Determining Code Words in Euphemistic Hate Speech Using Word Embedding Networks

Rijul Magu
Department of Computer Science
University of Rochester
Rochester, New York
rmagu2@cs.rochester.edu

Jiebo Luo
Department of Computer Science
University of Rochester
Rochester, New York
jluo@cs.rochester.edu

Abstract

While analysis of online explicit abusive language detection has lately seen an ever-increasing focus, implicit abuse detection remains a largely unexplored space. We carry out a study on a subcategory of implicit hate: euphemistic hate speech. We propose a method to assist in identifying unknown euphemisms (or code words) given a set of hateful tweets containing a known code word. Our approach leverages word embeddings and network analysis (through centrality measures and community detection) in a manner that can be generalized to identify euphemisms across contexts- not just hate speech.

1 Introduction

Euphemisms, as an instrument to mask intent, have long been used throughout history. For example, a rich lexicon of drug slang has developed over time, with entire communities subscribing to benign sounding words that allude to names of drugs, intake practices or other stakeholders (such as dealers). In these instances, the primary motive is often to get a message across while evading detection from authorities. The main obstacle in these cases is the inability to identify code words without gaining access to an explicit mapping between the words and their latent meanings. Often, these words are already embedded in such common parlance that they cannot be spotted without placing the correct context.

Notably, in late 2016, a number of users across social media platforms and internet forums (particularly 4chan) began a movement called 'Operation Google', a direct retaliation to Google announcing the creation of tools for the automated moderation of toxic content. Essentially, the idea was to create code words for communities within the context of hate speech, so that they would not be detected by such systems. The movement branched out to a number of social media platforms, and in particular Twitter (Magu et al., 2017). The complete list of all code words is presented in Table 1.

Recent work has begun to emerge on studying this instance of euphemistic hate speech. However, they deal largely with the impact of code words in hate speech. Our work instead focuses on moving towards automating the *discovery* of code words. The objective is to significantly decrease, or even eliminate the need for human effort required in manually evaluating what words could be euphemisms. That said, the solutions and processes we describe go beyond finding use within the hate speech context. While some parts of the preprocessing stages do benefit from being aware about hate speech associated issues, the fundamental approach is general enough to be applied to assist in extracting code words in any context that may be found in natural language.

Additionally, our technique, revolving around studying the structures of the word co-occurrence networks that emerge from instances of coded hate speech, also lies within a predominantly unexplored space. We find that the centralities of word nodes within such networks indirectly provide crucial hints about the context within which they arise.

2 Abusive Language

The past few years have witnessed an increased focus on abusive language (particularly hate speech) detection, with a variety of different approaches and applications to a diverse set of contexts. These have ranged from classification of hateful tweets using bag-of-words models and typed dependencies (Burnap and Williams, 2015), to using semantic structure of sentences to the study of tar-

Code Word	Actual Word
Google	Black
Yahoo	Mexican
Skype	Jew
Bing	Chinese
Skittle	Muslim
Butterfly	Gay
Fishbucket	Lesbian
Durden	Transsexual
Car Salesman	Liberal
A leppo	Conservative
Pepe	Alt-Right

Table 1: Some common code words.

gets of hate (Silva et al., 2016). Others have used related methods to study phenomena closely associated to hate speech such as cyberbullying (Raisi and Huang, 2017) and offensive language in general. From an application perspective, Magu et al. (2018) create models to predict the extent of hateful content in the comments section of news sources, using article titles and body information as input, indicating a relationship between the entities. Davidson et al. (2017), a recent work on hate speech detection, notes the difference between hate speech and offensive language.

Notably, Waseem et al. (2017) provide a typology of abusive language, categorizing abusive language across two dimensions: 1) target of abuse and 2) degree of abuse being explicit. In terms of target, they distinguish *directed* hate (hatred towards an individual. Example: 'Go kill yourself') and *generalized* hate (hatred towards a recognized group. Example: 'So an 11 year old n***er girl killed herself over my tweets? thats another n***er off the streets!!'). With regard to this, a number of studies have been carried out that measure and study this effect, particularly within the context of social media. For instance, ElSherief et al. (2018) deal with analyzing the characteristics of hate speech instigators and targets on Twitter.

