Project Beyond Good and Evil

Dakota Owen (15200586)
dakota.owen@ucdconnect.ie

Tony Veale
tony.veale@gmail.com

Table of Contents

List of Figures

List of Important Abbreviations Used Within

API	Application Program Interface
BOW	Bag-of-words
NLP	Natural Language Processor
TFIDF	Term Frequency, Inverse Document Frequency
SVM	Support Vector Machines

Abstract

In the modern internet more organizations than ever are trying to crack down on hateful and offensive language to allow users to continue feeling welcome and comfortable in their online communities. Any community of significant size becomes forced to rely on automated systems to flag malevolent content and automatically remove it from public viewing, as manual removal of all hateful or offensive content is incredibly labour and time intensive for large public forums.

State of the art machine learning algorithms and standard blacklists quickly run into a large problem: accountability. Neither system can provide either the user they are blocking or a system administrator with an explanation why that content was flagged as problematic. Blacklists are crude solutions that can only explain which word was flagged, while current machine learning algorithms operate as 'black boxes', unable to provide any explanation for the results they obtain.

This report creates a modular lexicon capable of replacing blacklists and 'black box' algorithms. By utilizing a series of lists, using a Jaro similarity algorithm to reinforce detection, and identifying grammatical relations, this project is capable of taking a larger context into account like a machine learning algorithm, while also providing an explanation to the user for why their content was removed and cleaning messages of hate speech where needed.

Project Specification

Core Demands:

The student will research alternatives to the blacklist and make a strong argument for one in particular or suggest their own. The student will then implement this solution on a graded non-binary basis that assigns confidence scores and probabilities to suspect content and test the solution on real data sets.

Advanced Demands:

The student will explore and implement explanation generation in the context of a solution to give users an understanding of why a text was flagged as suspect/malevolent. You will explore issues of context-sensitivity and the possibilities of self-learning and extensibility.

1. Introduction

This report will contain language that some may find offensive and hateful. While this report clearly does not agree with the language that will be presented in our examples, it is our firm belief that analysis of hateful and offensive language cannot be done properly while simultaneously attempting to avoid it. The duty of those who wish to find effective ways to block and remove hate speech is to 'get down and dirty' so that others don't have to. Proceed at your own discretion.

Over the past decade the growth of online message boards and social media platforms have allowed for an incredible breadth of expression from anyone who wishes to be heard. Not all of this expression is pleasant, however. With the volume of content being produced by users continuing to grow companies can no longer manually enforce their terms of service and regulate what kind of content appears on their platforms, forcing them to turn to automated systems.

Make no mistake, it is vital that negative content be regulated online. Social media platforms are often considered responsible for the kind of content they host, especially when they are seen to be allowing and even profiting from hateful and aggressive language that can encourage more violence. Youtube has even garnered a "reputation for nurturing a hotbed of extremist views." (Cooper, 2019) In March of 2016 Microsoft caused a public uproar when their Twitter AI chatbot Tay, designed to learn from those it spoke with on Twitter, started repeating abusing content "including sharing hate speech, referring to black people with racial slurs; harassing prominent women gamers; and scrawling the word *swag* on pictures of Hitler's face." (Schlesinger et al. 2018)

This fiasco was possible because Microsoft didn't implement a blacklist for what Tay could say in response to Twitter users. Even a basic list like Darius Kazemi's WordFilter (Kazemi 2013) would have been able to prevent some of the abusive language Tay was repeating, although not all of it. A mere list is an extremely simple solution, and hate speech is a complicated topic which will require an equally complex solution to successfully stop.

Unfortunately, the problem of managing such a large amount of offensive and hateful language is only exacerbated by the muddled definition of what constitutes hate speech. Some people have thicker skin than others, and not all people will find the same thing to be offensive. For the purposes of this report, we will be using a similar classification to label hate speech as Davidson et al. (2017), defining hate speech as *language that is used to express hatred towards an individual or targeted group or is intended to be derogatory, humiliating, or to insult an individual or group*.

In the on-going fight to identify and manage this language, many different strategies have been explored in the last two decades including lists, sentiment analysis, deep learning techniques, and more. (See Gitari et al. 2015, Badjatiya et al. 2017) However, in their pursuit of accuracy these methods have missed a key factor: explanation. "On the side of interpretability, this [means] going from brittle rule-based but glass-box approaches to more robust opaque-box approaches, with supporting statistics but no explicit 'real reasons', to ultimately neural black-box approaches." (Holzinger et al. 2017)

To create an approach capable of explaining itself to users and system administrators this report proposes a plan to create a modular lexicon, paired with multiple filters to give a message a confidence classification in how likely it is an example of hate speech. The lexicon will be broken into three lists of varying intensity; black, grey, and blue, where blacklisted words are forbidden, grey-listed words are questionable, and blue-listed words is simple profanity. After checking to make sure the word isn't misspelled, checking the original

poster's history of being banned, and using some clever Java, courtesy of Stanford, to evaluate the linguistic features to take the context of the message into account, a 'meta-algorithm' checks the results of each step and judges if it is "Not Confident", "Confident", or "Very Confident" that the original message was hate speech. It can also mark the message as "Clean" if no red flags were raised. Such a project may be capable of shedding some light on how these powerful, yet unaccountable neural processes function and ease tensions users may have when it appears that they have had their free speech struck down by an invisible corporate hand.

