, character level and N-gram level. For our case we have used TF-IDF vectors for individual words and bi-grams. One thing to note about TF-IDF is that it returns a sparse matrix as output. It is not possible to combine this sparse matrix with the already existing feature set that we have computed. Moreover, using the dense matrix is a computation intensive task even on a GPU.

4) Combining Features: The TF-IDF vector is a sparse matrix and therefore cannot be used directly with the other set of features. Therefore we need a mechanism to combine the set of heterogeneous features which could then be used to train the model. For this we make use of available tools in Scikit-learn, namely ColumnTransformer and Make Pipelines.

ColumnTransformer applies a number of transformers on a subset of columns of a pandas dataframe. This estimator allows different columns or column subsets of the input dataset to be transformed separately and the features generated by each of the transformer will be concatenated to form a single feature space. It is useful for applying several feature extraction mechanisms and transformers on heterogeneous feature sets. We specifically apply this custom TF-IDF transformer on the token_text fields and the other elementary features are taken in their original form.

C. Training

From the classification point of view, we computed the TF-IDF vector for the training dataset while also including bi-grams and tri-grams in the vector. We compared the performance of numerous machine learning algorithms to detect hate speech. We plan to come up with an ensemble based classification model taking weighted votes of the best performing algorithms. Moreover, we plan to include word embeddings as a feature on top of the TF-IDF vector given that word embeddings like word2vec would better capture the contextual text information.

IV. EXPERIMENTS

Overall, we ran 4 experiments using various conventional models as well as testing our trained model with state-of-the-art research on more advanced models to test how well our trained model works on new data.

A. ML Models

Our main experiment involved running the processed data set of different features on 8 different classification models.

1) Linear Regression: Logistic regression predicts the relationships between the various features (independent variables) of the training samples (word count, character count, verb count, etc) and the classification label (dependent variable) by using a Sigmoid function to calculate the probabilities and then return the binary classification.

2) Naïve Bayes: Naïve Bayes classification technique assumes the element of independence among the predictors, which are the features in this case based on Bayes' Theorem. Specifically, it means each feature in the class exist in total independence with respect to any other features and there is no relation at all.

3) Decision Tree: A Decision Tree is a Supervised Machine Learning where the data is continuously split according to a certain parameter. There are two types of decision tree: regression and classification. We make use of Classification Decision Tree in our case. The tree is built recursively through a process called binary recursive partitioning in which data is iteratively split into partitions, which is then further split on each of the branches to make decisions.

4) Bagging: The bagging classifier is an ensemble meta-classifier that selects a random subset of the original data set to fit on a base model. It then aggregates all the individual predictions (either by voting or taking the mean) to arrive at a final prediction. This method is normally used to cut down variance of a black-box estimator such as decision tree through the random selection of the features.

After picking the random subset of data, each classifier is trained in parallel and each of the training set is trained independent from each other. This tends to reduce overfitting (variance) through the averaging or voting process, but can lead to additional bias.

We choose Random Forest as our bagging classifier for our data set. Random forest classification is a supervised learning algorithm. This classifier works by creating multiple decision trees through random selections of data samples. Each tree than makes its own predictions and all the results are combined together through means of voting to provide the final results.

5) Boosting: The boosting classifier is an ensemble learning model which aims to create a strong model out of many weak classifiers. It first builds a model from the training data which does not perform very well. On subsequent runs, the algorithm creates the next models that attempt to improve on the errors from previous models. This process is repeated until the final model can correctly predict all training samples or a threshold number of models are added.

For this, we try three different boosting classifiers:

a) AdaBoost: AdaBoost is considered the pioneer of other boosting algorithms that is first to be adapted to solve practical problems. It works on the principle of combining 'weak classifiers' into a single strong one. Each weak learner in Adaboost is a decision tree with a single split known as decision stump. The boosting process places more weight on instances that failed to be correctly classified and less weight on those correctly classified. It is a well-known algorithm for both classification and regression problems.

b) XGBoost: XGBoost (Extreme Gradient Boosting) is one of the most popular algorithms in machine learning for both classification and regression problems because of its good performance compared to other methods. It makes use of regularization to remove overfitting, followed by cross-validation. We used gradient boosting to compute errors in the previous model and then the leftover is added to make the final prediction.

c) LGBM: LightGBM is also a gradient boosting framework based on tree-based learning algorithms. It works well for distributed environments because it supports parallel and GPU learning. It is efficient with faster training speed compared to other boosting algorithms. It also provides better accuracy with lower memory usage while also being capable of handling large-scale data

B. Voting Classifier

One observation that we made while analyzing the predictions of different models was that different types of models were producing incorrect results for different subsets of the test set. Thus one possible improvement was to use a voting classifier that comprised of the best three models from our previous experiment. We tried running both the voting criteria, hard voting and soft voting. Hard voting makes use of the predicted class labels for majority voting whereas soft voting computes the argmax of the sums of the prediction probabilities. For our dataset, hard voting performed better when compared to soft voting.

C. Testing using Prior Work

We also tested our best performing model against the state-of-the-art HateXplain model in addition to testing the performance of our model on unseen data from the HateCheck dataset.

1) HateXplain: HateXplain model [12] is the state-of-the-art in abusive language detection but it also predicts the user's rationale with the post. This model uses a modified version of the BERT model which uses some complicated natural language processing techniques. However, due to RAM limitations on the Google Colab Jupyter Notebook, we could only pass in a very small subset of the entire dataset for testing. The results of this test are given in section V-A.