For the second dimension, the authors differentiate between *explicit* (containing 'language that which is unambiguous in its potential to be abusive'. Examples could be language containing racial slurs) and *implicit* (containing 'language that does not immediately imply or denote abuse'. Example: 'Gas the skypes' (Magu et al., 2017)) hate. The authors discuss the role of *context* which is needed to correctly identify hate. As such, they touch upon a fundamental mode of expression that displays implicit hate: euphemistic hate speech. Euphemistic hate speech stands separate from other forms of implicit hate speech (namely micro-aggressions) because in truth, they are often direct toxic attacks as opposed to veiled or context dependent ones. They are implicit because they make use of clever word substitutions in language to avoid detection. Even a classifier trained on a hate speech dataset containing instances of code words, can fail to identify hateful content if new code words (either intentionally or otherwise) start being used by the community. Indeed, currently, the only available option to recognize such instances manually, which is often inefficient and burdensome. Therefore, this motivates the exploration of more automated methods to detect euphemisms within the hateful context.

2.1 Euphemistic Hate Speech

To clearly define the problem of euphemistic hate speech, we start by looking at the general definition of hate speech as given by Davidson et al. (2017). They define hate speech as:

> *"language that is used to expresses hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group."*

In this sense, we note that euphemistic offensive language also qualifies as hate speech because it targets communities based on race, religion and sexual orientation. As a result, *hate speech that relies on intentional word substitutions to evade detection* can be considered to be **euphemistic hate speech**.

As discussed earlier, one of the most prominent uses of euphemistic hate speech came as a result of Operation Google, ever since the movement started in 2016. For instance Hine et al. (2016) studied the usage patterns and behavior of a community on 4chan (where the movement first started) and Twitter. Finally, Magu et al. (2017) analyzed euphemistic hate speech on Twitter, creating a tool to automate the process of segregating tweets containing code words (eg. 'gas the skypes') from those containing code words, but benign in nature (eg. 'I like to make skype calls').

3 Dataset

We collected approximately 200,000 English tweets containing the code words using the Twitter Streaming API. The time range was crucial for us as we wanted to study the use of code words as they first propped up. Therefore, our extracted tweets were concentrated between September 2016 (when the code first gained prominence) up until November 2016 slightly after the election season in the United States of America. Next, we needed to select a single code word that could be used as a starting seed word for our analysis. Effectively, the aim was to retrieve other code words knowing only one code word beforehand. We chose the code word 'Skypes' (used in place of 'Jews') and manually extracted 850 random tweets that used it in a hateful context (for example saying 'Gas the Skypes' but not 'I like to Skype call him everyday'). From this point on, all of our analysis is carried out on this dataset of hate tweets containing the word 'Skype'. Note that the set of tweets can and do contain other code words also (example: *'If welfare state is a given it must go towards our own who needs. No Skypes, googles, or yahoos'*). As a proof of concept, we showcase the entire process starting with 'skypes' but this can be extended to other starting code words as well. While we recognize the value of a comprehensive study of the methodology across the entire spectrum of combinations (retrieving all other codes given any randomly selected initial word), it went beyond the exploratory nature of our work.

Importantly, as an artifact of the time range of the data (late 2016), we do not expect there to be any previously unidentified code words (or at least not ones used extensively by the community during that time) within the dataset. Therefore, to validate the working of our methodology, from this point on, we instead assume we have no knowledge of the existence of any other code word beside 'skype'. Indeed, our method does not incorporate or exploit any hints it may derive from knowing any of the other code words beforehand.

