BT4222 Final Project
Toxic Comment Classifier

Group 5

Leow Shyan Shin Vincent (A0155833B)
Lim Jin Ming, Jeremy Denzel (A0172720M)
Yeo Zhang Yi (A0122060A)

Introduction

Posting comments on online platforms has become a common way to exercise one's right to freedom of expression in the web. However, this basic right is currently undermined by some users who post toxic comments that can be defined as inconsiderate, disrespectful, or preposterous. Such comments could potentially result in other users to avoid discussing on online platforms. A subtask of sentiment analysis is toxic comment classification. In the following report, our group aims to present a classification model for toxic comments and how our model could be integrated into online applications for automated moderation.

Motivation

Maintaining healthy conversations on the internet has been a very important task for online administrators, especially so when users are free to express their thoughts on different online avenues such as social media platforms and forums. Thoughts of being abused and harassed online has prevented people from expressing themselves and seek different opinions online (Duggan, 2017). With the vast number of toxic comments and content posted online, even large companies like Facebook[1] are struggling to effectively facilitate conversations, leading to many other communities to limit or completely shut down user comments.

In recent years, toxic content on online platforms have resulted in unpleasant and upsetting experience for a lot of people, particularly women and minorities. According to a survey conducted by Amnesty International[2], 7.1% of tweets sent to the women in the study were problematic or abusive. By simple mathematical calculation, this would amount to a staggering 1.1 million tweets mentioning 778 women across the year, which also translates to one every 30 seconds. Women of colour, (black, Asian, Latinx and mixed-race women) were 34% more likely to be mentioned in abusive or problematic tweets than white women. The proliferation of toxic comments is becoming an increasing problem for such platforms as they result in lower number of users who engage in discussions which directly affects the number of visitors to their platform.

With the rise in the number of political campaigning or even agitation being distributed online, serious and safe platforms to discuss political topics and news in general has become more important. Additionally, new regulations in particular European nations have been legislated which requires administrators to delete toxic content in less than 72 hours[3]. From a legal and business standpoint, our examples above have painted a picture which illustrates the importance and urgency of combating against toxicity online. From a linguist's perspective, toxic classification can be challenging for the machine due to lexical and semantic ambiguity. Hence, to help us solve this complex issue we have to leverage on machine learning models, based on the basic foundations of natural language processing concepts, which will allow us to automatically classify toxic comments.

---

[1] https://www.vice.com/en_us/article/xwk9zd/how-facebook-content-moderation-works

[2] https://www.amnesty.org/en/latest/news/2018/12/crowdsourced-twitter-study-reveals-shocking-scale-of-online-abuse-against-women/

[3] https://www.bbc.com/news/technology-42510868

Classes of Toxicity

Toxicity comes in various forms and shapes, generalising every toxic comments into one class may be too broad and may not give additional information on why certain comments are deemed toxic. Furthermore, breaking toxic comments into smaller classes would allow us to conduct a more in-depth exploratory data analysis. In the following, we discuss a classification scheme which consists of five different toxicity classes.

Obscene Language

Example: “That rule is bullshit and should not be followed.” Obscenity considers profanities and in our case, the word “bullshit” contributes to the toxicity of this comment. This class of toxicity is arguably the most straightforward to solve as we could easily identify obscenity using a dictionary of profanities and check if there are comments who contain such words.

Insults

Example: “Has anyone told you that you are a little brat?”. From our example, we can observe that insults consists of rude or aggravated statements that involve an individual or a group. In our case, the comment is aimed at another individual, which is common but not necessary.

Threats

Example: “I will see to your slow and torturous death if I ever meet you”. From our example, we can observe that comments labelled under threats involve the life of another individual or parties closely related to the individual. Threats are usually statements that advocate for inflicting pain, injury or even death on oneself or others.

Identity Hate

Example: “Mate, you are such a nigger”. Converse to insults, identity hate is directed only at groups defined, which includes but are not limited by religion, ethnicity, gender, or sexual orientation. For instance, racist, and homophobic comments will be classified as identity hate.

Otherwise Toxic

Example: “Now get going and never come back, you are not welcomed here!” Comments that do not fall into the above four classes and are likely to cause other users to leave the discussion would fall into this category. Another example would be off-topic comments where individuals post about non-related content with the current ongoing discussion would constitute to trolling and fall into this category as well.

It is worthy to note that these classes are not mutually exclusive, in other words a single posted comment by an individual could be deemed toxic and at the same time fall into more than one out of the five defined classes above.

Data Description

To develop an effective machine learning model, we require a large dataset that contains user posted content and the dataset must also be human label based on the toxic category(categories) each content falls into. Our group was fortunate enough to find an ideal dataset which comes from the Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) and the data source is available for download from Kaggle. The Kaggle data source contains data pertaining to user posted content mainly from Wikipedia, which have been labeled by human raters for toxic behaviour. The data source consists of a training and testing dataset, which contains 160,000 records with 7 variables, and 153,000 records with 2 variables respectively. Each record represents a unique user posted comment. The fields in the training dataset include: unique id, comment, toxic, severe_toxic, obscene, threat, insult, identity_hate. While the fields in the testing dataset include only: id and comment. For the full list of features and their definitions, see Appendix A.

Data Munging

Regex Operations

In order to fully extract the information from comments posted by users, we are required to conduct transformation using regex operations as some verbal representations posted by users cannot be deciphered by the machine alone. The transformations we conducted include but are not limited to:

Transformation of emoticons

We conducted regular expression operations to transform smiley faces such as ":)" to a proper representation word such as "good". Furthermore, we have extracted a file of emoticons and its relevant representations so that we can conduct proper transformation such as; "☹" to "sad".

Transformation of non-words

We converted non words such as "&" to "and", "@" to "at" and numbers to its english translation.

Transformation of contraction words

We replaced contraction words to a better presentation that does not contain any special examples, such as "I'm" to "I am". This is due to the fact that Glove and FastText vectors are unable to recognize special symbols.

Other miscellaneous instances

Lastly we replaced newline tokens with empty spaces and at the same time removed url links and ip addresses as we believe that these characters will not provide extra information.

Exploratory Data Analysis

Before beginning our modelling, we performed some basic Exploratory Data Analysis to better understand our data. This section can be split into 4 sub sections: 1) Statistical Analysis, 2) Sentiment Analysis, 3) Syntactic Analysis and 4) Sentiment Analysis.

Statistical Analysis

We first begin with a simple breakdown of the frequency of classes in our data.

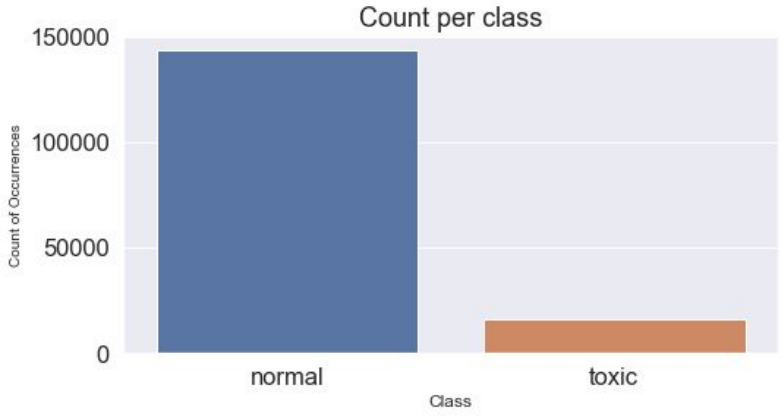


Figure 1: Label Count

We can observe from the plot that there is a class imbalance with 90% of the data comprising of normal text and the remaining 10% being toxic text. Imbalance in data class will be an issue in modelling as certain model may focus on predicting the majority class instead. To solve the issue of imbalance, instead of oversampling the minority class or undersampling the majority class, we will use the class_weight = 'balanced' in the model parameters which will give more weight to minority classes and have the same effect as oversampling. Next, we examine the count of each specific type of toxic comments.

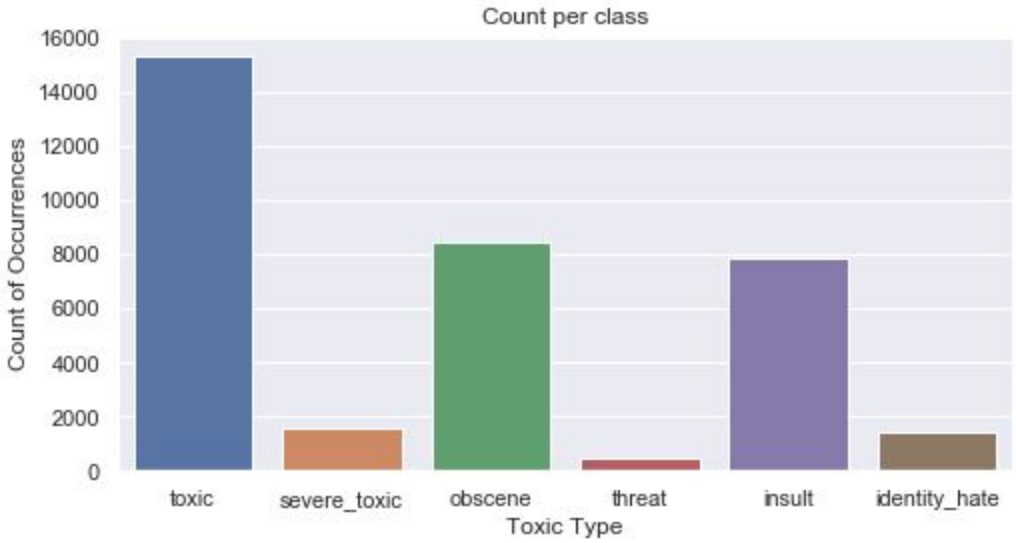


Figure 2: Toxic Label Count

A majority of the toxic comments are classified under toxic, followed by obscene,and insult with threats coming in at the last. Next, we will examine the number of overlapping classes.

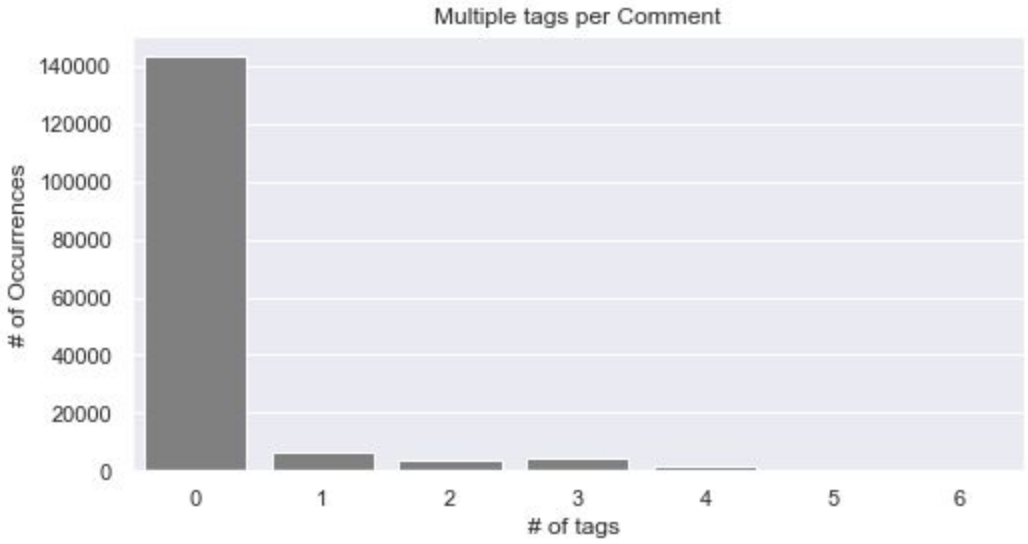


Figure 3: Multi Tags Count

Around half of the toxic comments actually belong in more than one class and have multiple tags. This indicates that some of the toxic comments may be correlated, therefore we shall examine their relationship next.

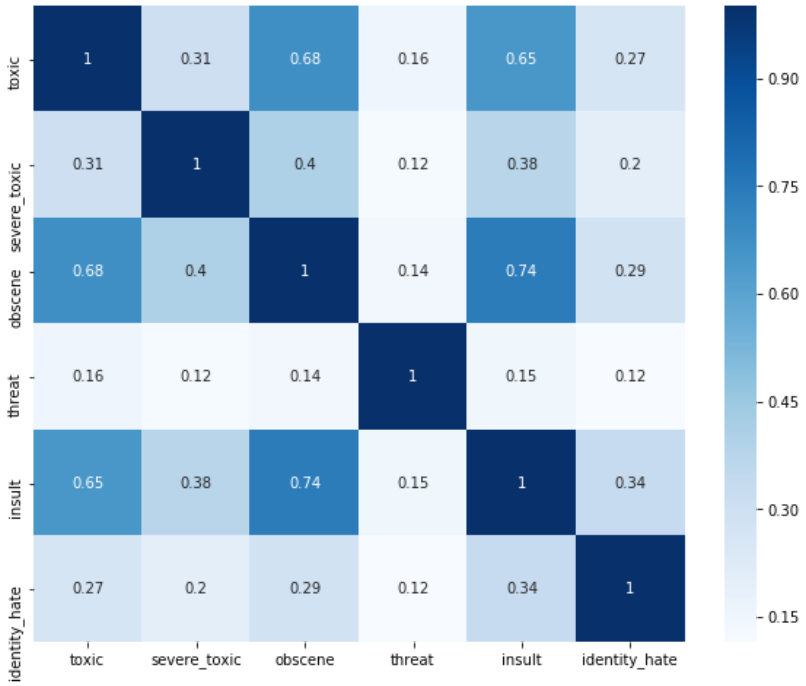


Figure 4: Correlation between Toxic Classes

From the plot of the correlation coefficient, we see that the pairs (toxic & obscene), (toxic & insult) and (insult & obscene) has a strong correlation with their correlation coefficient more than 0.6. Other pairs of classes with moderate correlation are (toxic & severe_toxic), (insult & severe_toxic) and (identity_hate & insult) with correlation coefficient between 0.3 and 0.6. The rest of the pairs have weak correlations.

We also examine the top 20 unigram and bigram in each toxic category as well as a wordcloud of each class. However, we will leave the results to the appendix section.

Sentiment Analysis

We perform sentiment analysis on the different class of toxic comment next. We utilise the SentimentIntensityAnalyzer function from the vader library to get the sentiment score of each text/comment. The function SentimentIntensityAnalyzer gives a score from 0 to 1 for how, positive, neutral and negative a text is. The closer the value is to 1 the stronger the particular sentiment is to the specific category. An example is shown below.

```
sentiment_dict['toxic']
```

	Text	neg	neu	pos	compound
0	cocksucker piss around work	0.773	0.227	0.000	-0.7783
1	hey talk exclusive group wp talibans good dest...	0.280	0.463	0.257	-0.4588
2	bye look come think comming back tosser	0.000	1.000	0.000	0.0000
3	gay antisemmitian archangel white tiger meow g...	0.189	0.739	0.072	-0.8451
4	fuck filthy mother ass dry	0.700	0.300	0.000	-0.7906

Figure 5: Sentiment Analysis Table Results

The function also gives a compound score, which is the overall sentiment of the text. The score ranges between -1 to 1. When the score is closer to -1 it means the text has a more negative sentiment, 0 means a neutral sentiment and 1 means a positive sentiment. As seen in the plot below, the majority of the normal text has a score of 0 and higher, while the toxic comments are all right skewed, majority having a very negative value. Having drawn the relationship that toxic comments have a more negative sentiment score, we could consider using it as a feature to identify toxic comments from normal text.