2. Related Work and Ideas

Character n-grams and BOW (bag-of-words) are two extremely common tools in hate speech analysis. A n-gram is simply sequence of n items. In hate speech this comes in the form of n words or characters that form word stems. The BOW method on the other hand is the process of converting a sample text into a set (or bag) of words, ignoring grammar and word order. A simple blacklist functions by converting a piece of text into a BOW before checking if any of those words contain one of the listed n-grams. Lone blacklists implemented in this manner are sometimes capable of having comparable precision to more advanced methods, however, they suffer from very low recall (see Sood et al. 2011). For this reason, simple functions using n-grams are often paired with advanced techniques. Waseem and Hovy (2016) extracted character n-grams from the most frequent terms used in racist and sexist speech found on Twitter and evaluated a character n-gram based approach, also using the stated gender and location of the author of a tweet. Sadly, these added features did not provide a significant increase in accuracy, however.

It can be easy to avoid detection by a list of word n-grams by intentionally or unintentionally misspelling them. To help counter this, a strong method is to check for misspellings by calculating the Levenshtein edit distance between the misspelled word and the word on the blacklist. This edit distance measures differences between words in the given sample text and the listed word n-grams to account for replacing characters ("asshole" vs "@sshole"), inserting characters ("fag" vs "faaaag"), or deleting characters ("zipperhead" vs "ziperhead"). If the edit distance is under a given threshold (depending on the term's length) the misspelling is determined to be the listed word. Accounting for the Levenshtein edit distance significantly improves the recall of blacklists. (Sood et al. 2011)

Further improvements can be made by adding more information than just the n-grams themselves. Full lexicons add data to each word in a list rather than only listing the word itself. Razavi et al. (2010) built a lexicon from the ground up, giving each word a weight that measures how likely it is going to be hateful. Lexicons are also often formed by giving each word a classification. Burnap and Williams (2016) built a blended lexicon that classifies what specific type of hate speech it is (racist, anti-LGBT, etc) which not only enabled detecting general hate speech but classifying what type of hate speech was detected. These classifications assisted a supervised learning model in identifying context. Davidson et al. (2017) gathered Twitter messages according to a list gained from Hatebase.org and manually classified them as either hateful, offensive, or neither to allow supervised learning to differentiate between dangerous hate speech and merely offensive language.

Sentiment analysis is also a frequent tool in natural language processing that builds on top of a lexicon. Most sentiment analysis starts with a lexicon of word n-grams that are tagged with either a positive or negative polarity based on the sentiment they have out of context. Words like "good" and "trust" are given a positive polarity while words like "evil" and "distrust" are given a negative polarity. Sood et al. (2012b) uses sentiment analysis as "an alternative to 'eyeballing' and/or relying on user 'abuse' reportage" by finding negative comments in social media and bringing them directly before administrators for manual evaluation. In order to allow context to be considered in sentiment analysis, Wilson et al. (2005) evaluated linguistic features in each piece of text. These features allow things like the negation of a n-gram with negative polarity to be correctly identified as having a positive polarity, and their use significantly improved on prior results.

Thanks to tools like The Stanford Parser linguistic features are relatively easy to obtain and study. In order to manage hate speech invisibly to the user, Xu and Zhu (2010) used the parser to identify typed dependencies (de Marneffe and Manning, 2008) that were dependent on an offensive word and remove them. This allows what is left of the message to stay visible and be proper English while successfully removing the offensive part of the text. For example, "It is an aston martin you crying fag" becomes "It is an aston martin" without the recipient of the message being aware it ever had to be edited.

Supervised learning has become a popular form of analysing hateful language in recent years. Support Vector Machines (SVMs) were first explored for text analysis in Joachims (1998) and have since been used with much success Burnap and Williams (2015) used SVMs by transforming each word into a 'vector', making sure to ignore separate tenses and conjugations of a word. "For example, 'attacked,' 'attackers,' and 'attacking' can all be reduced to 'attack' ". Davidson et al. (2017) and Sood et al. (2012) both used SVMs in their analysis, with a combination of SVMs and the Levenshtein edit distance providing Sood et al. with their highest maximal F1 score.

Unsupervised learning has also been put to the task of language analysis. Le and Mikolov (2014) developed *Paragraph Vector,* a framework made of a multitude of word vectors that could be applied to variable-length texts, from a sentence to a 'paragraph'. Djuric et al. (2015) used this algorithm "and the same assumption of n-gram language models that words that are close in a sentence are also statistically more dependent" to outperform TFIDF and BOW approaches to identifying hate speech. Despite obtaining respectable results, Badjatiya et al. (2017) managed to improve on them by testing three different neural networks on a set of 16K annotated messages from Twitter using classifiers like SVMs, GBDTs (Gradient Boosted Decision Trees), and LSTMs (Long Short-Term Memory Networks) as the learning methods. The most successful one used a combination of these classifiers to obtain both precision and recall scores of 0.930, significantly outperforming prior attempts at identifying hate speech.

3. Data Considerations

The black and grey lists used by the algorithm this report proposes will be created by scanning the Hatebase.org API. Hatebase is "the world's largest structured repository of regionalized, multilingual hate speech" (Hatebase.org) and a source used by multiple studies

(i.e., Davidson et al. 2017, Nobata et al. 2016) to help identify the n-grams commonly used in hate speech. The n-grams this report works with are sequences of words considered hateful or profane, usually only one or two words in length (n=1 or n=2).

The fact that Hatebase allows user contributions also means that every time we access the API to update our lexicon, it will have to go through some filtering to determine which list a term will be added to. Each term labelled either *extremely* or *highly offensive* with over two hundred "sightings" will be added to the 'black' list, while each term labelled either *moderately* or *mildly offensive* with over two hundred "sightings" will be added to the 'grey' list. This is done to filter out low quality user input and ensure the lexicon only contains words that are widely agreed to be hateful. Each word on Hatebase.org also contains tags for who or what the word is used to offend (i.e. "Ethnicity" or "Disability").

Table 1 below shows some of the information Hatebase stores that we can use to filter for what will appear in our lexicon. Since users can contribute to the database, the language we use in the lexicon will be kept up to date as new hateful words become commonplace.