4 Baseline

To asses how well our method performed, we needed to establish a baseline method. Currently, the simplest way to identify code words in natural language would be to manually sift through a series of tweets of users belonging to known hate communities. Clearly, this is an arduous and ineffective process. A reasonable approach is to rank all the words in the corpus (on the basis of some metric) so that higher ranked words have a greater chance of being code words than those lower on the list. For a small time frame like ours and without any prior information (note there might not be any qualifying indicator for a specific word to be a code word), a good bet would be to use word frequency as the metric. This is based on the idea that rarely used words would be unlikely candidates for code words. This method forms our baseline. As a result, the base ranks of our code words were the ranks of the words on this baseline list. Thus, we can quantify the performance of the method we develop by comparing evaluating the rank improvement of code words against their base ranks. Baseline ranks can be found in Table1.

Code Word	**Rank**
Google	10
Yahoo	67
Bing	195
Skittle	23
Butterfly	459
Fishbucket	998
Durden	471
Car Salesman	232
A leppo	667
Pepe	137

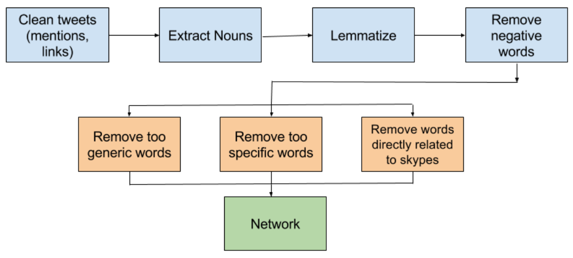
Table 2: Baseline ranks of code words.

5 Preprocessing

We made use of a set of strong assumptions about hate code usage in natural language to inform our data processing decisions (visualized through Figure 1). It is worth considering that while these assumptions help immensely in narrowing the space of possibilities, they are general enough to be applied to any scenario involving coded speech. Additionally, it is important to note that the numbering of these assumptions does not indicate a sequential order of processing (in terms of trimming down words). This is because some of these steps benefit from having the entire, unaltered tweet available to them. Pruning is carried out after determinations are made for each word through all of the assumption steps. Thes assumptions are as follows:

- A1: The code words are nouns.
- A2: They are not words normally considered negative.

- A3: They are not either extremely generic nor are they very specific



Clean tweets (mentions, links)
Extract Nouns
Lemmatize
Remove negative words
Remove too generic words
Remove too specific words
Remove words directly related to skypes
Network

Figure 1: Flow chart depicting the pre-processing steps.

It is safe to assume that the words would be nouns, in the context they are written within tweets. It is to be noted that all code words are used in place of references to real communities, which in turn are necessarily nouns. As a result, although the references themselves are replaced, the syntactic structures of the tweets remain intact. Hence, the code words, like the communities themselves would be detected as nouns by syntactic parsing. Therefore we use part-of-speech tagging to extract the set of all nouns from the entire list of words tokens.

Next, we lemmatize these tokens to bring them to their base form. This is of critical importance because we would like to construct a network at some point, and without standardization, we would be left with multiple duplicate nodes and unclear results. Additionally, carrying out lemmatization at this stage considerably reduces the space of possibilities early on, allowing for faster computation during later stages.

We move onto our next assumption A2, namely 1code words are not words normally considered negative'. Using words that are already negative (for example, 'terrorist', 'vermin' etc) defeats the purpose of using code words to begin with, asides from immensely confusing readers. For instance, consider replacing 'Jews' with 'terrorists' in the sentence 'Kill the Jews'. In the new sentence, 'Kill the terrorists', even if 'terrorists' had been adopted as a code word, it would become impossible for the author to convey whether they meant killing of Jews or actually terrorists. Hence, we attempt to remove all negative words at this stage. We do this by importing the list of negative words assembled by Hu and Liu (2004) and removing words from our lemmatized set that match those in this list.