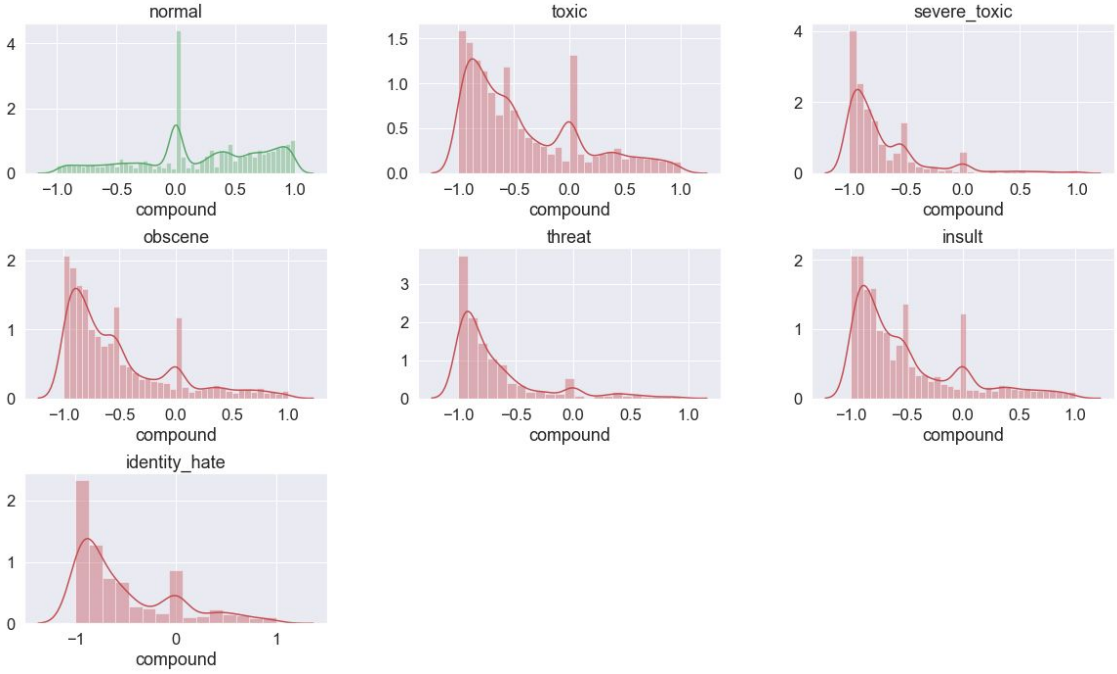


Figure 6: Sentiment Analysis Graphical Results

Syntactic Analysis

Next, using the stanford NLP package, we tried to perform some syntactic analysis to better understand our data. One of our ideas is to use Named Entity Recognition to discover the top locations and person mentioned in this comments. We got the following results:

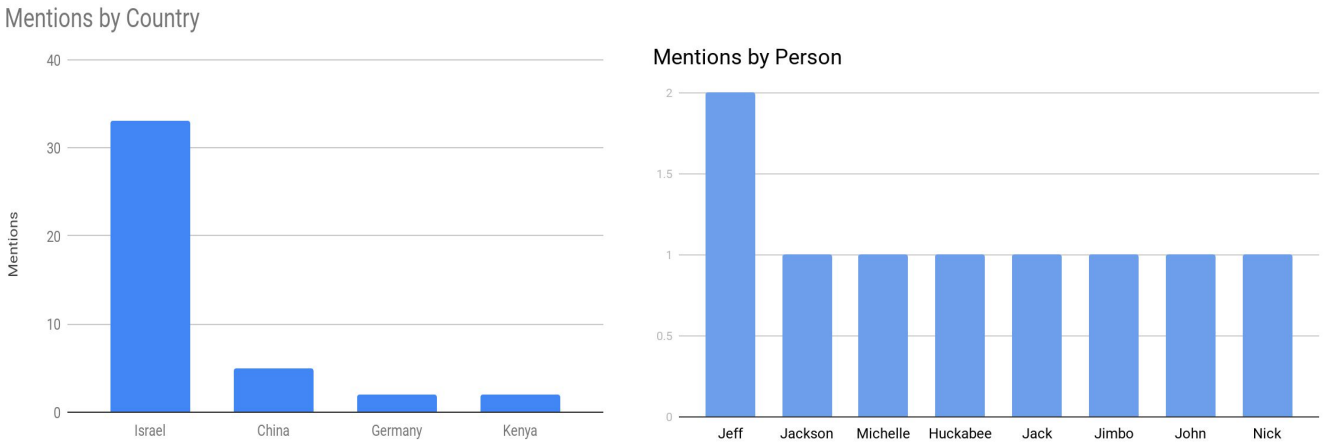


Figure 7: NER Graphical Results

Note that due to a limitation on our memory space, we could only perform the analysis on a subset of the data. Through this analysis, we were hoping that it could be used to identify who are the targets of such toxic and malicious comments. This part of the analysis could be integrated into our application as we are developing a telegram bot to identify toxic comments and we could identify victims who are under consistent cyber and verbal abuse. The application will be elaborated on in a later section.

Semantic Analysis

In this section, we perform Topic Modelling to understand the topics in each class of toxic comment. A flow chart of our process of performing the topic modelling is as shown.

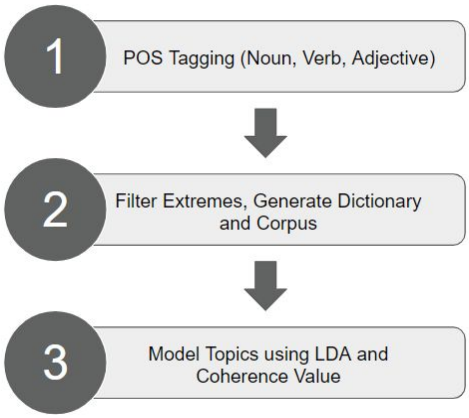


Figure 8: Topic Modelling Process Chart

We first perform POS tagging in order to keep the important words - Noun, Verb and Adjective. We realise through a series of trials that keeping this categories of word removes unnecessary words and produce more coherent words.

Next, we filter out extreme words that either occur many times or a small amount of time as we often feel these words does not really provide any valuable insights- they could be just common words or special terms (rare words).

Finally, we utilise LDA to build our topic models and coherence value to pick the most ideal number of topics.

Clean Text

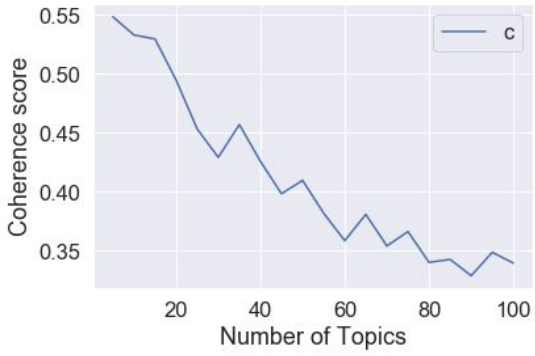


Figure 9: Coherence Score

We first perform topic modelling on the clean/normal text to understand our data. From the plot of the coherence model, it seems that the model with 5 topics seems to have the highest coherence. However, we will examine the model with 10 topics instead.

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9	Topic 10
word 1	utc	style	lol	article	world	wikipedia	page	english	article	discussion
word 2	category	game	year	deletion	school	image	talk	language	say	review
word 3	redirect	color	old	delete	high	use	edit	country	source	move
word 4	god	top	life	tag	system	page	wikipedia	state	think	new
word 5	john	video	picture	wikipedia	group	question	please	american	use	issue
word 6	christian	background	record	template	force	welcome	get	people	see	consensus
word 7	king	white	man	page	theory	copyright	user	city	make	project
word 8	religion	get	film	speedy	law	please	make	war	need	day
word 9	january	team	great	add	power	help	block	british	section	community
word 10	july	series	date	notable	public	fair	thanks	german	point	concern
word 11	march	width	character	please	study	edit	comment	government	find	month
word 12	august	border	music	subject	university	thank	want	united	write	week
word 13	muslim	movie	hey	guideline	control	ask	know	live	people	result
word 14	february	align	photo	criterion	woman	file	good	greek	name	dispute
word 15	april	red	big	note	science	free	editor	national	fact	involve
word 16	december	award	love	talk	political	test	revert	population	give	vote
word 17	october	episode	day	remove	military	talk	look	century	reference	process
word 18	town	best	song	notability	church	link	take	region	link	support
word 19	jun	class	release	place	map	medium	see	jew	seem	run
word 20	irish	season	play	web	human	article	edits	jewish	way	debate

Figure 10: Topics Result

We have tried interpreting some of the topics generated. They are as follows:

Topic 1: Summarise the months in a year
Topic 2: Colors and orientation eg style,background,width
Topic 3: Talking about some film/play and character
Topic 4: Removal of some wikipedia page. Talking about the guideline/criteria of using the page
Topic 5: Politics, Science, Religion - probably the topics of wikipedia
Topic 6: Again the guideline of using the wikipedia page-copyright issue and how to edit a page
Topic 7: Discussing about commenting on the page
Topic 8: Different races,countries and religion
Topic 9: Talking about having to cite a source and giving reference
Topic 10: Having discussion, reviews and debates on topics/issues.

We will proceed to examine the topics in the different toxic category next.

Toxic

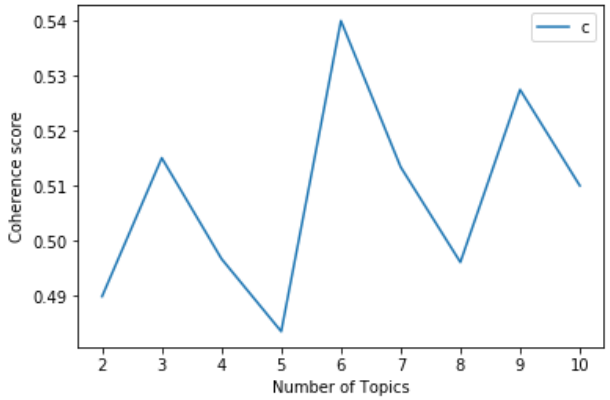


Figure 11: Coherence Score

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6
word 1	fuck	faggot	get	aid	huge	suck
word 2	hate	shit	page	die	care	fat
word 3	nigger	gay	know	fag	dont	jew
word 4	bitch	moron	wikipedia	bullshit	penis	cock
word 5	cunt	bad	people	hey	fucking	dick
word 6	dog	wiki	make	fucker	retard	love
word 7	eat	piece	article	dickhead	super	nipple
word 8	stupid	bush	block	wanker	take	wikipedia
word 9	bastard	vandalism	talk	guy	damn	old
word 10	shut	ball	say	cocksucker	ban	rice
word 11	pussy	admins	think	kill	anal	asshole
word 12	mother	cocksucking	stop	mothjer	fart	sex
word 13	hanibal	jones	edit	chicken	terrorist	boob
word 14	faggots	hitler	time	hairy	computer	vagina
word 15	want	admin	want	fool	arse	lick
word 16	nigga	poop	idiot	image	come	user
word 17	delanoy	cheese	delete	fuckin	block	homo
word 18	egg	hell	see	tell	sexual	dirty
word 19	vomit	rule	life	fan	small	whore
word 20	cline	george	stupid	white	say	big

Figure 12: Topics Result

The model with the highest coherence has a total of 6 topics. We will not be explaining the topics as the words are explicit. However, in general, the topics summarise the different common ways that people send toxic comments.

Next, we will examine the severe_toxic comments.

Severe Toxic

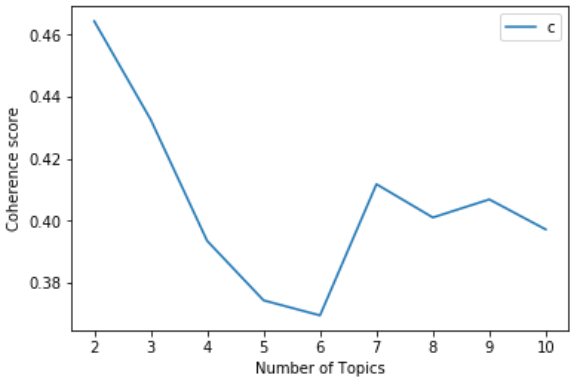


Figure 13: Coherence Score

	Topic 1	Topic 2
word 1	fuck	shit
word 2	suck	faggot
word 3	bitch	cunt
word 4	die	fuck
word 5	cock	piece
word 6	wikipedia	penis
word 7	nigger	eat
word 8	fucker	bastard
word 9	kill	huge
word 10	asshole	shut
word 11	cocksucker	damn
word 12	mothjer	rape
word 13	dick	fat
word 14	gay	stupid
word 15	dog	anal
word 16	mexican	fucking
word 17	dickhead	nigger
word 18	idiot	pussy
word 19	love	get
word 20	bush	want

Figure 14: Topics Result

The model with the highest coherence has a total of 2 topics. Again, We will not be explaining the topics as the words are explicit. However, we can see that the severe toxic category has a higher concentration of explicit word-showing the intensity of toxicness.

Next, we will examine the obscene comments.

Obscene

Figure 15: Coherence Score

	Topic 1	Topic 2
word 1	fuck	faggot
word 2	suck	nigger
word 3	bitch	huge
word 4	shit	stupid
word 5	love	fuck
word 6	hey	get
word 7	guy	page
word 8	dick	know
word 9	cock	fag
word 10	fucker	wikipedia
word 11	cunt	block
word 12	chicken	shit
word 13	bastard	make
word 14	cocksucker	die
word 15	mothjer	people
word 16	penis	say
word 17	vagina	edit
word 18	nipple	article
word 19	bullshit	talk
word 20	hanibal	think

Figure 16: Topics Result

The model with the highest coherence has a total of 2 topics. Again, We will not be explaining the topics as the words are explicit. In general, obscene comments are those sexual in nature.

Next, we will examine the threat comments.