Table 1: Hatebase n-grams

n-gram	Language	Tags	Sightings	Strength
camel jockey	English	Ethnicity	3,129	Highly offensive
eyetie	English	Nationality, Ethnicity	216	Highly offensive
jap	English	Nationality, Ethnicity	416	Moderately offensive
zipperhead	English	Ethnicity	1109	Extremely offensive

A final 'blue' list will be created containing common profane terms from *noswearing.com*, the same list used by Sood et al. (2012) in their work to improve profanity detection. The terms obtained from *noswearing.com* will then be checked against our already created lists to ensure we do not add duplicates. This will prevent users from being penalized twice for the same word.

The data hosted on GitHub (see Davidson et al. 2017) containing over 25,000 sample messages obtained from public Twitter messages will be used as the testing data for the algorithm. This data has been annotated as offensive speech, hate speech, or neither through crowdsourcing, the results of which were manually reviewed by Davidson et al. This thorough checking of the data ensures that little cleaning will be necessary for our filters used by the meta-algorithm to be effective. The only check we will be forced to run the data through is to convert some Unicode decimal codes back into their original characters (i.e. &#8221 appearing in place of ", the right double quotation mark). Table 2 below contains some examples of the data collected by Davidson et al. to demonstrate how it is annotated. The class for each tweet (0 for Hate Speech, 1 for Offensive, 2 for Neither) is determined by the majority vote of the crowdsourcing workers that reviewed the sample texts. Twenty percent of this data will be put aside to be used during our last test for the final report.

Table 2: Hate Speech Detection Tweet Samples

ID	HS	O	N	Class	Sample Tweet
30	0	3	0	1	"I txt my old bitch my new bitch pussy wetter"
443	2	1	0	0	"Don't worry about the nigga you see, worry about the nigga you DON'T see… Dat's da nigga fuckin yo bitch."
1650	0	0	3	2	“@Yankees: #Yankees and Red Sox are scoreless as we play in the 2nd inning.” THE BAMBINO IS WATCHING!
11079	3	0	0	0	I never had any friends the entire time I lived in #Virginia. No one was cool to me. I hope those white trash Southerners all die.

The lexicon and testing data will be made public on Github at the end of development to ensure the environment used to produce our final results remains archived and to allow others to check our results. Peer review is an important process to continue accurate research into this complex problem.

4. Approach Outline

The bulk of the lexicon proposed in this report will be the 'black' and 'grey' lists created by pulling key words and phrases from Hatebase.org using an academic license for their API. While the black list will contain n-grams that are not appropriate in any context (i.e. "coon") the grey list will contain n-grams that may be used in either innocent or hateful ways depending on their context and words that are not offensive enough to belong to the black list (i.e. "coloured"). The tags each n-gram has on Hatebase will also be recorded in the lexicon.

The blue list will function as a supplementary filter when determining if messages containing terms in the black or grey lists are hateful. If profanity is related to a term appearing in our lexicon the message is much more likely to be hateful. For example, while some people are attempting to reclaim hateful slurs like "negro" and do not use it in a hateful context, the combination of an offensive n-gram and profanity is almost certainly not going to be used to 'reclaim' the word, or even in a neutral context.

When the lexicon is completed, test messages will be scanned for the presence of any of these offensive n-grams. After an initial check for any direct matches to our lexicon, a second pass will be made while calculating the Levenshtein edit distance between words in the given message and the lexicon (Levenshtein 1966). If the edit distance is below a threshold (determined by word length), the test message will be flagged as having the un-edited word. Using the Levenshtein edit distance ensures such tactics do not allow users to easily bypass the extensive lists forming our lexicon.

The Stanford parser will be used to find typed dependencies (de Marneffe and Manning, 2008) within a message. For the purposes of this report we will be focusing on the *amod* (adjectival modifier) and the *neg* (negation modifier). These typed dependencies will allow us to view some context in each message much more easily than a full-blown sentiment

analysis learning algorithm. The *amod* dependency will be used while checking the blue list in order to confirm that the detected profanity is being directed at the n-gram detected from the black or grey list. "Fucking japs! I say we nuke em again!!" has the amod(japs, fucking) dependency, showing how the profanity was used to further increase the offensive nature of the slur used. The *neg* dependency will be used to see if someone is working against the slurs they are forced to quote (much like this report itself). "Never gook. Never, ever, say gook." Has the neg(gook, never) and neg(say, never) dependencies. In such a case the use of a word from the lexicon will be negated in the meta-algorithm.

Additionally, our records will be checked for if the author of the given sample has had previous messages flagged as problematic. After the confidence level that a message is hateful has been reached, the author and confidence level of the given message will be recorded for future reference. Review of hate speech data sets have shown that a large majority of messages flagged as hateful were written by a vocal minority of authors. If the writer of a sample has a history of hate speech with a high confidence level, it is likely that a new, more questionable sample is also being used in a hateful manner.

These individual steps come together to be evaluated in our meta-algorithm, the centre piece of the report. Each message will receive a classification of Clean, Not Confident, Confident, or Very Confident. Firstly, as one might suspect any n-gram on the blacklist will not be allowed in any circumstances. Whether a message directly contains these n-grams or the Levenshtein edit distance is short enough to confirm their presence, these are n-grams that are either don't have alternative, non-offensive meanings, or are so offensive that we can be very confident that the message should be given at minimum a "Confident" rating and blocked. Naturally if the message also contains a second confirmed offensive variable (such as multiple n-grams on the blacklist) the message will be given a "Very Confident" rating instead.