The third assumption A3 ('They are not either extremely specific nor are they very generic') is checked for next. Clearly, like in the previous case, it is very confusing if the code words are too broad. For instance, in each of the following cases 'These people are disgusting', 'These men are disgusting', 'Something is disgusting', the potential code words 'people', 'men' and 'something' have alternate meanings that fit well within the context, but are too generic to be used as code words. As a result, we discard all such instances. Similarly, sample 'This Hillary is stupid!' in place of 'This Jew is stupid'. It is hard to decipher whether the author refers to a particular individual ('Hillary' the name) or the target community ('Hillary' as a euphemism for Jew). In these cases, the words are too specific to be useful as code words. Therefore, we use a mix of named entity recognition and manual pruning to remove these tokens.

Next, it is imperative to also exclude words that are directly related to the known code word. For example, we need to remove instances like 'Jew', 'Gas', 'Holocaust' etc when using the dataset for skypes because these already exist within the context of anti-semitic hate speech and cannot be used as code words by definition. Yet they may affect our analysis because of their expectedly high common usage. This is the only part of the filtering process that requires some basic domain knowledge of the problem.

As a final step, we discard the word skypes itself, since it occurs in every single tweet and provides no additional information.

6 Detecting Euphemistic Hate Speech

The main idea behind how our system works, is that *code words are an abnormality within the context defined by hateful users*. Words like 'google', 'skype', 'yahoo' etc are not expected to occur regularly in tweets aiming to attack groups. For example, the occurrence of 'skypes' and 'googles' is an aberration with respect to the surrounding words within the following:

"fucking googles and skypes kill em all"

6.1 News Data Word Embeddings

As a result, we can exploit this effect by representing all of our word tokens by their word embeddings that have been pre-trained on a regular, not necessarily hateful context, and evaluating how dissimilar they are with respect to each other. For our purpose we use word2vec (Mikolov et al., 2013) embeddings trained on a Google news dataset (built on google news articles with over a 100 million words) and find the pairwise cosine distances for all words. Essentially, we can expect the code words to be further apart than other words which belong to the hate speech context. A subgraph, with a few representative nodes can be seen in Figure 2.

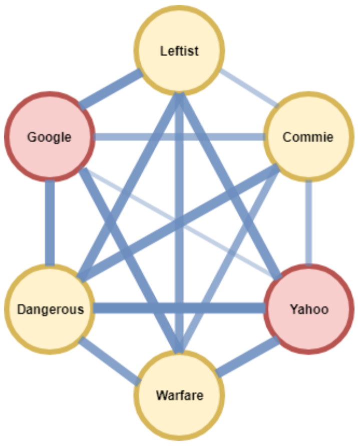


Figure 2: A subgraph of the network composed of some sample nodes. The red nodes are code words whereas the yellow nodes are non-code words. The diagram is for representation only and these labels are not provided beforehand.

The primary limitation to this approach is that some of the code words do not have representations within this pre-trained model and might be missed entirely. These words are 'leppos', 'fishbuckets' and 'durdens'.

6.2 Hate Network

Since 1) the pairwise relationships between words are now quantified and 2) that these relationships cannot be assumed to be independent of each other, an intuitive way to study this structure would be to model it as a network (seen in Figure 3). The degree distribution is graphically represented in Figure 4. Specifically, we created a network where each node represented a particular word, and the edges denoted the cosine distance between their respective word embeddings. In addition, we decided to leverage the fact that a sizable number of words did not co-occur together in tweets, thus providing us with additional information about context. As a result, we pruned all edges where the connected words did not ever occur together in any tweet. Some characteristics of the graph such as number of edges and average degree are given in Table 3.