Threat

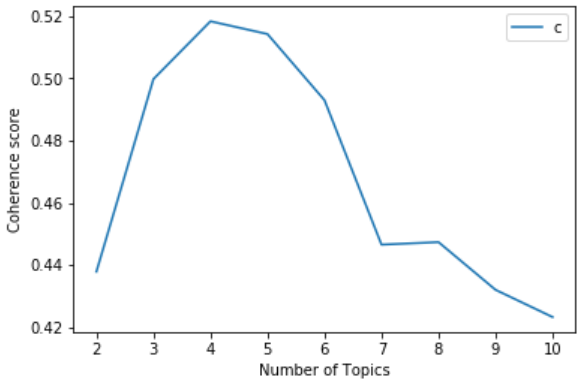


Figure 17: Coherence Score

	Topic 1	Topic 2	Topic 3	Topic 4
word 1	fuck	kill	die	fuck
word 2	time	block	wikipedia	shit
word 3	get	ban	live	kill
word 4	stop	talk	pathetic	get
word 5	know	page	fool	bitch
word 6	people	murder	fuck	hope
word 7	want	fuckin	bitch	hell
word 8	see	take	rape	come
word 9	edit	steal	fat	delete
word 10	stupid	continue	head	death
word 11	look	fuck	jew	know
word 12	cunt	gonna	shoot	piece
word 13	life	user	mom	dead
word 14	head	destroy	ground	life
word 15	fat	skin	stupid	rape
word 16	make	shove	hope	say
word 17	hope	cut	brain	burn
word 18	revert	ass	splatter	think
word 19	edits	use	son	house
word 20	give	pull	nazi	page

Figure 18: Topics Result

The model with the highest coherence has a total of 4 topics.We can observe that for threatening comments, they are often related to killing/shooting/murdering someone.

Next, we will examine the insult comments.

Insult

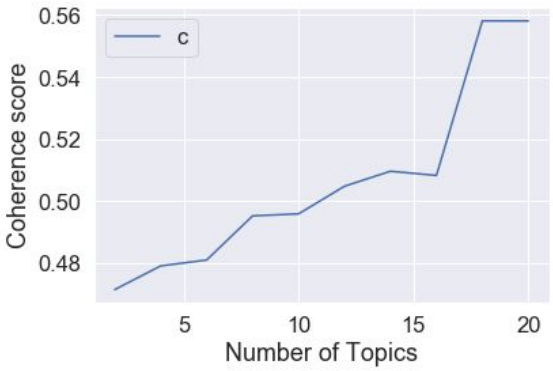


Figure 19: Coherence Score

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9	Topic 10
word 1	nigger	fat	fucking	cock	suck	know	fuck	wikipedia	get	asshole
word 2	moron	jew	penis	huge	dick	die	shit	guy	block	idiot
word 3	chicken	old	fire	twat	mexican	fool	eat	love	little	page
word 4	middle	repeat	small	retard	john	pathetic	shut	please	life	delete
word 5	ethnic	dumb	mouth	whore	slap	time	admins	stop	come	people
word 6	cracker	scum	house	dirty	thomas	people	cunt	think	say	make
word 7	bridge	coward	god	eat	voice	live	cocksucking	see	fuckin	wikipedia
word 8	talkpage	american	use	lol	july	image	mother	change	big	article
word 9	damage	mod	douchebag	jerk	fukin	tommy	atheist	dont	make	leave
word 10	association	limp	dipshit	blah	plain	year	semen	stupid	man	person
word 11	slavic	asian	damn	sucker	insignificant	world	ban	say	post	think
word 12	username	sue	opinion	indian	reverts	respect	cant	want	information	stop
word 13	blind	commie	police	meet	cena	claim	warn	give	computer	vandalism
word 14	iraq	muslim	large	fuk	monitor	hope	sign	care	shit	user
word 15	detail	sikh	turd	tool	dam	way	write	edit	day	get
word 16	participate	rant	treat	online	direct	nothing	mum	call	real	edit
word 17	wash	arab	hot	inch	pope	mean	face	people	shithead	right
word 18	oppose	rich	expert	bro	bigoted	fact	pic	anything	job	good
word 19	united	redirect	boy	fuckwit	alert	find	fucken	like	hope	talk
word 20	royal	thick	miserable	pain	hog	country	gwernol	everyone	mom	put

Figure 20: Topics Result

The model with the highest coherence has a total of 18 topics. There are probably more insult topics, as there could be a variety of ways to insult a person. In general, we can observe that insulting comments general involve calling people explicit and ugly names as well as insulting one's intelligence.

Next, we will examine the identity hate comments.

Identity Hate

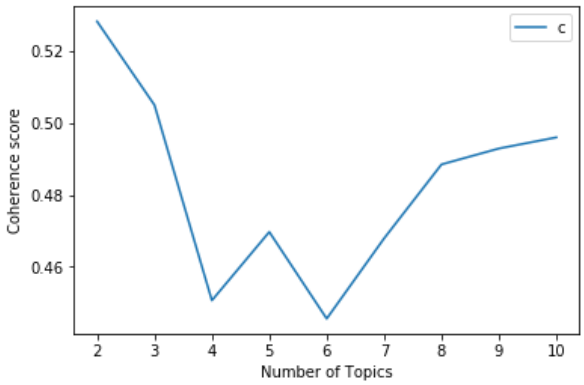


Figure 21: Coherence Score

	Topic 1	Topic 2
word 1	faggot	nigger
word 2	fuck	jew
word 3	suck	fat
word 4	huge	die
word 5	mexican	fuck
word 6	gay	gay
word 7	get	cunt
word 8	jewish	stupid
word 9	make	hate
word 10	people	shit
word 11	shit	bitch
word 12	bitch	licker
word 13	page	kill
word 14	know	drink
word 15	dick	nigga
word 16	wikipedia	keep
word 17	think	eat
word 18	say	homo
word 19	want	faggot
word 20	stop	utc

Figure 22: Topics Result

The model with the highest coherence has a total of 2 topics.We can observe that for the identity hate comment, the comments are often about the country and race of a person eg Mexican, Jew or insulting one's sexual orientation.

Vectorization

Machine learning models only accepts numerical attributes as inputs which would allow them to perform certain mathematical operations such as matrix factorisation, dot product etc. However as texts are non-numerical, we have to convert texts into word vectors before feeding the inputs into the machine learning models. On the basic level, we use **CountVectoriser**, a One-hot Representation method to represent a word as a vector in which only one element is 1 and the other elements are 0 in the vector. Although one-hot representation is simple, there are weak points: it is impossible to obtain meaningful results with arithmetic between vectors. Another weak point is the vector tend to become very high dimension. Since one dimension is assigned to one word, as the number of vocabulary increases, it becomes very high dimension.

As not all words are equally important in representing toxicity in comments, it is useful to capture the word importance during vectorization. We thus uses **TF-IDF** which stands for Term Frequency-Inverse Document Frequency that can tell us the importance of the word in the corpus or dataset. TF-IDF is basically a multiplication between two concepts Term Frequency(TF) and Inverse Document Frequency(IDF). Term Frequency tells us how frequent each word appears in the document while inverse document frequency tells us the importance of each word based on the fact that less frequent words are more informative and important.

However, the two vectorization method above create sparse, high dimensional word vectors. We thus explore **Word embeddings techniques** that express word as a low-dimensional real valued vector, typically 50 to 300 floating-point numbers and thus serve as the input layer. As opposed to sparse, one-hot encoded vectors, these dense vectors can capture and represent word similarity by cosine similarity of the vectors. Beyond simple distance measurements, arithmetics with words can be performed as presented with the **Word2Vec model**. The similar approaches **GloVe** and **FastText** provide alternative ways to calculate word embeddings.[4]

Word2Vec

Word2vec was created and published in 2013 by a team of researchers led by Tomas Mikolov at Google. It consists of a two-layer neural network that takes text corpus as an input and produce a set of vectors for its output. It is able to transform text into numerical representations so that can parse the output into deep neural networks. The resulting output positions words with similar context close to one another in the vector space.

GloVe

GloVe, stands for global vectors for word representation, is an unsupervised learning algorithm created by developer from Stanford which is primarily built for obtaining vector representations for words by aggregating global word-word co-occurrence matrix from a corpus. The developers at Stanford

[4] https://medium.com/@Hironsan/why-is-word-embeddings-important-for-natural-language-processing-6b69dd384a77

hypothesized that GloVe's superiority is contributed to its ability to achieve analogy preservation under linear arithmetic that uses only fundamental statistical properties of the corpus as inputs. GloVe uses log-bilinear regression model for unsupervised learning of word representations, it combines the features of two model families, namely the global matrix factorization and local context window methods

FastText

FastText is basically an extension of the word2vec model, difference is – it process each word as compositions of character n-grams. Therefore a word is essentially the sum of its character n-grams. This difference enables FastText to generate better word embeddings for rare words and also construct vector for a word from its character n-gram even if it does not exist in the training corpus.

Comparison of embedding techniques

FastText is particularly suited for toxic comments because it uses subword embeddings. The advantage of subword embeddings is that they overcome the out-of-vocabulary problem. Toxic comments often use obfuscation, for example "Son of a B****", "***k them!!!!" but also misspelled words, which are common in online discussions. Fast-paced interaction, small virtual keyboards on smartphones, and the lack of editing/correction tools reinforce this problem. Word2Vec and GloVe fail to find a good representation of these words at test time because these words never occurred at training time. These words are out-of-vocabulary. In contrast, FastText uses known subwords of the unknown word to come up with a useful representation. The ability to cope with unknown words is the reason why previous findings on the inferiority of word embeddings in comparison to word n-grams have become outdated.

Detection Modelling

After performing EDA on our data, we will proceed to do modelling to identify the different classes of toxic comments. We will test out 2 main methods of detection - Machine Learning Modelling and Deep Learning Modelling and compare their respective results.

Machine Learning

For Machine Learning Modelling, we will primarily be using sklearn's library of models available. The following is a flowchart of the process of our Machine Learning Modelling.

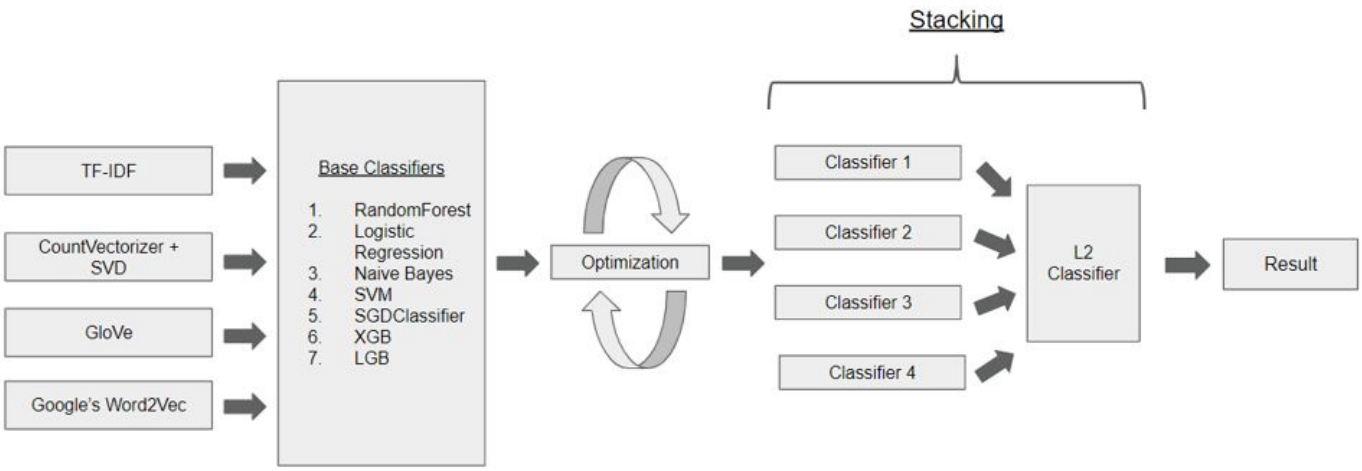


Figure 23: Machine Learning Flowchart

In step 1, we will try out the various methods of vectorization and word embeddings before fitting it into some of the popular ML Classifiers for NLP. We will then optimize the hyper parameters of the models to improve the performance of top models. We will then select the top 3/4 models to form Level 1 estimators for our stacking model. The output from the L1 estimators will be feed into the level 2 classifier before outputting the final results.

Note that the metrics that we will be using will the ROC_AUC because 1) It is the metrics used in the competition and it will be a good gauge to compare our results with the top competitors in the competition. 2) It gives a good measure of True Positive Rate and False Positive Rate instead of the overall accuracy. In this case, it is important to identify as many toxic comments as possible(TP) but also reduce the number of false positives such that we do not wrongly flag normal comments especially when we are using it for commercial purpose/in real life.

Base Classifiers

We fit the base classifiers(default parameters) with the different embeddings to find the best models/embeddings to further optimize and focus on. The results are as follows.