The presence of a single offensive n-gram (directly or a match from its edit distance) from the grey or blue lists will be given a "Not Confident" classification however, therefore not getting the message immediately blocked as they are less severe n-grams and/or have non-offensive meanings. Instead for one of these messages to gain a minimum of a "Confident" rating and be blocked a second confirmed offensive variable will be required. For example: the presence of a second n-gram from the grey list in a single message, or a n-gram from the blue list used in relation to the n-gram from the grey list, the user having a history of hate speech in our records. Three or more offensive variables paired with a grey n-gram will give it a "Very Confident" rating before being blocked.

The exact requirements for each layer will require some testing to fine tune to ensure maximum accuracy. These requirements are also designed to be easily edited so the meta-algorithm can be edited for a given domain, recall, or precision goal. For example, administrators using this algorithm who are more concerned about false positives than false negatives can easily increase the confidence threshold even higher by requiring more variables to all be pointing at a message being offensive.

Finally, two responses will be generated for any blocked message: one for the user and one for the administrator. Due to the progress of XAI (explainable artificial intelligence) deep learning on text only being "modest" (Holzinger et al. 2017) over the years, it still extremely difficult to build a reliable XAI for qualitative content. Due to this limitation, the responses generated from the meta-algorithm will simply follow a template. A state machine can easily choose the appropriate template for the given combination of variables. For example:

This Message has been blocked.

The phrase "Fuck japs" in your message is not allowed on this service. This is your second offense.

5. Project Work Plan

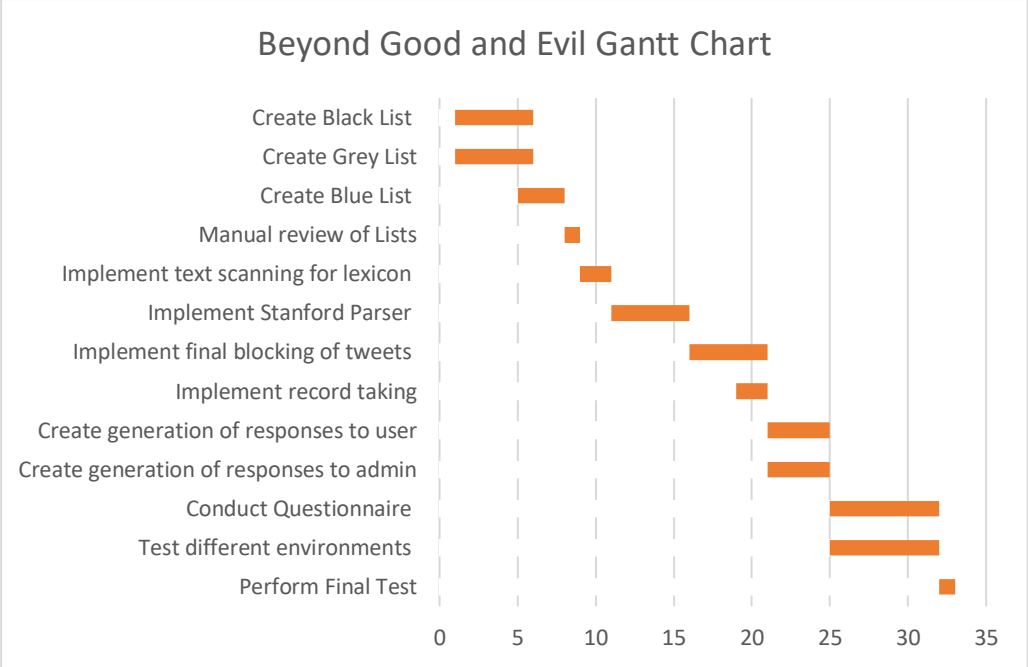
The first task during the project will be to compile the lexicon that to be used. As both the black and grey lists will be taken through the Hatebase.org API their progress will be tied to each other. The blue list however is obtained from *noswearing.com*, so its download can begin once the Hatebase API is in progress downloading the other two lists. Once both downloads are finished the lexicon can go through some brief manual review to ensure the lexicon's quality.

After the lexicon is complete it will be easy to implement a basic text scanner to check for the presence of any recorded n-grams. This will be done in Java since the Stanford Parser is already in Java. Once reading in text is functioning work to implement the Stanford Parser can begin. It is suspected this will be one of the more difficult parts of the project as the UD (Universal Dependencies) module is complex.

Once the Stanford Parser is properly functioning work on the meta-algorithm can take place, followed by setting up a record. While the record is one of the filters that will be used in the meta-algorithm, the record also requires the information from all other filters to be accessible in order to keep track of the context for a given ban as much as possible. The meta-algorithm itself can be easily edited to include the records once they are in place.

With all filters and the blocking in place, accurate explanations can be created using a state machine choosing the appropriate template for a given context. While straightforward, this step is expected to take a few days. At this point a baseline for the project has been created, and testing can take place both in the form of editing the meta-algorithm and in performing reviews on the quality of the generated responses. This part of the planned schedule is the most flexible, as it is hard to predict how much testing will be required to find an optimal F1 score and quality explanation templates. Once testing is over the meta algorithm will be run against the twenty percent messages explicitly left untouched. This final run will create the data used for the final report.

Figure 1:



Due to its nature, the project's ability to classify hate speech will be quite empirical and easy to evaluate. Each test environment will have its precision, recall, F1 scores, and overall accuracy. The environment with the highest F1 score will then be used on the twenty percent of GitHub data that was set aside for the final report. The quality and accuracy of responses generated by the report are much more qualitative in nature. As such, it will be necessary to conduct a questionnaire with volunteers or crowd sourced manpower that evaluates the automatic responses.