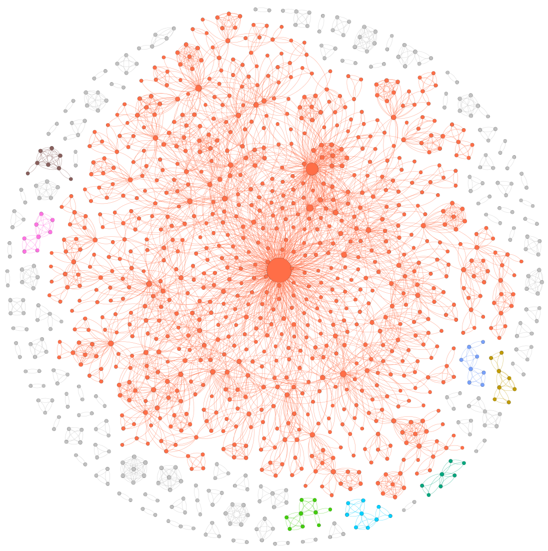


Figure 3: The hate word network. Note that the nodes are colored by the connected component they belong to. Those belonging to components insignificant in size are colored grey. The giant component is colored orange. Also, the size of the nodes corresponds to the degree. The largest node in the center is 'Google'.

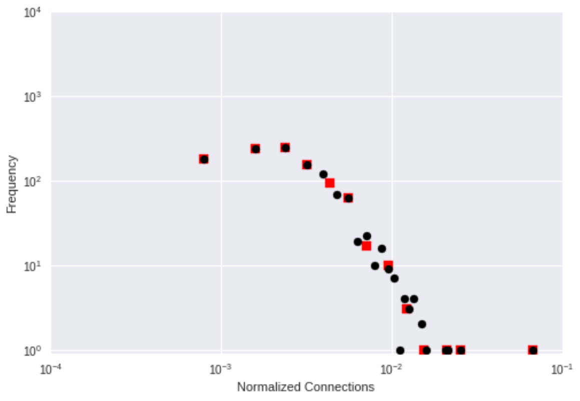


Figure 4: The degree distribution (in black) and log-binned degree distribution (in red) of the network.

The network displays characteristics typical of a complex network. For instance,the network shows the small world effect (Watts and Strogatz, 1998). We evaluated the value of the cluster coefficient (C) and mean shortest path length (L) for the network (as can be seen in Table 3) and then found the average of those metrics (C_r and L_r) across an ensemble of 100 random graphs with the same degree distribution. This allowed us to calculate the value of the small-world coefficient (σ), which is found using equation 1. The value was noted to

Metric	Value
Number of nodes	1129
Number of edges	2188
Average Degree	3.88
Clustering coefficient	0.76
Mean shortest path length	4.53

Table 3: Hate network properties. Clustering coefficient and mean shortest path length are for giant component.

be 20.46 which is much greater than the required threshold of 1, for a graph to be expected to show the small world effect.

$$\sigma = \frac{C/C_r}{L/L_r} \tag{1}$$

6.3 Word Ranks

As noted by Magu et al. (2017), the code words tend to extensively co-occur with each other. There are a number of possible explanations for this effect. First, as people warm up to a new code, they are incentivized to use as many different words as possible so that the code gains traction amongst their followers. Using too few code words within tweets (at the very beginning of adoption) could lead to those words being overlooked, or be treated as mistakes. Second, the alt-right (the primary users of the code) tend to display blanket hate towards a number of different communities across spectrum of race, religion and sexuality. Therefore, their tweets often simultaneously target a number of different groups, a pattern which in turn is replicated in their code word usage.

In this circumstance, we decided to use eigenvector centralities (Bonacich, 1972a,b) words as our ranking metric. Intuitively, this was an appropriate choice. Eigenvector centrality estimating techniques attribute higher centrality values to nodes that are connected to other important nodes. This was very relevant to our context since we know that the existence of certain words (which are hard to pre-determine) within hate speech have an effect on whether surrounding words could be code words or not. The ranks of code words are shown in Table 4. As we can see all codewords (barring those without word embeddings) substantially move up the ranking order, when compared to the baseline model (with a mean jump of 134 positions). Interestingly, the improvement for ‘butterfly’ was not as dramatic (in terms of its final rank) likely because it might have occurred with words considerably different than those connected to the other code words.