TF-IDF

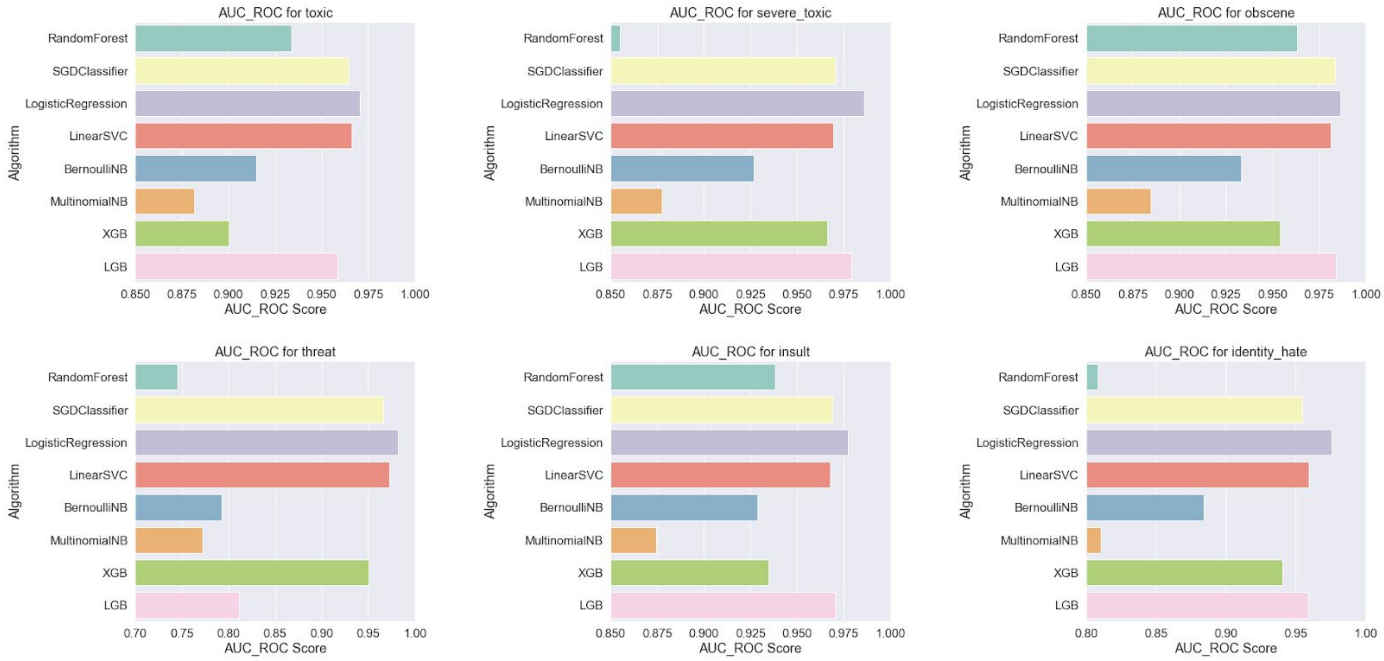


Figure 24: TF-IDF Model Results

CountVectorizer

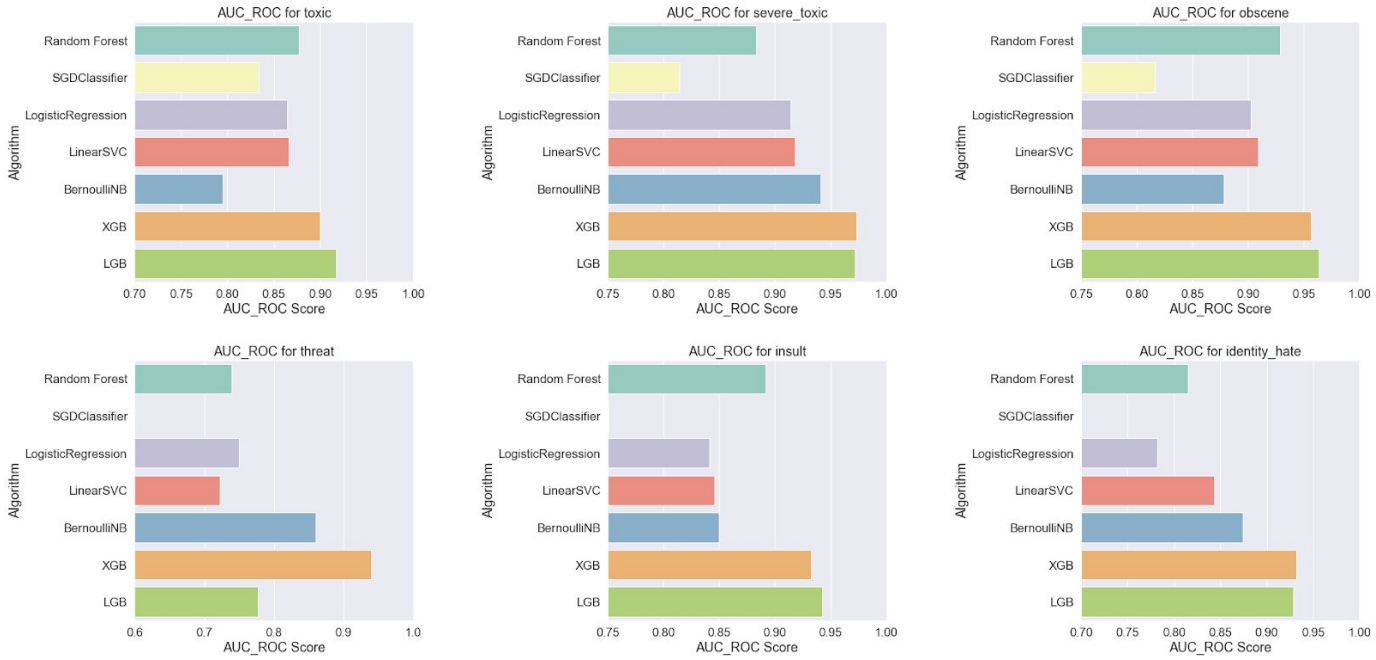


Figure 25: Count Vectorizer Model Results

GloVe

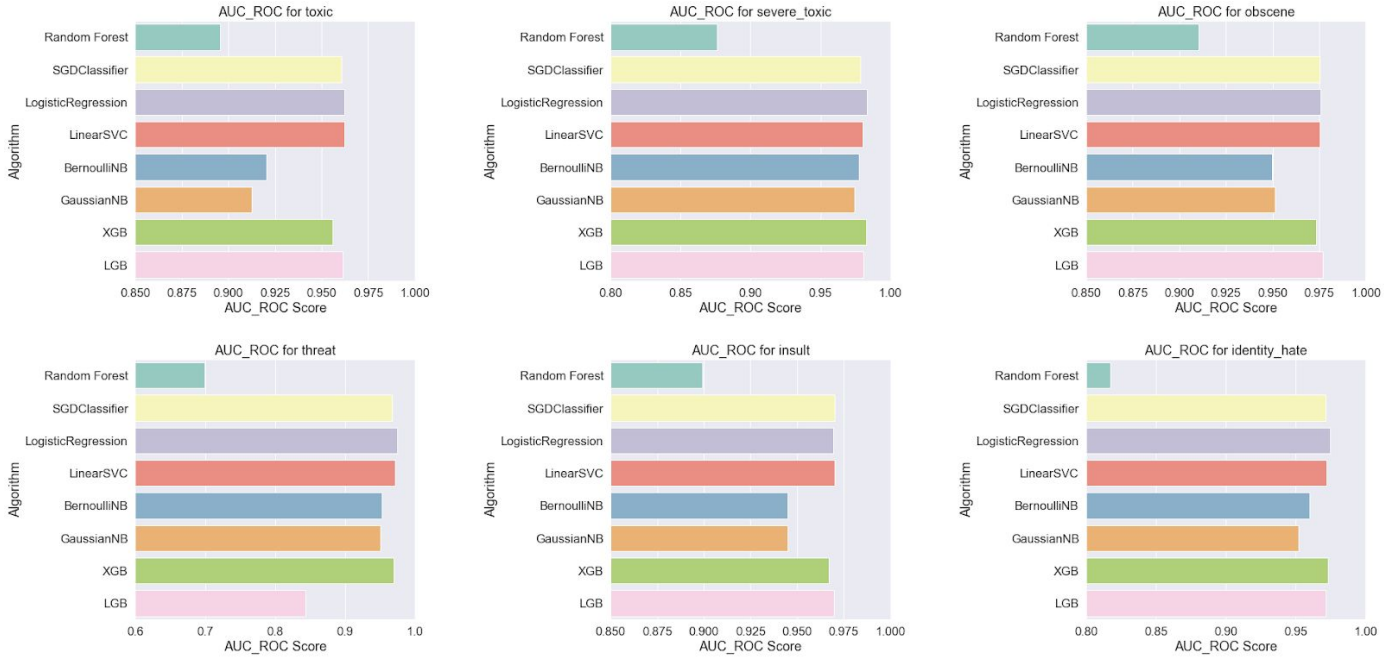


Figure 24: GloVe Model Results

Google's Word2Vec

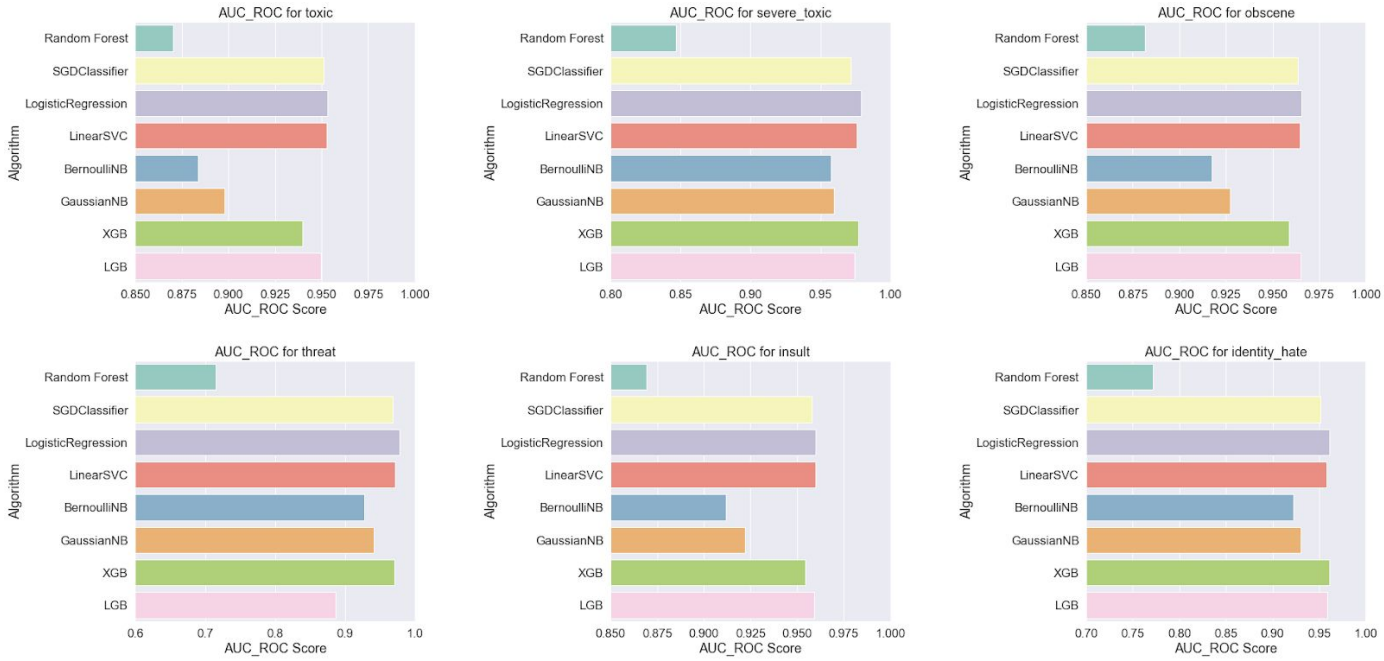


Figure 24: Google Word2Vec Model Results

From the results of using the 4 different vectorization and embedding methods, we see varying results.We can conclude from the results 1) Word Embeddings can have an adverse impact on model performance 2) Certain Models perform better in predicting a particular category and with a particular word embedding.

For 1) we can see that after using stanfordnlp's GloVe word embedding and Google's word2vec, the ROC_AUC score of the models improve tremendously on a whole as compared to vectorization, this is due to the strength of the embeddings as mentioned in the section above. Therefore, we could focus on optimizing the GloVe and word2Vec model instead.

For 2) Models can be split into 3 main categories: Regression, Tree-based and Naive Bayes techniques. Generally, the Regression based techniques eg Logistic Regression, SGDClassifier, Linear SVC as well as Tree based models with the exception of Random Forest performs superiorly better than the Naive Bayes techniques.

A possible reason why Naive Bayes technique performs poorly could be due to its assumption of independence between features, which is not the case most of the time. There could be a strong violation in this case, which resulted in the poor performance. Another reason could be due to the class imbalance issue which may resulted in toxic words to appear in smaller quantity and becomes harder for the Naive Bayes

method to capture it. As for the reason why Random Forest performs poorly could be due to its nature of classification which is rule based and also because we are using the default setting of the model which could result in overfitting.

In this case, it seems that models with some form of statistical based classification eg Regression and Gradient Boosted tree performed better. In particular, Logistic Regression and Gradient Boosted Tree models are known to perform well for NLP related problems. Therefore, we should focus on optimizing these models.

Optimization

After identifying the top performing models, we will perform hyperparameter tuning to further improve the ROC_AUC of the models. In particular, we will be using Bayesian Optimization instead of GridSearchCV or RandomSearchCV for our hyperparameter tuning. Bayes Optimization focuses on finding the minimum/maximum of an objective function, therefore it tends to require fewer iterations to find the most optimal set of parameter as compared to GridSearch and RandomSearch and is shown to be equally effective.

```
def bayes_opt(classifier,param_space,X_train,y_train,metric):

    def objective_function(param_space):

        clf = classifier(random_state=1234, **param_space)
        score = cross_val_score(clf, X_train,y_train, cv=5,scoring=metric).mean()
        return {'loss': -score, 'status': STATUS_OK}

    trials = Trials()
    best_param = fmin(objective_function,
                      param_space,
                      algo=tpe.suggest,
                      max_evals=100,
                      trials=trials,
                      rstate= np.random.RandomState(4222)
                     )

    loss = [x['result']['loss'] for x in trials.trials]

    print("")
    print("##### Results")
    print("Score best parameters: ", min(loss)*-1)
    print("Best parameters: ", best_param)
```

Figure 25: Optimization Implementation

To utilise Bayes Optimization, we first have to define an objective function. As shown in the figure above. We then define the parameter space to search for the most optimal parameter just like GridSearchCV.

```
log_param_space1 = {"penalty": hp.choice("penalty", ['l1','l2']),
                    "fit_intercept" : hp.choice("fit_intercept", [True,False]),
                    "C": hp.choice("C", list(np.logspace(-3,5,500))),
                    "max_iter":hp.choice("max_iter", [50,100,200,500]),
                    "n_jobs":hp.choice("n_jobs", [3])}

bayes_opt(LogisticRegression,log_param_space1,xtrain_glove,train_y['toxic'],"roc_auc")
```

```
100%|██████████| 100/100 [3:31:49<00:00, 127.10s/it, best loss: -0.9622984966056105]

##### Results
Score best parameters:  0.9622984966056105
Best parameters:  {'C': 208, 'fit_intercept': 0, 'max_iter': 1, 'n_jobs': 0, 'penalty': 1}
```

Figure 26: Parameter Space

The function will then search over the parameter space and return the set of parameters that results in the highest ROC_AUC score as shown above. We will perform bayesian optimization for the top selected models.

The following are the classifiers optimized and the parameter space that are searched.

Classifier	**Parameters Tuned**
Logistic Regression	"penalty": hp.choice("penalty", ['l1','l2']), "fit_intercept" : hp.choice("fit_intercept", [True]), "C": hp.choice("C", list(np.logspace(-3,3,300))), "max_iter":hp.choice("max_iter", [50,100,200,500]), "class_weight":hp.choice("class_weight", ["balanced",None])
Light Gradient Boosting	"boosting_type": hp.choice("boosting_type",["gbdt","dart","goss"]), "learning_rate":hp.choice("learning_rate", [0.01,0.05,0.1,0.5,1]), "gamma":scope.int(hp.quniform('gamma', 1, 5, 1)), 'reg_alpha': hp.choice('reg_alpha', [1e-5,1e-2, 0.1, 1, 100,200]), 'reg_lambda':hp.choice('reg_lambda', [1e-2, 0.1, 1, 100,200]), "n_estimators" : hp.choice("n_estimators", [10,50,100,300]), "class_weight":hp.choice("class_weight", ["balanced",None])
Stochastic Gradient Descent Classifier	"loss":hp.choice("loss",["hinge", "log", "modified_huber", "squared_hinge", "perceptron"]), "penalty": hp.choice("penalty", ['l1','l2']), "alpha" :hp.choice("alpha", [0.0001,0.001,0.01,0.1,1,10,100,1000]), "fit_intercept" : hp.choice("fit_intercept", [True,False]), "max_iter":hp.choice("max_iter", [500,1000,2000]), "class_weight":hp.choice("class_weight", ["balanced",None]), "learning_rate":hp.choice("learning_rate", ["constant","optimal","invscaling","adaptive"]), "eta0":hp.choice("eta0", [0.0001,0.001,0.01])
Linear SVC	"fit_intercept" : hp.choice("fit_intercept", [True,False]), "C": hp.choice("C", list(np.logspace(-3,3,300))),

	"max_iter":hp.choice("max_iter", [100,200,500,1000,2000]), "class_weight":hp.choice("class_weight", ["balanced",None])
XGBoost	"learning_rate": hp.choice("learning_rate", [0.05, 0.07, 0.1, 0.15, 0.2]), "gamma": scope.int(hp.quniform("gamma", 0,4,1)), "subsample": hp.uniform("subsample", 0.6, 1.0), "colsample_bytree": hp.uniform("colsample_by_tree", 0.4, 1), "reg_alpha": hp.choice("reg_alpha", [1e-5, 0.1, 1, 100]), "reg_lambda": hp.choice("reg_lambda", [1e-5, 1e-2, 0.1]), "n_estimators": hp.choice("n_estimators", [100, 200, 500])

Stacking

Finally, after optimizing the top performing model, we will proceed to perform stacking.