6. Data and Context

A downloaded copy of the terms and details in the Hatebase API was used to build both the black and grey list in the lexicon for this project. This allowed for the detection of hateful terms in the analysed messages and provided important details for each term such as a definition, level of severity, and which type of minority the term targets. I gained an Academic License from Hatebase to access their API for free, however this license restricts me from redistributing their data or using it for commercial purposes. However, the ability to have access to the same API which is used in hate speech research done in universities such as Harvard, Stanford, and Princeton outweigh these limitations.

The data contained in the Hatebase API is frequently updated to contain new hateful terms as they appear and update old ones with new details. These updates would allow us to continue to use the Hatebase API to detect developing hate speech in new sample data.

A list of "amplifiers", or words used to exaggerate the negative effects of these terms, was also created for the purposes of this project. While largely made of a list provided by noswearing.com containing profanities such as "asshole", the list was also custom-modified to better suit our objectives by ensuring it had no conflicts with the Hatebase API and adding words such as 'kill' which while not swearing, do add a negative impact when used as adjectives or verbs in connection to hateful terms.

The message samples that were used to test our detection were taken from the work Davidson et al. (2017) posted for public use on GitHub. The decision to use this data instead of alternatives that could be found online was made for multiple reasons. Most importantly this data provides an exceptionally large spread of examples to test against. The text of these messages are also recorded directly, as opposed to many other sets of Twitter data publicly available which only include links to tweets which have since been removed, making them unusable.

Unfortunately, this repository is no longer actively curated. This means the data is static and will not be updated to reflect new forms of hate speech, which would have allowed the program to be tested against developing hate language online.

The Jaro edit distance is a formula to calculate similarity which was used to detect terms from the lexicon in the sample messages. This ensured accidentally or intentionally misspelled terms in test messages were still properly identified as the original term contained in the lexicon, and made detection of hate terms more robust than using a simple list while simultaneously having fewer false positives than using n-grams to detect hate language.

A Java implementation of lexicalized parsers made available by the Stanford Natural Language Processing Group was used while analysing each tweet to determine some context in the text. These parsers provided the universal dependencies for each sentence present which helped determine if the presence of hateful words were being used in a negative manner or were unrelated to the offensive content despite their presence in the sentence. Since this project is written in Python notebooks, it uses the Python interface for the Stanford Parser built by Stefanie Tellex.

7. Core Contribution

7.1 Data Shapes

This project was designed to demonstrate the strengths of using a modular lexicon in a field being flooded with natural learning algorithms. While we have seen some natural learning algorithms can achieve very impressive accuracy scores, it is nearly impossible to determine how they came to their determined results. The modular lexicon however enabled this project to provide reasoning for why it labelled any given message.

The largest modules of this lexicon were created by using the Hatebase API. At the time of writing this report, Hatebase contains 1,530 different hate terms that will be used as one module of our lexicon and allow us to detect hate speech. Each of these terms have a unique ID for identification purposes and Boolean flags to state if the term is ambiguous and if it targets a person's nationality, ethnicity, religion, gender, sexual orientation, disability, or class. Many terms also include defined meanings and ratings on a scale of zero to one hundred for their average offensiveness when used. This project split the Hatebase data on a term's ambiguity to easily detect and flag the most extreme hate speech in messages. Terms that were considered unambiguously hate speech formed the blacklist and terms that were considered ambiguously hate speech formed the grey list in the lexicon.

The third module of the lexicon used in this project is a list of 295 words which are often used to amplify hatred. When the Stanford Parser detects dependencies between these words and hate terms from the black or grey list it provides a clear indication that the terms are being used in an even more hateful manner than average.

This lexicon was used to scan the 24,782 sample tweets provided by Davidson et al. (2017). Additionally, each of these samples have been presented to a minimum of three CrowdFlower workers who were asked to mark the message as hate speech, offensive, or neither. This previous work being attached to the messages provided us with real human opinions to use as a baseline for comparison to the results of the lexicon and algorithm this project has created.

7.2 Preparation

The devil is in the details though. Before it could be used, the tweet samples had to be thoroughly cleaned so they could be more easily processed by the parser. This involved removing ascii codes for emojis which were used in many of the messages, removing Twitter "@" tags, as well as some equals symbols and new line characters which were unique to this data set. Once cleaned, the sample messages were separated into three sets for easier testing: a set containing only messages which were universally deemed hateful, a set of messages which were deemed universally clean, and the database in its entirety.

Once an access key for the Hatebase API was obtained, I was able to download it to a local copy for easy use. This involved sending a ping to the API to gain authorization and a unique session token before sending a query for the number of pages the API has for English results. At the time of writing this report Hatebase contained sixteen pages of results for English hate terms. Once this max page count was obtained, a loop through the pages was made to record the results from each. The response for each page from the API came in the form of JSON containing that page's terms. After the JSON results of each page were recorded, it was converted to a CSV file and transposed to ensure each term had its own row.

Before the Stanford Parser could be used, an environment variable, "CORENLP_HOME", had to be defined in Python. Then custom functions were defined to prevent unnecessary repetition and allow for increased simplicity when reading the bulk of the code. The recorded API, list of amplifiers, and sample messages were all read into the notebook before defining global variables for the Jupyter notebook. These variables included a lower and higher level of offensiveness a message must reach before it starts being considered hateful, a similarity minimum for the Jaro edit distance to recognize misspelled terms in the message, and a list of dependency types that will not prove helpful to determine if terms are being used in a hateful manner.

The lower offensiveness cutoff was set in the same one to one hundred scale used to describe the average offensiveness of terms in the Hatebase data. The higher offensive limit was set to three times the lower to be capable of registering the difference between a single offensive term and a multitude of them. The Jaro global variable however represented the minimum percent of similarity a term from the sample message needed with one of the terms in the lexicon to be considered identical. These variables were designed to allow easy access

and editing to core features to allow for customizing the algorithm to be used in different settings or samples.