Code Word	EC	IoB
Google	1	9
Yahoo	3	64
Bing	22	173
Skittle	2	21
Butterfly	215	244
Fishbucket	-	-
Durden	-	-
Car Salesman	4	228
A leppo	-	-
Pepe	30	107

Table 4: Centrality rank of code words in comparison to baseline. EC: Eigenvector Centrality Rank. IoB: Improvement over Baseline. Note that the centrality values and improvements for some words is absent since they did not have a word-embedding within the pre-trained model. As a result, they were removed at an earlier stage.

6.4 Candidate Cluster

There is one major issue with the above discussed approach: manually moving down a ranked list to discover code words can be cumbersome. Additionally, there is no bound or limit till which we can expect to cover all or most code words. Therefore, perhaps a more useful technique could be to suggest a group or cluster of candidate words which have a high chance of being code words. Community detection analysis on the network is hence a viable approach.

First, since the graph is disconnected, we focus on the largest component (or the giant component) of the graph to further carry out our analyses (visualization in Figure 3). Since we know that the code words are likely to be closely connected to each other, we expect to find cliques (sets of completely connected subgraphs) within the network. Therefore, we applied the clique percolation method (Palla et al., 2005) on our graph to achieve overlapping communities of words. Essentially, the clique percolation method works by constructing communities from k-cliques (where k is a natural number). A community is obtained by finding the maximal union of k-cliques that are within reach from each other through a sequence

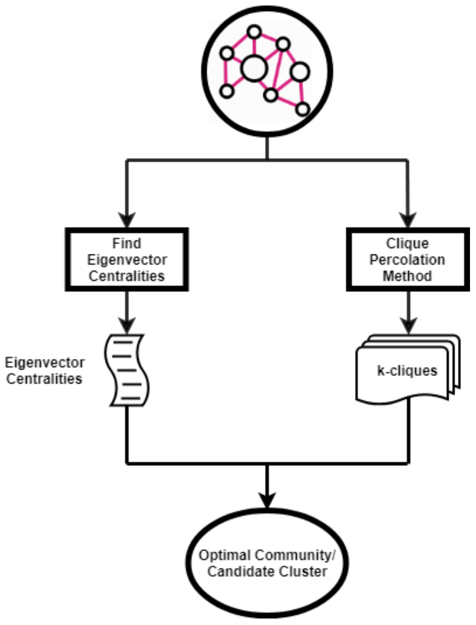

Find Eigenvector Centralities
Clique Percolation Method
Eigenvector Centralities
k-cliques
Optimal Community/ Candidate Cluster

Figure 5: Finding the optimal community. This is a high level visualization of the approach used to determine a set of candidate nodes using community detection analysis.

of adjacent k-cliques. Here, adjacent k-cliques are those cliques which have k-1 nodes in common. Since we do not know the optimal value of k, we carried out separate analyses for each value of k starting from k=4 to k=8, which is the largest possible value such that no (k+1) cliques exist in the graph. Judging by the extremely high clustering coefficient of the graph, there are an immensely large number of triangles in the graph. As a result, the algorithm is non-trivially affected because essentially, all or most nodes are grouped together into a single community. This is why values of k less than 4 were not considered.

Next, we needed to find the optimal community out of all the possible communities we have achieved for each value of k. A common approach is to simply select the largest community, which implies the community with the most number of nodes (largest length). This approach assumes each node to have the same weight (of value 1). However, since we know that eigenvector centralities serve as a useful indicator to finding the code words, we can weigh each node instead by its eigenvector centrality. As a result, in place of simply finding the community with the highest length, we summed over the eigenvector centralities of all nodes in every community and returned the one with the highest value. The resulting community was: [‘blind’, **‘skittle’**, **‘google’**, ‘don’, ‘commie’, **‘car’**, **‘salesman’**, ‘youtube’, **‘yahoo’**, **‘bings’**]. Figure 5 depicts a high level representation of the process.