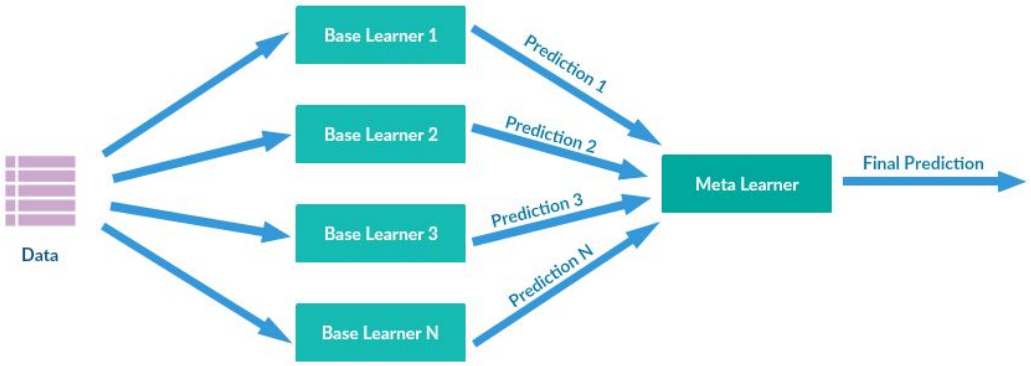


Figure 27: Stacking Infrastructure

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regression. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features.[5]

Note that due to a limitation of time, we are not able to perform stacking on all six categories, as it takes a significant time to optimize existing models. However, we have managed to build a total of 3 stacked models. The results are as follows

[5] https://blog.statsbot.co/ensemble-learning-d1dcd548e936

Results

Toxic

L1 Classifiers	TF_IDF Logistic Regression	TF_IDF LinearSVC	GloVe SGDClassifier	Glove LinearSVC
ROC_AUC CV Train Score	97.15%	97.30%	96.27%	96.38%

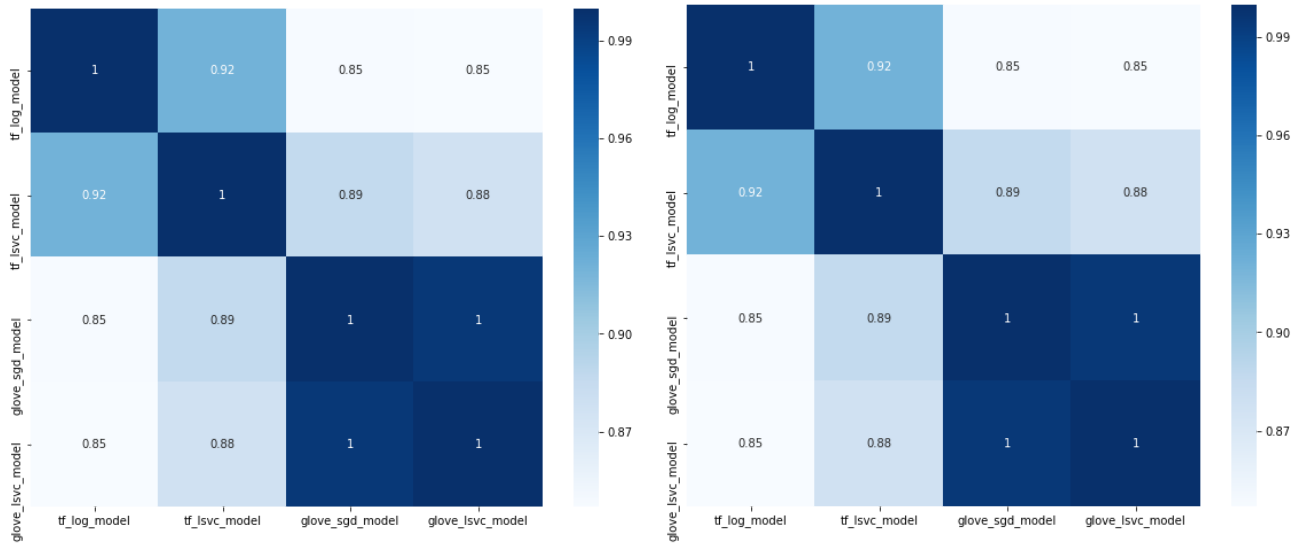


Figure 27: Train Test Results Correlation

We plot out the correlation coefficient graph of the L1 results from the L1 estimators. We notice that the estimators using similar embeddings/vectorizations are highly correlated to each other while estimators with different embeddings are also highly correlated albeit slightly lower than those with similar embeddings. This could be because the existing base models already have very high ROC_AUC score and stacking will only have slight improvement to the score. Nevertheless, we will still proceed with building the L2 Model.

L2 Classifier	XGBoost
ROC_AUC CV Train Score	97.71%
ROC_AUC Test Score (Predict_Proba)	96.67%
ROC_AUC Test Score (Predict)	89.83%

We can see from the results stacking do improve the results but only by a slight amount. The CV train score of the L2 classifier was able to outdo the performance of all the L1 classifiers. The final test score was also not far off from the train score. Note that the second test score is using the function model.predict instead of model.predict_proba. The probabilities were converted to 0 or 1 based on an optimal threshold found during cross-validation. The final confusion matrix is as shown.

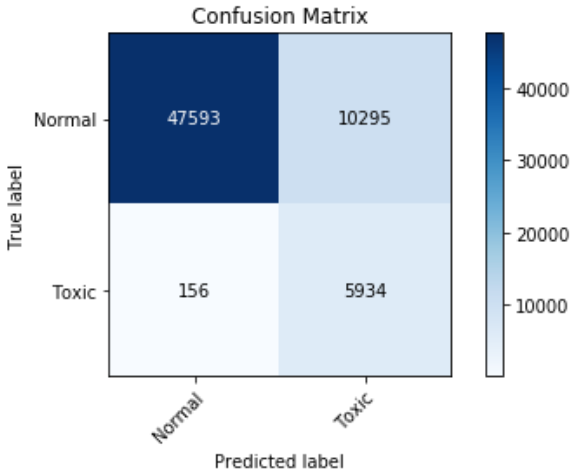


Figure 28: Confusion Matrix for Toxic

We can see that we manage to catch most of the toxic comments, failing to catch about 2.56% of it only. However, there is a high false positive, where normal comments are labelled as toxic. Depending on the objective of the moderator using our model, the moderator could be more precautious and allow for a higher false positive rate in order to catch more of the Toxic comments. Nevertheless, for future improvement we would need to work on improving the f1 score which is a balance of recall and precision to lower false positives while capturing most of the true positive.

Severe Toxic

L1 Classifiers	**TF_IDF Logistic Regression**	**TF_IDF LGB**	**GloVe Logistic Regression**	**Glove LGB**
ROC_AUC CV Train Score	98.62%	98.17%	98.46%	98.55%

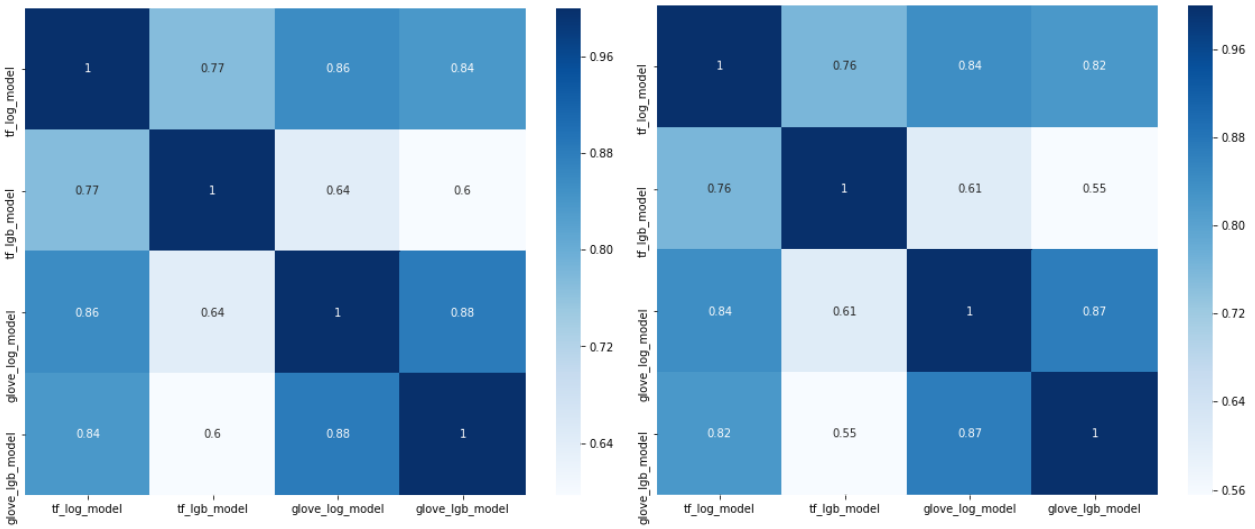


Figure 29: Train Test Results Correlation

We notice a strong correlation between models using similar embeddings/vectorization. However, an additional observation is how the tf-idf logistic regression model has a high correlation with all other models. This could be due to its' already high ROC_AUC score. We will proceed with the L2 Modelling.

L2 Classifier	**XGBoost**
ROC_AUC CV Train Score	98.82%
ROC_AUC Test Score (Predict_Proba)	98.77%
ROC_AUC Test Score (Predict)	94.73%

Again, we can see from the results a minor improvement from stacking. The CV train score of the L2 classifier was able to outdo the performance of all the L1 classifiers. The same method to convert the probabilities of the test set to 0 and 1 was also used here as mentioned above. The technique will be used for the remaining classes. The confusion matrix is as follows:

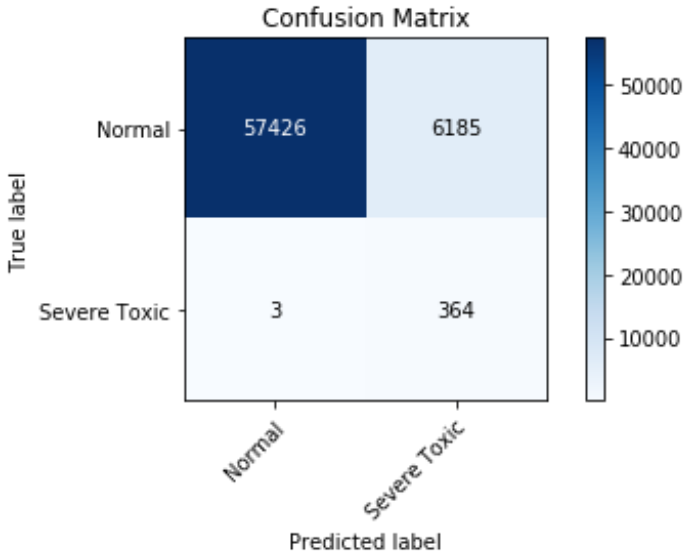


Figure 30: Confusion Matrix for Severe Toxic

Again, we can see that we manage to catch most of the severe toxic comments, failing to catch about 0.82% of it only. There is a high false positive, where normal comments are labelled as severe toxic. Therefore for future improvement we could work on improving the f1 score which is a balance of recall and precision to lower false positives while capturing most of the true positive.

Obscene

Estimator	**TF_IDF Logistic Regression**
ROC_AUC CV Train Score	98.72%
ROC_AUC Test Score (Predict_Proba)	97.57%
ROC_AUC Test Score (Predict)	92.03%

As mentioned due to a lack of time, we only manage to build 3 stack models, however the single model for class obscene still has a relatively high ROC_AUC score for its train and test results.

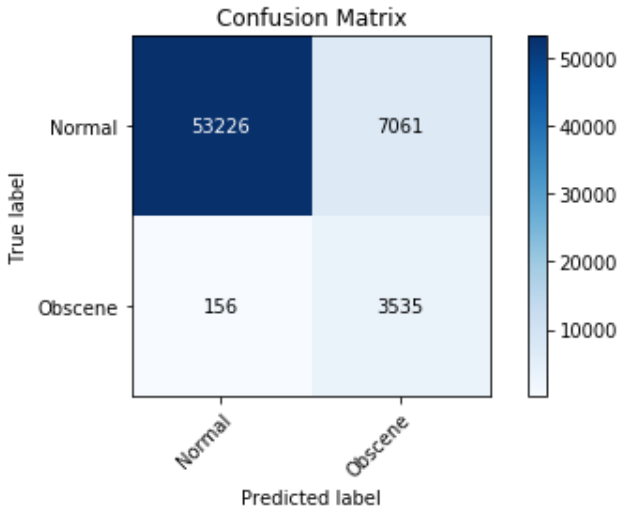


Figure 31: Confusion Matrix for Obscene

The model also successfully manage to catch most of the obscene comment but face a slightly high false positive rate, which could be improved in the future.

Threat

L1 Classifiers	TF_IDF Logistic Regression	TF_IDF LinearSVC	GloVe Logistic Regression	Glove LinearSVC
ROC_AUC CV Train Score	98.33%	98.19%	97.68%	97.64%

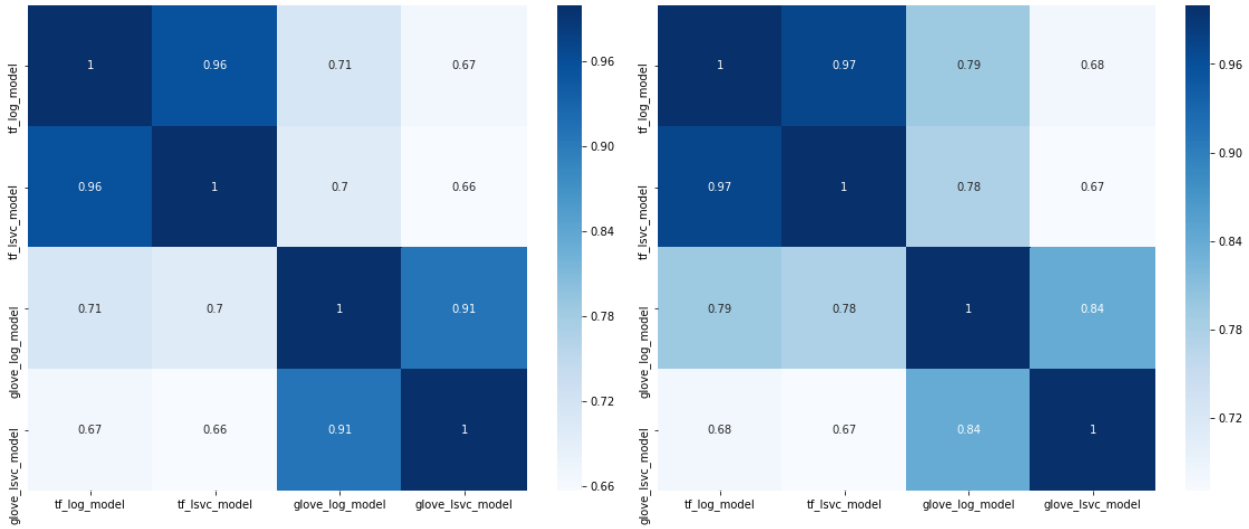


Figure 32: Train Test Results Correlation

For the threat L1 models, models with same embeddings/vectorization have high correlation probably due to the same reasons as explained before. However, we will still proceed with the stacking to try and improve the overall results.