7.3 Message Analysis

With all necessary preparations completed the first message was fed to the parser. The parser split the message into its given sentences and annotated each sentence with the dependencies deemed appropriate by the Stanford Parser. Some of these dependencies, such as punct("before", "."), were useless to us. This dependency marks the connection between words in the sentence and basic punctuation. Simple format dependencies could not help determine how hateful terms were being used in a sentence. Others like amod("useless", "gook") however were exactly what we are looking for. The 'amod' dependency revealed the relationship between a subject and an adjective, showing hate terms from the lexicon which were used in an even more hateful manner than average.

Once the parser did its job the message was scanned for any hate terms contained in the lexicon. Since the parser only marked dependencies between word pairs, not phrases, two-word phrases contained in the hate terms module of the lexicon were the first to be scanned for. Any phrases detected with a Jaro similarity equal to or greater than the defined cut-off had their data from Hatebase retrieved using one of the custom functions mentioned above and sent to a "termInfo" data frame for later analysis. Since one-word terms from the lexicon could have been given dependencies from the Stanford Parser, any terms detected within the Jaro edit distance limit were 'tagged' and saved in a new data frame to have their dependencies examined.

For each unambiguously hateful term from the Hatebase API that was detected within the defined Jaro edit distance, the label of the message was increased by two. This was done to ensure that under no circumstances would words that are very clearly, always, used in a hateful manner would escape detection. Words of this extreme nature could never be allowed to bypass the filters established here unimpeded.

Once all negative terms from the message were successfully identified, the full data pertaining to them was retrieved from the recorded Hatebase data. Notably, this included the average offensive value for each tagged word. If one of the terms was not given an average offensiveness score by Hatebase, it was temporarily given the average offensive value of a hate term from Hatebase. If the term was ambiguous, it was given the average offensive value of all ambiguous terms, if the term was unambiguous it is given the average offensive value of all unambiguous terms.

With any terms from the lexicon in the message recorded it became possible to directly check the dependencies the Stanford Parser had produced. First any dependencies not tied to words that had been 'tagged' were ignored, followed by dependencies which had been deemed unhelpful to determining the hate term's use in the sentence. What remained were the useful dependencies connected to terms in the lexicon which were 'flagged'. The words in question, position of the dependency, type, and whether the source or the target of the dependency was modifying the 'tagged' term were all added to the "flags" data frame using another custom function for further review.

Table 3: "Flags" Data Frame Containing Dependencies

Sentence	Edge	Source	Source Words	Target	Target Words	Dependencies	Modifier
1	9	11	retarded	10	also	advmod	target words
1	8	11	retarded	9	're	auxpass	target words
1	7	11	retarded	8	you	nsubjpass	target words
1	6	6	fag	2	only	cc:preconj	target words
1	4	6	fag	11	retarded	conj	target words
1	3	6	fag	7	but	cc	target words
1	2	6	fag	5	a	det	target words
1	1	6	fag	4	you	nsubj	target words
1	0	6	fag	3	are	cop	target words

These flags were immediately reviewed for connections between hate terms obtained from Hatebase and the list of amplifiers. If a dependency centred around a tagged hate word did not contain any problematic terms (e.g. the first row of Table 3), it was ignored. When a dependency containing both a hate term and an amplifier was detected, the offensive value of that hate term was set to double, showing the extra malicious way that term was being used. With this careful examination of dependencies the parser produced, we ensured that the mere presence of dependencies was not all that was required to demonstrate higher than average hateful content. Only dependencies that demonstrated increased malicious intent were used.

After all offensive values and dependencies had been evaluated the final label was determined. If the sum of average offensive values, after being modified by any relevant dependencies, was greater than or equal to the previously established lower offensiveness cut off than the label increased by one. If the sum of average offensive values was greater than or equal to the higher offensive limit the label increased by two instead. This goes on top of the label increase caused by the detection of any unambiguous terms detected during the original scan of the message but did not include the offensive value of unambiguous hate terms to avoid counting them twice. Combined, the label increases from the sum of offensive values and detection of unambiguous terms determined the message's final label.

7.4 Message Generation

With the analysis of the message contents completed, the next step was to prepare the warning, or lack thereof, which would be displayed to the user. For terms which have a stated

meaning this was relatively easy. Most definitions could be retrieved directly from the Hatebase data with minimal interference. Some however were given multiple definitions, or exceedingly long ones which did not suit the purpose of trying to create a short, concise message for the benefit of users. For these the definition was split and either only the first sentence or the first definition for those with multiple was used. This was done by checking for brackets which were indicative of multiple definitions for a single term and checking for periods contained in extended definitions and splitting the definition on those delimiters so only the first part could be selected.

Things became more complicated when a tagged word was missing a definition altogether, however. In such an event we were forced to look to the other relevant Boolean flags attached to the tagged word and generate as appropriate of a description as possible. This was done by checking which categories in the recorded Hatebase data the term belonged to (e.g. “is_about_nationality”, “is_about_ethnicity”, etc) and generating an appropriate message response from those flags which could be presented in readable English. If a term was said to belong to multiple categories, then all categories that applied were included in this generated description. With all term descriptions successfully retrieved or created, they were ready to be shown to the user.

Finally, all necessary work to provide a readable response for the original message to the user was finished. From here, exactly what that response contained depended on the tweet’s final label. Since the labels worked in ascending order to represent an increasing amount of hate speech that had been detected, the lower the label the less intervention was needed for the original message. If the message currently being tested received a label of zero, then the user saw a notification telling them “This message is clean of hate speech.” and the original message was shown to the user.

If the analysis gave the message a label of one, a warning appeared using the descriptions created earlier to warn the user of the type of hate speech the message may have contained before the message was printed unaltered for the user to view at their own discretion.