The cluster is extremely tight- consisting of only 10 members. Yet, it contains all the code words that are present within the word2vec pretrained model except ‘pepe’ and ‘butterfly’ with only a few outliers. While some of these are likely noise, the occurrence of terms like ‘commie’ is explainable.The left is frequently targeted by the alt-right, the most common users of the euphemisms. Therefore, these users seem to often group ‘commies’ (or communists) together with the other target communities (which have euphemisms) within their tweets. This is why they form part of the clique that was uncovered through our analysis. For example:

> ”In theory, I agree, but with a congress filled with skypes , yahoos, googles, and commies, @realDonaldTrump won’t get anything done”

Thus, other than providing us with a set of strong candidates for euphemisms, this approach also reveals useful information about the posting patterns of this community of hateful users.

7 Limitations

There are some limitations that we would like to work on in the future. The major drawback is that we need one starting seed code word to find others. We would like to be able to identify code words in a manner in which we require no prior information about any code words, even at the cost of robustness. Second, it would be useful if we could iteratively improve our performance. For example, if we are able to identify a second code word using our technique, the suggestions for the next candidates should adapt appropriately to generate better results. Finally, we wish to achieve the word embeddings on larger, varied datasets so that when they are used to find cosine distances, some important words are not automatically missed out on.

8 Conclusions

We discussed the problem of euphemistic hate speech and how we could transform the challenge of finding unknown code words into a network science problem. We presented insights that can be derived during the preprocessing stage (such as the code words being nouns and neither too generic nor too specific). Finally, we showed how by using cosine distances between word embeddings could be coupled with analyzing the structure of the resulting network to achieve likely candidates for code words.

Our approach can be used to detect code words not only within the context of hate speech, but anywhere else where a community may feel the need to use euphemisms within natural language.

References

Phillip Bonacich. 1972a. Factoring and weighting approaches to status scores and clique identification. *Journal of mathematical sociology*, 2(1):113–120.

Phillip Bonacich. 1972b. Technique for analyzing overlapping memberships. *Sociological methodology*, 4:176–185.

Pete Burnap and Matthew L Williams. 2015. Cyber hate speech on twitter: An application of machine classification and statistical modeling for policy and decision making. *Policy & Internet*, 7(2):223–242.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. *arXiv preprint arXiv:1703.04009*.

Mai ElSherief, Shirin Nilizadeh, Dana Nguyen, Giovanni Vigna, and Elizabeth Belding. 2018. Peer to peer hate: Hate speech instigators and their targets. *arXiv preprint arXiv:1804.04649*.

Gabriel Emile Hine, Jeremiah Onaolapo, Emiliano De Cristofaro, Nicolas Kourtellis, Ilias Leontiadis, Riginos Samaras, Gianluca Stringhini, and Jeremy Blackburn. 2016. A longitudinal measurement study of 4chan's politically incorrect forum and its effect on the web. *arXiv preprint arXiv:1610.03452*.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM.

Rijul Magu, Nabil Hossain, and Henry Kautz. 2018. Analyzing uncivil speech provocation and implicit topics in online political news. *arXiv preprint arXiv:1807.10882*.

Rijul Magu, Kshitij Joshi, and Jiebo Luo. 2017. Detecting the hate code on social media. In *Proceedings of the eleventh International AAAI Conference on Web and Social Media*, page 608.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814.

Elaheh Raisi and Bert Huang. 2017. Cyberbullying detection with weakly supervised machine learning. In *Proceedings of the IEEE/ACM International Conference on Social Networks Analysis and Mining*.

Leandro Silva, Mainack Mondal, Denzil Correa, Fabrício Benevenuto, and Ingmar Weber. 2016. Analyzing the targets of hate in online social media. *arXiv preprint arXiv:1603.07709*.

Zeerak Waseem, Thomas Davidson, Dana Warmsley, and Ingmar Weber. 2017. Understanding abuse: A typology of abusive language detection subtasks. *arXiv preprint arXiv:1705.09899*.

Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440.