L2 Classifier	**XGBoost**
ROC_AUC CV Train Score	98.51%
ROC_AUC Test Score (Predict_Proba)	99.14%
ROC_AUC Test Score (Predict)	95.50%

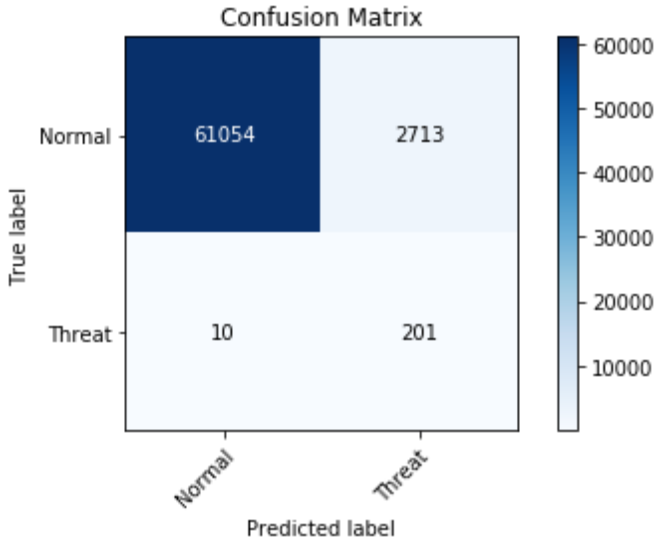


Figure 33: Confusion Matrix for Threat

The results from stacking shows a slight improvement. From the confusion matrix, we see that false positives are lower this time round while the amount of true positives is high. This could be due to a smaller proportion of threat comments in the test set. Nevertheless, the false positives could still be reduced further in the future.

Insult

Estimator	TF_IDF Logistic Regression
ROC_AUC CV Train Score	97.82%
ROC_AUC Test Score (Predict_Proba)	96.77%
ROC_AUC Test Score (Predict)	90.52%

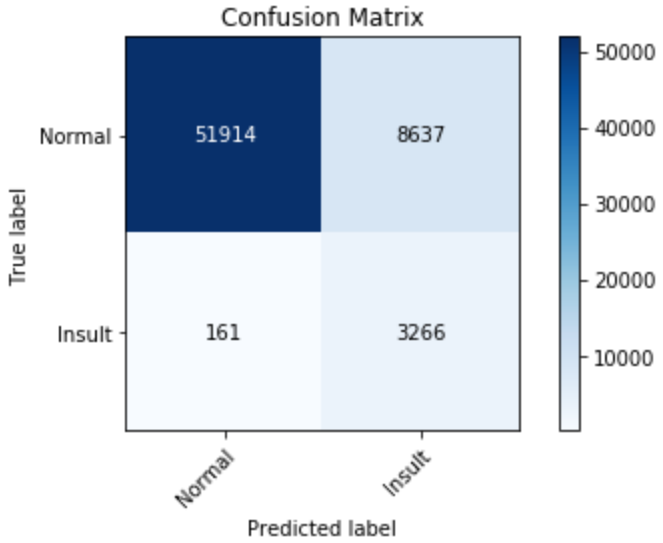


Figure 34: Confusion Matrix for Insult

For the insult model, we are only using one Logistic Regression model which has been optimized. However, the ROC_AUC score is still as high as the other categories/class. From the confusion matrix, we observe that it has a high true positive but also a high false positive. Again, this will be an area of improvement in the future.

Identity Hate

Estimator	TF_IDF Logistic Regression
ROC_AUC CV Train Score	97.68%
ROC_AUC Test Score (Predict_Proba)	98.08%
ROC_AUC Test Score (Predict)	92.84%

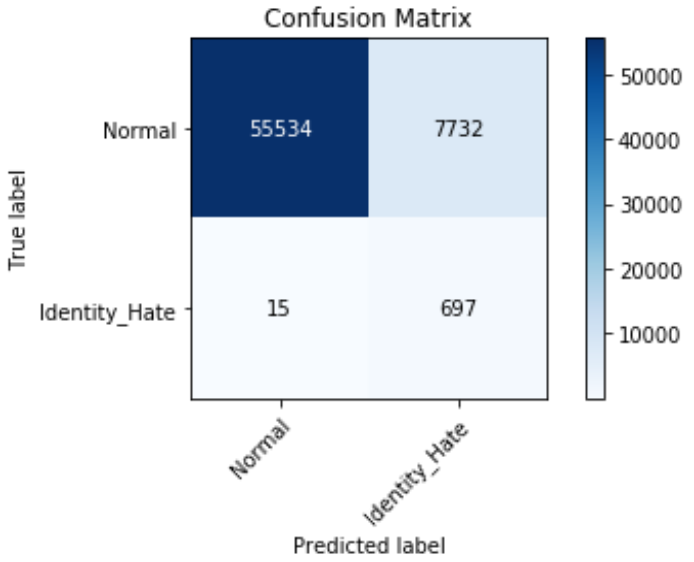


Figure 35: Confusion Matrix for Identity Hate

Finally, for the identity model, we are also using only one Logistic Regression model which has been optimized. The ROC_AUC score is still as high as the other categories/class. From the confusion matrix, we observe that it has a high true positive but also a high false positive. Again, this will be an area of improvement in the future.

Chain Modelling

As mentioned in our exploratory data analysis, we could observe that there are certain pairs of classes that has a strong correlation e.g. toxic & obscene. Our group thus decided to explore chain modelling to use the correlation of labels to our advantage. The intuition of chain modelling is simple; a chain of binary classifiers C_0, …, C_n is constructed where C_i uses the predictions of all classifier C_j, where j is smaller than i.[6] In other words, the total number of classifiers needed for chain modelling is equal to the number of classes.

Chain Modelling Illustration

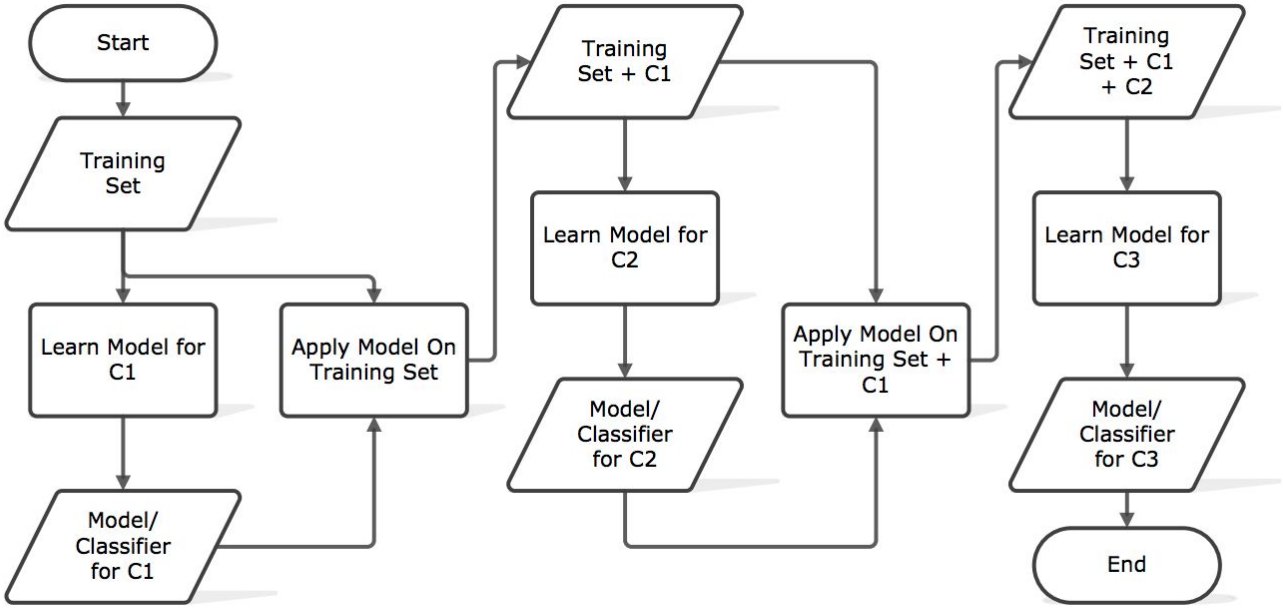


Figure 36: Chain Modelling Illustration

The above figure is a simplified illustration of a chain modelling process with three different classifiers (which also means the problem has three different classes). From the start of the process, we first train the binary classifier for class one, subsequently we use the output of the first classifier plus the training set to train the second binary classifier for class one. Finally, we use the output of the first and second classifier plus the training set to train the third binary classifier for class three.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.ClassifierChain.html

Chain Modelling Result

In our case, we only fit one model (Logistic Regression) into the entire chain due to limitation of time. Instead we focus on finding the optimal chain of predictions. As there are a total of 6 classes/Labels, this translates to a total of 6! = 720 permutations of possible chain and we put every permutation into the test.

Our final results show that [5, 4, 3, 2, 1, 0] is the optimal chain for prediction and we got a ROC_AUC score of 95.96% on the test set. There is definitely room for improvement given that we only tried one model without any optimization of the feature or additional model.

Machine Learning Summary and Future Improvement

To summarise, we can see the improvement and effectiveness from performing stacking. However, it is unfortunate that due to a limitation of time and computing power we could not explore more models and technique. For example, we left out the use of Fast_Text embedding and did not perform stacking for all 6 labels. The common issue faced by stack models is also that the L1 models are highly correlated which could be attributed to the same embedding/vectorization used as well as the already high ROC_AUC score. Perhaps, for the L1 model we could use models with completely different embeddings/vectorizations so that it could complement each other better.

Next, a common issue faced by the final prediction is a high False Positive even though the True Positive is high as well. Therefore, if a moderator or Data Scientist would like to lower the False Positives while maintaining the True Positives could instead focus on optimizing F1 Score which is a balance of Recall and Precision instead.

Class	Toxic	Severe_Toxic	Obscene	Threat	Insult	Identity Hate
ROC_AUC Score	96.67%	98.77%	97.57%	99.14%	96.77%	98.08%
Average	97.83%					

The table above shows the overall average ROC_AUC test score. It is a result which can certainly be further improved.

Finally, we could also explore more on multilabel classification such as chaining as we have only currently tested on one model which has not been optimized too. In addition, perhaps the results from chaining could be used in the stack as well, which we feel will produce an even better result.

Deep Learning

We will use try out different types of vectorization and word embeddings as inputs to the three types of deep learning models: Convolutional Neural Networks (CNN), Long Short-term Model (LSTM) and Bidirectional LSTM. The types of vectorization that we are exploring are the CountVectorizer, GloVe, FastText and Word2Vec. We will then optimize the hyper parameters of the deep learning models to improve the performance of top models, and then select the best model among the CNNs and the best among LSTMs to to form Level 1 estimators for our stacking model. The output from the L1 estimators will be feed into the level 2 classifier before outputting the final results. The deep learning workflow can be visualised in the below figure (Figure)

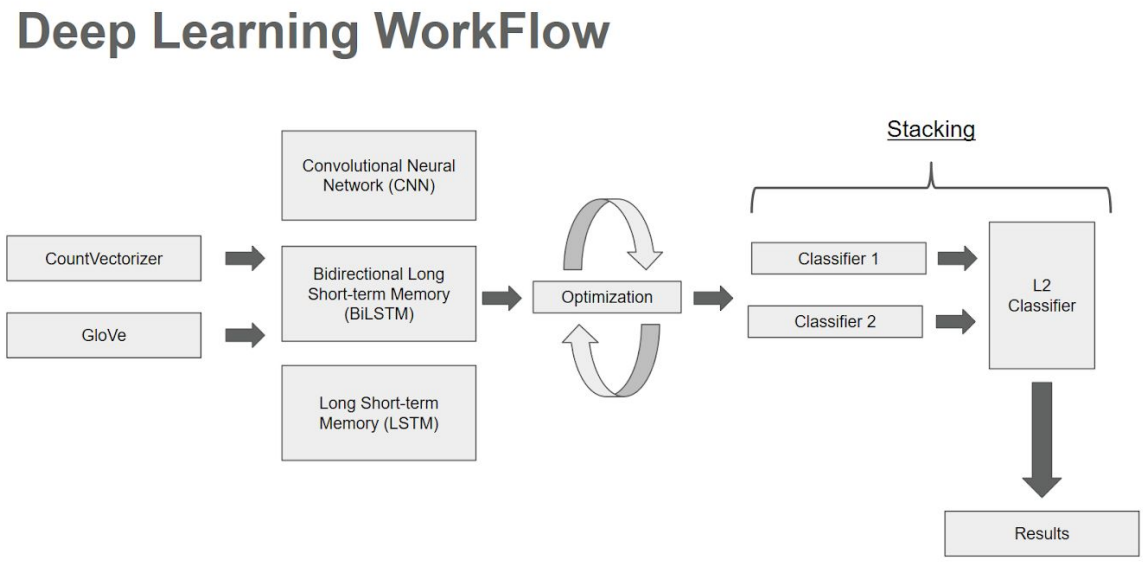


Figure 37: Deep Learning Workflow

Tokenization

These are the steps that needs to be performed so that we can convert each word of our vocabulary into a unique integer. Tokenizer is initialized in first step. Then fitting on the text will help us create a vocabulary so that each word is assigned with a unique integer. Then we convert the whole sentence of the comment into a sequence of numbers which are assigned by the tokenizer.

Padding on text sequences

Padding the sequence helps in making all the sentence of the same length. maxlen is the parameter which decides the length we want to assign to all the sentences. Padding is done by adding 0 on either the end of sentence or prior the sentence if the sentence is having length less than max length. This is also a parameter

which user can change, by default its prefix. If the length of the sentence is more than 100 then it is pruned which brings down the length to 100 (maxlen).[7]

Suitability of using LSTM and CNN for text classification

As LSTM is a variant of RNN, we will discuss the applicability of using RNN for text classification. RNN is a class of artificial neural network where connections between nodes form a directed graph along a sequence. It is basically a sequence of neural network blocks that are linked to each other like a chain. Each one is passing a message to a successor. This architecture allows RNN to exhibit temporal behavior and capture textual data since text is naturally sequential. Here, RNN is useful in text classification problem where sequential information is clearly important, because the meaning could be misinterpreted or the grammar could be incorrect if sequential information is not used.