With a label of two some of the message needed to be censored before a cleaned version of the original message could be presented. This was done by going through the original message and replacing any occurrences of hate speech with the generated descriptions. This was done to still maintain some sense in the message. However, an argument could be made to merely replace any hate terms detected with asterisks to remove as much hatred from the message as possible.

For any message that was given a label of three or higher a similar warning notification as for a label of one or two was presented to the user. Unlike a label of two however, for hate speech this grievous the entire message was blocked, and the user was informed that no part of the message could be displayed due to the extremely hateful content it contained.

Figure 2 Sample Result for Label of 2:

WARNING: Parts of this message have been censored due to hate speech.

these "woman" like "black person" that spend money not talk bout it

This process repeated for each message in the sample data until all sample messages were successfully examined.

8. Evaluation

Davidson's data was fundamentally different from the results this report created, which caused a disconnect when evaluating the new results and comparing them to the sample data. As stated above, the samples provided by Davidson et al. (2017) split tweets into one of three categories: hate speech, offensive speech, and clean speech. This is clearly not the same style of results this algorithm created to both detect and grade the severity of hate speech.

The key to being able to compare this reports labels to a simple classification model came in the form of the included individual votes that comprise each classification decision in the original Davidson data. As described above, each sample was evaluated by a minimum of three people. These votes were not always unanimous, however. Opinions about what is and what is not hateful differs. This crucial input helped determine if a lesser label which still detected hate speech would be accurate.

If a message received zero votes that it was hate speech it should clearly have been given a label of zero by the algorithm created in this project. If all votes on a message claimed it was hate speech, then it should have been given a label of three or higher by the algorithm. The key difference was samples which did not receive such a uniform vote distribution. For example, messages which received only three out of four votes that it was hate speech, but not four out of four votes, it should clearly have been given a label of two to demonstrate the confirmed presence of hate speech, but not hate speech so egregious that it must be completely blocked for anyone to see.

Out of the 24,782 different messages which were tested, we can split the ratio of hate votes into four categories to match the four labels this project proposes. Messages which received zero hate votes should be assigned a label of zero, messages which receive `one to fifty percent hate votes should be assigned a label of one, messages which receive a fifty-one to ninety-nine percent should be assigned a label of two, and finally messages which only received votes declaring it was hate speech should be assigned a label of three or greater.

Checking the counts for each of these splits of ratios from the Davidson samples revealed 19,789 tweets that received zero votes indicating they were hate speech, 3,571 tweets where up to half of the reviewers rated it as hate speech, 1,159 tweets that had over half their voters consider them hateful, and 263 tweets which were uniformly considered hate speech.

The data provided by Davidson et al. (2017) was tested with a Jaro similarity cutoff of 95%, a first offensive cut-off of 75, and a second offensive cut-off of 225. The first test perfectly labelled 15,222 of the samples, which provided an accuracy of 61.14%. In this set of results 7,669 samples (or 30.95%) were given minorly inaccurate labels. These were results such as giving clean messages a warning, or messages a minority of reviewers believed was hate speech a completely clean label. Only 1,893 tweets (or 7.64%) were given labels that grossly mischaracterized the actual content of the message according to human review. Of those

messages which were seriously mislabelled 1,473 were given higher labels than necessary (5.94% of all messages) and 420 were given lower labels than required (1.69% of all messages). After comparing perfect labels to inaccurate labels of any degree, the first test provided a 67.91% level of precision and a recall score of 86.53%.

Table 4: First Test Results

Hate Ratio	0.0	0.01-0.50	0.51-0.99	1
Label				
0	13,629	1,506	295	54
1	4,923	1,131	335	71
2	1,024	736	433	109
3	152	126	60	9
4+	61	72	38	20

A second test was performed with increased levels required for terms to be deemed offensive at 90 and 270, respectively. With these changed parameters 17,272 tweets were perfectly labelled, giving this test an accuracy of 69.70%. Minor mislabels occurred with 5,692 (or 22.97%) samples, meaning the test applied reasonable labels to 92.67% of all tweets in the original data. The second test performed slightly worse than the original though in as it allowed additional some hate speech to go undetected. Of all examples, 488 (1.70% of all messages) were given lower labels than should be applied. The second test managed to obtain a precision score of 77.89% and a recall score of 86.88%, clearly demonstrating the increased offensive cut offs significantly brought down the number of messages falsely labelled as hate speech, while not significantly impacting the detection of tweets which were actually hate speech.

Table 5: Second Test Results

Hate Ratio	0.0	0.01-0.50	0.51-0.99	1
Label				
0	15,850	1,726	362	65
1	2,786	947	282	61
2	1,004	756	450	112
3	90	77	30	6
4+	59	65	35	19

Now, why do Table 4 and Table 5 both include a row for messages which received a label of 4 or higher? This is since only one or two, extremely offensive terms could cause a message to be given a label of three. This is by design. Under no circumstances do we want to allow the word “nigger”. Even when used in a positive manner words like this can be extremely

distressing for some people to read. However, for a message to be given a label of four or higher it must have contained at least two unambiguous hate terms. While still clearly incorrect results as deemed by human reviewers, such messages contain so many pieces of what usually constitutes hate speech that they would be extremely difficult for most automated systems to appropriately label.

Naturally, responses were much harder to evaluate with any empirical measurements. To gauge their effectiveness, I sent a set of sample results from this project to three volunteers asking for feedback on their construction, accuracy, and quality. All three informed me that the user responses were well crafted, however a common complaint focused on some of the definitions for common terms. Definitions like “human female” for “bitch” felt obtuse. While future work could fine tune some of the definitions provided by Hatebase for frequently used hate terms, the time constraints on this project made such work infeasible.