2) HateCheck:

V. Results

TABLE II
PERFORMANCE COMPARISON OF THE TESTED MODELS

Model	Accuracy	F1	AUC
Random Forest	0.9223	0.8989	0.9564
Decision Tree	0.9111	0.8865	0.9061
Logistic Regression	0.7770	0.6770	0.8218
Naïve Bayes	0.8228	0.7370	0.8761
AdaBoost	0.9269	0.9032	0.9506
XGBoost	0.9191	0.8898	0.9411
LGBM	0.9289	0.9061	0.9606
Voting	0.9286	0.9061	0.9621

A. Testing against the HateXplain model

We tested our LGBM model against the HateXplain model and got fairly close AUC scores — the HateXplain model had an AUC score of about 0.97 while our LGBM model had an AUC score of 0.91 on the small subset. This is a remarkably close score given that our model doesn't use NLP techniques. A plot of the ROC curves of our models and the HateXplain model is given in figure 3.

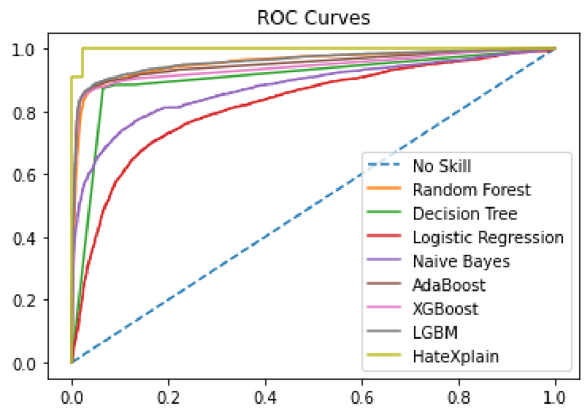

Fig. 3. ROC curves of all tested models

B. Testing on the HateCheck Dataset

VI. Conclusion

Our models do not use any deep neural networks or NLP techniques in our hate speech detector. As a result, we can more easily explain what features the classifier is weighing heavily. If such models were to be used by social media sites, users would have a more transparency as to what content is acceptable. While this could result in some users gaming the algorithm, any manually reviewed posts could be reintroduced to the dataset. Overall, our model performs fairly close to the state-of-the-art and could improve the explainability of automatic moderation decisions on social media sites.

VII. Future Work

Hate speech detection and removal is a constant cat-and-mouse game, with users constantly finding new, creative ways to evade censorship. A hate speech detection model unaffected by obfuscated words or carefully crafted euphemisms could radically change the landscape around internet discourse. This current model could potentially be applied to abusive speech of other domains and tested against a wider array of ambiguous real-world situational data, to combat cyberbullying or harassment. A multi-paradigm classifier to address different types of offensive language in different ways would be the endgame of these models.

References

[1] Tommaso Caselli, Valerio Basile, Jelena Mitrović, and Michael Granitzer. 2021. HateBERT: Retraining BERT for Abusive Language Detection in English. In Proceedings of the 5th Workshop on Online Abuse and Harms (WOAH 2021), Association for Computational Linguistics, Online, 17–25. DOI:doi.org/10.18653/v1/2021.woah-1.3

[2] Ellen Spertus. Smokey: Automatic Recognition of Hostile Messages. 8.

[3] Paula Fortuna and Sérgio Nunes. 2018. A Survey on Automatic Detection of Hate Speech in Text. ACM Comput. Surv. 51, 4, Article 85 (July 2019), 30 pages. DOI:https://doi.org/10.1145/3232676

[4] Sean MacAvaney, Hao-Ren Yao, Eugene Yang, Katina Russell, Nazli Goharian, and Ophir Frieder. 2019. Hate speech detection: Challenges and solutions. PLOS ONE 14, 8 (August 2019), e0221152. DOI:doi.org/10.1371/journal.pone.0221152

[5] Paul Röttger, Bertie Vidgen, Dong Nguyen, Zeerak Waseem, Helen Margetts, and Janet Pierrehumbert. 2021. HateCheck: Functional Tests for Hate Speech Detection Models. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Online, 41–58. DOI:doi.org/10.18653/v1/2021.acl-long.4

[6] Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving Google's Perspective API Built for Detecting Toxic Comments. arXiv:1702.08138 [cs] (February 2017). Retrieved December 5, 2021 from arxiv.org/abs/1702.08138

[7] Antigoni Maria Founta, Constantinos Djouvas, Despoina Chatzakou, Ilias Leontiadis, Jeremy Blackburn, Gianluca Stringhini, Athena Vakali, Michael Sirivianos, and Nicolas Kourtellis. 2018. Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior. In Twelfth International AAAI Conference on Web and Social Media.

[8] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. (2017). Automated Hate Speech Detection and the Problem of Offensive Language. In Eleventh International AAAI Conference on Web and Social Media.

[9] Maarten Sap, Dallas Card, Saadia Gabriel, Yejin Choi, and Noah A. Smith. 2019. The Risk of Racial Bias in Hate Speech Detection. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 1668–1678. DOI:doi.org/10.18653/v1/P19-1163

[10] Anna Schmidt and Michael Wiegand. 2017. A Survey on Hate Speech Detection using Natural Language Processing. In Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media, Association for Computational Linguistics, Valencia, Spain, 1–10. DOI:doi.org/10.18653/v1/W17-1101

[11] Yashar Mehdad and Joel Tetreault. 2016. Do Characters Abuse More Than Words? In Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue, Association for Computational Linguistics, Los Angeles, 299–303. DOI:doi.org/10.18653/v1/W16-3638]

[12] Punyajoy Saha (punyajoy) & Binny Mathew (binny-mathew), HateXplain, (2020), GitHub repository. github.com/hate-alert/HateXplain.

[13] Jing Quian, A Benchmark Dataset for Learning to Intervene in Online Hate Speech (2018), GitHub repository. https://github.com/jing-qian/A-Benchmark-Dataset-for-Learning-to-Intervene-in-Online-Hate-Speech.