CNN is a class of deep, feed-forward artificial neural networks where connections between nodes do not form a cycle. The result of each convolution will fire when a special pattern is detected. By varying the size of the kernels and concatenating their outputs, CNN models are able to detect patterns of multiples sizes (2, 3, or 5 adjacent words). Patterns could be expressions (word ngrams?) like “I hate”, “very good” and therefore CNNs can identify them in the sentence regardless of their position. Based on the above explanation, CNN is useful in detecting key phrases in what constitutes as toxic sentences. [8]

Comparison between LSTM and CNN

The relevance of using LSTM or CNN on the classification of toxic comment is dependent upon long semantics or feature detection. Based on our interpretation of the dataset, we identified that identifying semantic meaning of texts and feature detection is both important. In examples 1, 3 and 5, we believe these sentences are being flagged as toxic because of a single toxic word like 'cocksucker' or toxic expressions like 'stupid kid' and 'faggot pussy'. In such cases of feature detection, CNN works better. In examples 4 and 6, the sentences are considered toxic due to the semantic meaning that they convey like discrimination against religion and race. Also, in example 2, the entire text of 8 word length is deemed as a toxic expression. In such cases where the length of text and text semantic are important, RNN words better.

	Examples	Flagged expressions
1	cocksucker before you piss around on my work	Cocksucker
2	fuck your filthy mother in the ass dry	Fuck your filthy mother in the ass dry
3	would you both shut up you do not run wikipedia especially a stupid kid	Stupid kid
4	atheism is full of bias shit	Discrimination against religion

[7] https://realpython.com/python-keras-text-classification/
[8] https://towardsdatascience.com/text-classification-rnns-or-cnn-s-98c86a0dd361

5	fuck you block me you faggot pussy	Faggot pussy
6	kill all niggers i have hard that others have said this should this be included that racists sometimes say these	Discrimination against race

Figure 38: Flagged expressions of toxic comments

LSTM architecture

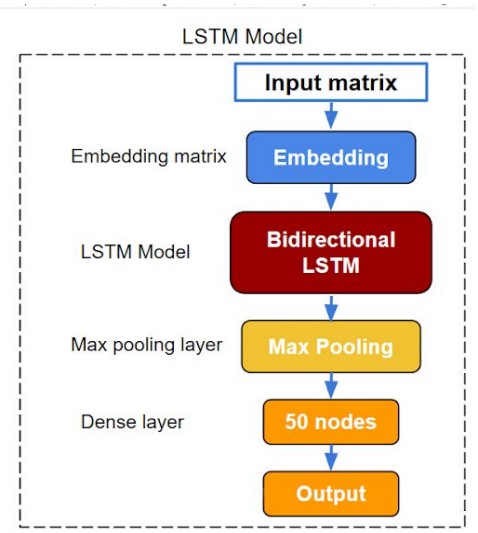


Figure 39: LSTM architecture

Here this embedding layer is important as this will help us in training of the sentences with their respective embeddings which we have assigned above. The first parameter is the size of our vocabulary. Second parameter is the output embeddings length which is 50 in this case as we used the 50 glove embeddings of each word. The length of each observation which is expected by the network is given by input_length parameter. We have padded all the observations to 50 hence we set input_length = 50. Weights parameter shows that the embeddings which we want to use is embeddings_matrix and it should not be altered hence trainable is kept false. If we want to train our own embeddings we can simply remove the weights and trainable parameter.

Building a LSTM model. LSTM networks are useful in sequence data as they are capable of remembering the past words which help them in understanding the meaning of the sentence which helps in text classification. Bidirectional Layer is helpful as it helps in understanding the sentence from start to end and also from end to start. It works in both the direction. This is useful as the reverse order LSTM layer is capable of learning patterns which are not possible for the normal LSTM layers which goes from start to end of the sentence in the normal order. Hence Bidirectional layers are useful in text classification problems as different patterns can be captured from 2 directions.

Following up, we use a pooling later to progressively reduce the spatial size of representation to reduce the amount of parameters and computation in the network. We use Global Max Pooling which is like an

ordinary max pooling layer with pool size equals to the size of the input (mins filter size +1, to be precise). As the input matrix contains two dimensions: the time-step dimension and the feature vector dimension, applying one-dimensional (1D) max pooling operation only on the time-step dimension outputs a fixed-length vector.[9]

Dense Layers is a fully connected layer that is each input node in one layer is connected to all the neurons in the next layer. It is basically helping us out in classification of the dependent variable with an activation function which tells the neuron when to be activated. Relu activation function is used here. Dropout layers is helping us to avoid overfitting. In dropout few neurons are randomly turned off. This is done in order to remove the dependency of neurons on each other. No neuron is particularly responsible for learning a specific feature. The argument 0.1 specifies that 10% of the neurons in this layer are going to the turned off.

The last layer is the dense layer with number of neurons equal to the number of classes of dependent variable (6 in this case). The activation used here is sigmoid because its a multi label classification problem (one observation can belong to more than one class). Sigmoid is used when we are dealing with a binary classification problem. Probability less than 0.5 is assigned to 0 class and more than 0.5 is assigned to 1 class.

Compiling the model is the final step to complete building the model. Optimizer is like the cost function (similar to gradient descent). Adam is one of the best performing functions used in deep learning models. Binary cross entropy is the loss function which checks the loss in predictions made by the model by calculating the distance between the predicted value and the actual value. We use categorical_crossentropy when dealing with multiclass classification problem and binary_crossentropy when dealing with binary classification. This is a special case for multilabel classification problem hence we are looking for probability of observation belonging to each class.

Accuracy is being used for checking the performance of the model. Accuracy is one of the evolutions which can be used to check if the model is overfitting or underfitting. If the training accuracy is very high as compared to testing accuracy we can see that the model is overfitting as it is not generalizing well on unseen data (test data). To avoid overfitting we can reduce the complexity of the model, add more data, add more regularization (dropout, batch normalization). If the training accuracy is much less than testing accuracy then the model is underfitting which means model is not able to capture the features so we need to train the model with more number of epochs.

Batch size is the parameter which defines how many observations are fed into the model at a time for training. More batch size would require more computational resources and memory for processing, while lower batch size will cause training and convergence speed to be slow and model to overfit easily. We use batch size of 32 and epoch size of 2.[10]

[9] https://www.aclweb.org/anthology/C16-1329/
[10] https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

From the plot of accuracy we can see that the model could probably be trained a little less as the trend for train accuracy intersects the trend for validation accuracy for the last few epochs. We can also see that the model has over-learned the training dataset, showing comparable skill on both datasets.
From the plot of loss, we can see that the model has slightly higher performance on training set than validation datasets. As these parallel plots start to depart consistently, it might be a sign to stop training at an earlier epoch. Below are the plots of accuracy and loss of lstm models with no embeddings, gloVe and FastText embeddings.

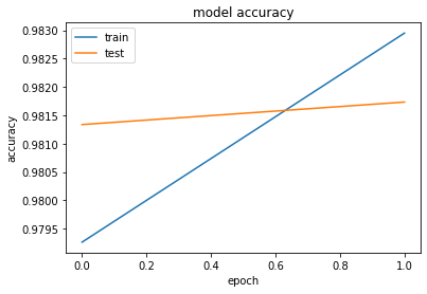
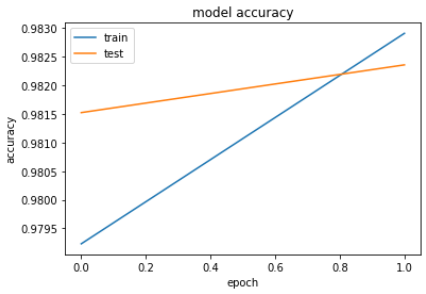
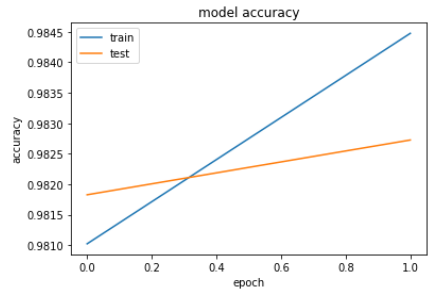
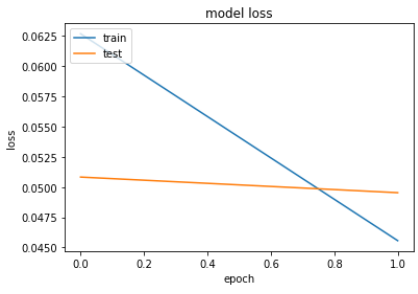
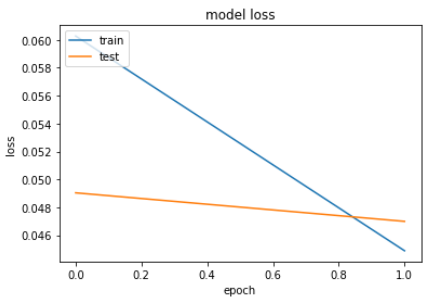
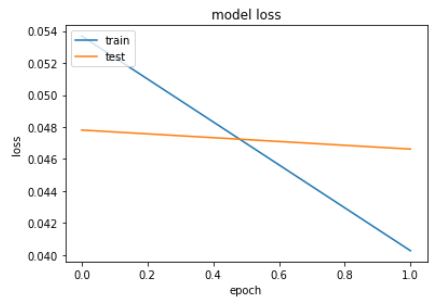
LSTM with no embeddings	LSTM with GloVe embeddings	LSTM with FastText embeddings
		
		

Figure 40: Accuracy and loss graphs of LSTM models

CNN architecture

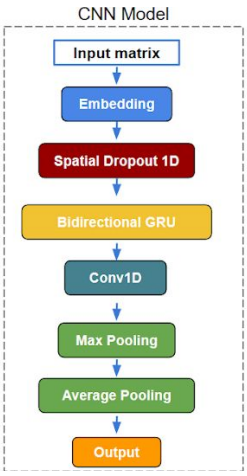


Figure 41: CNN architecture

The first layer is an embedding layer which helps us in training of the sentences with their respective embeddings which we have assigned above. Here, each word of a text document is transformed into a dense vector of fixed size.

The next layer is the spatial dropout 1d layer with 0.2 dropout rate. Here we use spatial dropout 1D layer instead of the standard dropout. The reason is to account for adjacent pixels correlation, especially in the early convolutional layers. Effectively, we want to prevent co-adaptation of pixels with its neighbors across the feature maps, and make them learn as if no other feature maps exist. With spatial dropout 1D layer, entire row of pixels are removed instead of just one.

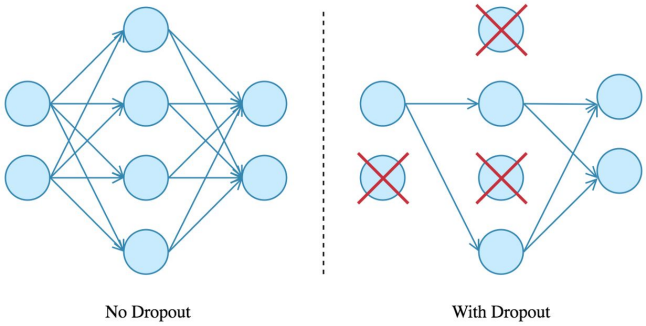


Figure 42: Dropout diagram

A bidirectional GRU layer is used next. The Grate Recurrent Unit (GRU) is a simplified LSTM cell structure that is designed to solve RNN gradient vanishing problem. Taking advantage of its simple cell,

GRU can get a close performance to LSTM with less time. The purpose of applying a GRU layer before the CNN Conv layer is that context feature cannot be incorporated very well by multi-size kernels. To combine a word and its context together, we first process words using a bidirectional GRU network in the following way. One word in a sentence can be learned from forward and backward twice in bidirectional structure, which can get more word representation features and avoid some gradient vanishing problem in long sentences.[11]

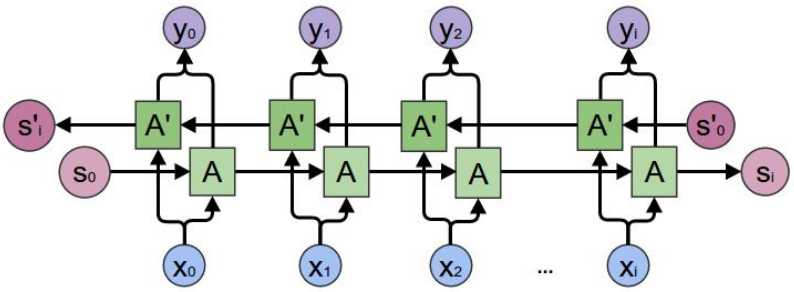


Figure 43: Bidirectional GRU

The main building block of CNN is the convolutional layer. Convolution is a mathematical operation to merge two sets of information. In our case the convolution is applied on the input data using a convolution filter to produce a feature map. We use kernel size of 3 to capture the trigram feature of text. Padding is also applied to preserve the size of the features maps, otherwise they would shrink at each later.

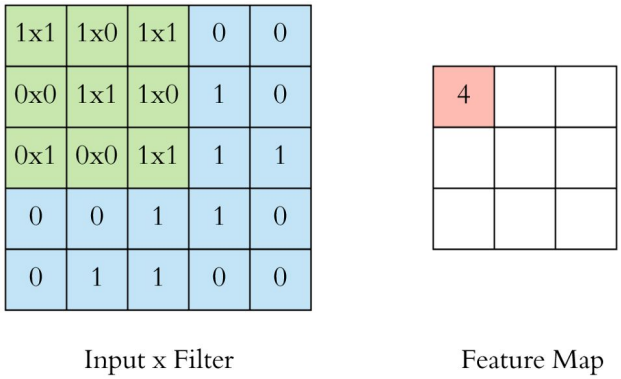


Figure 44: Convolutional layer with kernel size of 3

The last layer is the pooling layer which performs dimensionality reduction of the input feature images. This enables us to reduce the number of parameters, which shortens the training time and combats overfitting. Pooling layers make a subsampling to the output of the convolutional layer matrices combining neighboring elements. Here we use the max-pooling function, which takes the maximum value of the local neighborhoods. We also use average pooling function, which takes the average value of the local neighborhoods.

[11] https://www.anlp.jp/proceedings/annual_meeting/2019/pdf_dir/F3-1.pdf

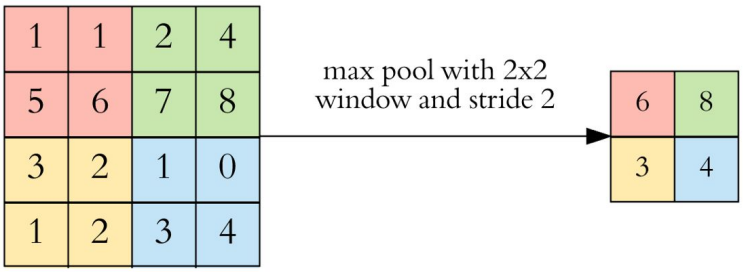


Figure 45: Max pooling

For our training of CNN models, we use batch size of 128 and epoch size of 4. From the plot of accuracy we can see that the model could probably be trained a little less as the trend for train accuracy intersects the trend for validation accuracy for the last few epochs. We can also see that the model has over-learned the training dataset, showing comparable skill on both datasets.