Although the test with a higher tolerance for hate speech did obtain a noticeably higher accuracy, it also allowed more hate speech to go undetected. Depending on the implementor’s preference, this may or may not be desirable. While a perfect accuracy is clearly the ultimate goal, the reality is that some messages will always slip through the cracks in an automated system. When that happens, a choice must be made: to allow some users guilty of hate speech to walk free, or to falsely accuse innocent users of hate speech.

9. Future Work

There is still a lot of work to be done to improve hate detection, however. Crowd work to analyse the tens of thousands of results the lexicon returned could provide greater insight to the results we have already obtained, as naturally it is infeasible for me to check nearly 25,000 results of the lexicon with only myself and friends who are willing to volunteer. Further testing of the lexicon and work this project presents could also yield improved results.

In specific work could be done to add another module to the lexicon. Like the “amplifiers”, a list of “neutralizers” could be made to help prevent false positive results. This list would consist of words that are used by someone who is clearly not being hateful; “love”, “caring”, etc. If one of these “neutralizers” is found being used as an adjective or verb as determined by the parser, it could be indicative that the hate term is in fact being used in a positive manner.

10. Summary and Conclusions

Continuing research into this topic is incredibly important, even though it is by its very nature unpleasant to do. Without automated systems hateful, aggressive speech would spread across online forums like a virus. It is crucial that this work is done to reign in and prevent atmospheres like this capable of causing real world danger to many people from spreading.

Once complete, this modular algorithm will be capable of answering some of the questions still hovering over hate speech detection, telling us what algorithms of the past are recognizing, and where we can improve on them further. The degree of confidence generated

from checking the results of each module of our algorithm will also allow future research to easily refine the algorithm for use in their given domain.

These same explanations will also help prevent frustration in users who may feel that a silent algorithm appears to work at random. Making sure people do not end up feeling like they are being unjustly silenced is essential. Automated systems like the one this report plans to develop should be seen as reasonable adjudicators of what speech is problematic, not as an invisible iron fist quashing views its corporate overlords disagree with.

Some may claim that automatic censorship is a serious threat to free speech, but it is particularly important to remember your free speech in both the United States and the European Union is not protected from any and all interference. The European Convention on Human Rights and the first amendment of the constitution of the United States protect citizens from having their free speech from being abridged by the government, but neither of them stop private corporations from regulating what you are able to say on the platforms they provide. This means these social media platforms that are seen almost as public services have the implicit right to enforce their terms of service, terms every user agrees to. Facebook's terms of service explicitly state: "We do not allow hate speech on Facebook because it creates an environment of intimidation and exclusion and in some cases may promote real-world violence", with similar statements from websites like Twitter.

Acknowledgements

Thank you to the many, many people referenced in this report and the years of hard work they have put into researching the best ways to identify and combat hate speech online. This report was only possible because it stands on the shoulders of giants.

This report is entirely my own work. I have ensured that all outside material is appropriately quoted and credited in the bibliography below.

References

Badjatiya, Pinkesh, Shashank Gupta, Manish Gupta, and Vasudeva Varma. 2017. "Deep Learning for Hate Speech Detection in Tweets"

Burnap, Pete and Matthew L. Williams. 2015. "Cyber Hate Speech on Twitter: An Application of Machine Classification and Statistical Modeling for Policy and Decision Making"

Burnap, Pete and Matthew L. Williams. 2016. "Us and them: identifying cyber hate on twitter across multiple protected characteristics"

Davidson, Thomas, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. "Automated Hate Speech Detection and the Problem of Offensive Language."

Djuric, Nemanja, Jing Zhou, Robin Morris, Mihajlo Grbovic, Vladan Radosavlijevic, and Narayan Bhamidipati. 2015. "Hate Speech Detection with Comment Embeddings"

Holzinger, Andreas, Chris Biemann, Constantinos S. Pattichis, and Douglas B. Kell. 2017. "What do we need to build explainable AI systems for the medical domain?"

Joachims, Thorsten. 1998. “Text Categorization with Support Vector Machines: Learning with Many Relevant Features”

Kazemi, Darius 2013, WordFilter. *https://github.com/dariusk/wordfilter*

Le, Quoc and Tomas Mikolov. 2014. “Distributed Representations of Sentences and Documents”

Levenshtein, V.I. 1966. “Binary Codes Capable of Correcting Deletions, Insertions, and Reversals”

de Marneffe, Marie-Catherine and Christopher D. Manning. 2008. “The Stanford typed dependencies representation”

Nobata, Chikashi, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. 2016. “Abusive Language Detection in Online User Content”

Razavi, Amir H., Diana Inkpen, Sasha Uritsky, and Stan Matwin. 2010. “Offensive language detection using multi-level classification”

Schlesinger, Ari, Alex S. Taylor, and Kenton O’Hara. 2018. “Let’s Talk about Race: Identity, Chatbots, and AI”

Sood, Sara Owsley, Elizabeth Churchill, and Judd Antin. 2012. “Using Crowdsourcing to Improve Profanity Detection”

Sood, Sara Owsley. Elizabeth F. Churchill, and Judd Antin. 2012b. “Automatic identification of personal insults on social news sites”

Sood, Sara Owsley, Elizabeth Churchill, and Judd Antin. 2011. “Automatic identification of personal insults on social news sites”

The Stanford Parser: A statistical parser. *http://nlp.stanford.edu/software/lex-parser.shtml*

Waseem, Zeerak, Dirk Hovy. 2016. “Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter”

Wilson, Theresa, Janyce Wiebe, and Paul Hoffmann. 2005. “Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis”

Xu, Zhi, Sencun Zhu. 2010. “Filtering Offensive Language in Online Communities using Grammatical Relations”