From the plot of loss, we can see that the model has slightly higher performance on training set than validation datasets. As these parallel plots start to depart consistently, it might be a sign to stop training at an earlier epoch. Below are the plots of accuracy and loss of lstm models with no embeddings, gloVe and FastText embeddings.[12]

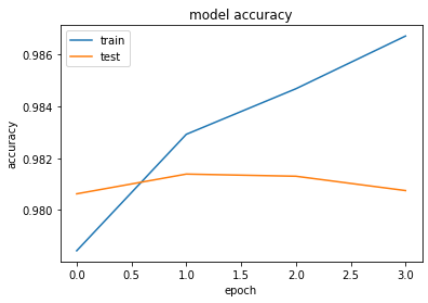
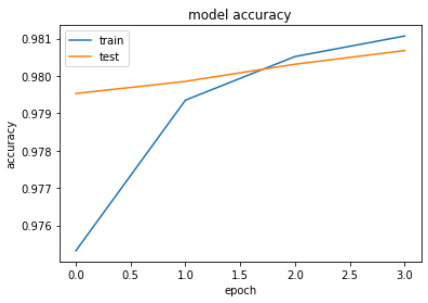
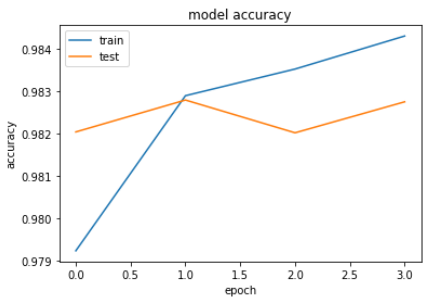
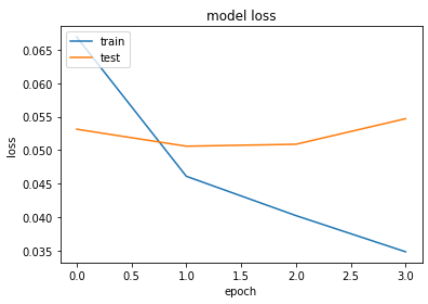
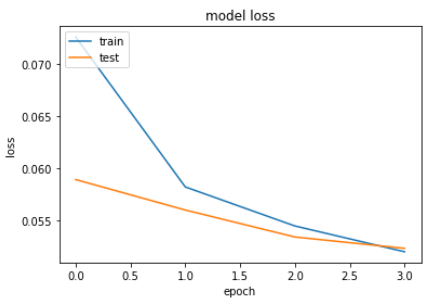
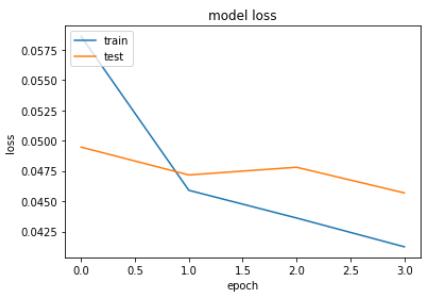
CNN with no embeddings	CNN with GloVe embeddings	CNN with FastText embeddings
		
		

Figure 46: Accuracy and loss graphs of CNN models

[12] https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Performance

The deep learning models give us a prediction probability of the various class labels. We hence use benchmark probability of 0.5 to allocate a binary outcome for each class. (e.g. a comment is considered to be toxic if the predicted probability of the toxicity class is more than 0.5) From there, we evaluate the models using ROC-AUC scores. Generally, the deep learning models perform rather decent in terms of ROC-AUC scores and the scores of the models can be visualised from the chart below.

Model	Lstm (no embedding)	Lstm with glove embedding	Lstm with fasttext	Cnn (no embedding)	Cnn with glove embedding	Cnn with fasttext
ROC_AUC score	0.720	0.722	0.736	0.753	0.742	0.808

Figure 47: ROC_AUC Performance of deep learning models

As expected, the CNN model using FastText embedding perform the best overall with 0.808 roc-auc score. Among the LSTM models, LSTM with FastText embedding perform the best overall. Generally, CNN models perform better than LSTM models. The reason could be that feature detection matters more in the detection of toxic comments whereby most of the time, toxic comments would contain a toxic phrase or expression. Also, the use of FastText embeddings would give more superior performance than other embeddings or without. This could be because of advantage FastText enjoy using subword embeddings as mentioned before.

Optimisation of Deep Learning Models

We try to further improve the performance of deep learning models by varying the benchmark probability values of 0.05, 0.075, 0.1, 0.3 and 0.5 whereby higher probability would mean stricter criteria of classification. We perform this optimisation on the LSTM and CNN model with FastText and the results are as follows.

Benchmark probability	**0.05**	**0.75**	**0.1**	**0.3**	**0.5**
LSTM roc_auc score	0.95	0.94	0.93	0.85	0.75
CNN roc_auc score	0.95	0.94	0.94	0.87	0.81

Figure 48: AUC_ROC Performance of different benchmarks

From the results, the roc_auc score decreases generally as benchmark probability increases. This is to be expected as with higher benchmark, the model would not be able to identify the toxic classes of comments. Even though benchmark of 0.05 gives us the best ROC-AUC score, we would not use that value because there is a high risk of overfitting to the training dataset. Instead we use benchmark of 0.1 because the ROC-AUC score is comparatively similar to that of 0.05 and 0.75 and also the ROC-AUC score drops drastically for the next benchmark value. Hence using 0.1 as benchmark is likely to improve the model's predictive ability while avoiding the problem of overfitting.

Next, we intend to optimise the training parameters like the batch size and epoch size to improve our models' performance. From the accuracy and loss plots, there are clearly signs that show the model is slightly overfitting to the training dataset. We believe the ROC-AUC score of our models can be slightly improve if we reduce the epoch size. However due to a limitation of time, we are not able to rerun the deep learning models with different epoch parameters as it takes a significant time.

Stacking

Given that CNN and LSTM models capture different types of features, it is possible that the toxic comments CNN fail to capture can be captured by LSTM. As such, there is relevance in using stacking to improve the overall performance of deep learning models. Among the deep learning models, we choose LSTM with FastText embedding and CNN with FastText embedding as the base classifiers and observe their correlation for different class labels.

Class labels	toxic	severe_toxic	obscene	threat	insult	Identity hate
LSTM accuracy	0.86	0.98	0.92	0.99	0.90	0.98
CNN accuracy	0.85	0.97	0.91	0.99	0.90	0.98
Correlation	0.82	0.77	0.86	0.70	0.85	0.74

Figure 49: Correlation of base classifiers

Generally, the correlation of the two LSTM and CNN models are low comparative to the high accuracy scores across the class labels. This low correlation is a good sign that the stacking model will perform better.

We then use logistic regression with C=0.01 as the meta classifier with LSTM and CNN prediction labels as inputs. The L2 classifier has a ROC-AUC score of 0.849 which is significantly higher than the score of LSTM model (0.75) and CNN model (0.81). Therefore, we use the stacking model as the final model for the toxic comment classification problem.

Application

With the creation of a decent model developed by our group, we believe that our efforts should not stop at model development. Hence, we decided to take our project one step further by putting our model into good practical application. One idea that resonated well with all of us is the integration of telegram bot and our machine learning model. There are many perks for the use of telegram bots be it for business or personal use[13]; bots are available around the clock, help save labour costs of having an administrator to moderate content in the telegram group, and enables the automation of repetitive work.

Primary Objective

The main aim of our telegram bot is to facilitate a healthier virtual environment by preventing the appearance of toxic content in designated telegram groups. Our bot's current capability includes:

- Detection of toxic content and labelling it into 6 categories; severe toxic, toxic, obscene, hate speech, identity hate, threat
- Immediate deletion of flagged messages
- Issue warnings to respective users
- Ban users who have exhausted 3 warnings

Infrastructure

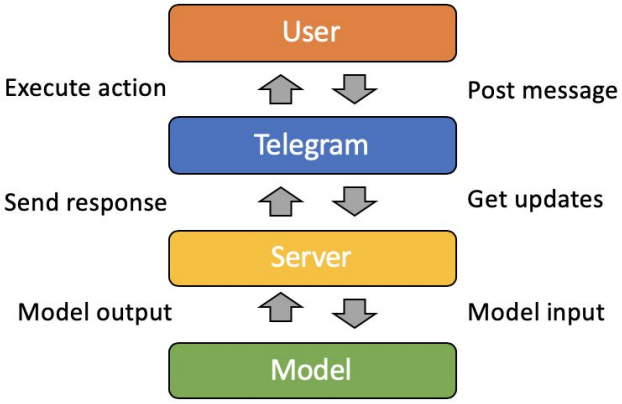


Figure 50: Communication flow for our Telegram bot

The above figure depicts a simple communication flow of how our telegram bot work. We can observe that the communication flow is bidirectional where an individual user will be the one who triggers the initial process when the user would post a message on the telegram group. On the other hand, our bot API will be running on our local hosted server where it would ping for updates every 0.5 second. If a message posted by any user is detected, our bot will extract the message and parse the text as input into our model. Our machine learning model would then predict the toxicity of the message based on the 6 different classes we

[13] https://chatbotsmagazine.com/top-5-benefits-with-using-chatbots-for-your-business-159a0cee7d8a

have trained our model on. Thereafter, our model would then produce an output and based on the output, if the user's content has been flagged as toxic, the relevant user would be given a warning or even banned from the group if he or she has already been worn three times.

Demo

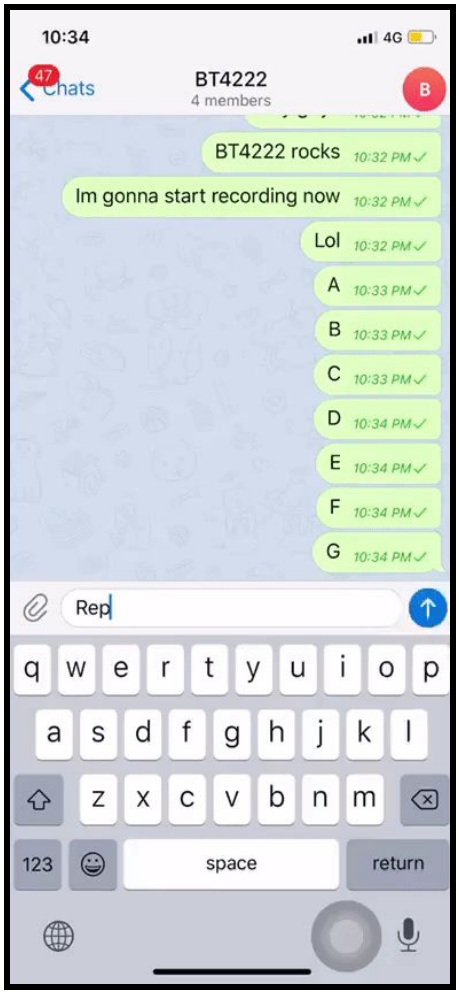


Figure 51: Demo on how our bot work

The above figure perfectly demonstrates how our telegram bot would work. Our user named Vincent initially posted: "Reply me guys fuckers!", our bot correctly identifies that it is toxic, contains obscenity and at the same time directed at other group of users hence an insult. Hence, first warning has been given to Vincent. Subsequently, Vincent posted two other toxic comments which our bot correctly identifies as toxic, obscene, insult and identity hate and in response the bot issued Vincent two more warnings totalling it to a count of three warning. Since Vincent has reached the maximum limit of three warnings, the bot then kicked him out of the telegram group which prevented Vincent from further posting on the group chat.

Moving Forward

Moving forward, we can further improve reducing the proliferation of toxic comments is by taking into account the context of a comment. Instead of using a single comment as input, the full discussion and other context, such as user history can be used. A motivation for this additional input is the way that humans read chat comments. Because of the page layout of social networks and news platforms and the chronological order of comments, early comments receive the most attention. To read later comments, users typically need to click through dozens of subpages. For this reason, we assume that the first few comments play a special role in setting the tone of further discussion as respectful or disrespectful. In other words, users who instigate toxic comments are likely to influence others to retaliate by giving toxic comments. As such, there's usefulness in identifying instigators and retaliators to penalise instigators more. (E.g. banning instigators from the the chat group while handing out warnings to retaliators)

An approach to identify instigators would be to construct a timeline of toxic comments, and track back the first user who started it. Furthermore, we can identify retaliators names' through Named Entity Recognition (NER) analysis of toxic comments. From the NER analysis, we may also identify victims of cyberbullying who are the targets of those comments. These victims are usually higher at risk of depression, loneliness and suicidal thoughts. As such, keeping a profile of cyberbullying victims can allow authorities to keep an eye on them and exercise preventative measures if necessary.

Other applications

Our toxic comment classification model can be widely applied in many social media platforms that facilitates communication between users. Apart from Telegram chatbots, they can be implemented in popular forums like Whatsapp, Reddit or Youtube. The toxic comments identified in these platforms can be useful in understanding the type and distribution of 'toxic users' in different platforms. For instance, the distribution of threats comments may be higher in Telegram and Whatsapp groups where users personally know other users in real life and have the potential to personally carry out the threats whereas distribution of identity hate comments may be higher on Reddit and Youtube where users identity are kept anonymous and they can freely spread hate messages without repercussions. With better understanding of users' toxic behaviour in various platforms, authorities can implement suitable preventive measures to curb toxicity online.

Currently, we have defined toxic comments in these five dimensions: Toxic, Severely toxic, Obscene, Insult and Identity hate. However, the characteristics of toxic comments can be versatile and not be restricted to these five dimensions. We identified a new area of toxicity in comments that pose a pressing need to resolve: Online Radicalization. Radical comments can pose a danger to society because they can influence acts of terrorism. We feel that radical comments can have overlapping characteristics with comments that have severely toxic, insult and identity hate components. As such, we can investigate this

further and see how much of radical comments have components in these three dimensions. If a sufficiently high number surfaced, we can classify a comment as radical if a comment has all these three types of classifications. Otherwise, we can create a new 'Radical' dimension and train our model to identify them.

Conclusion

Users posting toxic comments on online community is a serious problem that threatens the freedom of speech. Not only are toxic comments hurtful to the individual, they also set the tone of the community and inspire a toxic culture online. To solve this pressing issue, our team created a toxic comments classifier model to detect toxic comments and implemented it in a Telegram bot to flag out toxic users. When building the model and exploring the topic, we incorporated concepts and techniques taught in class in the phases of text preprocessing, data exploration, visualisation, feature engineering, modelling and optimisation.

Finally, we would like to thank Prof Qiao Dandan and TA Siva for their continual guidance throughout the semester and for imparting their knowledge and giving us the basis of the concepts used in this paper. It has been an insightful and enjoyable module. We certainly gained a lot during the course of the module and through this project.

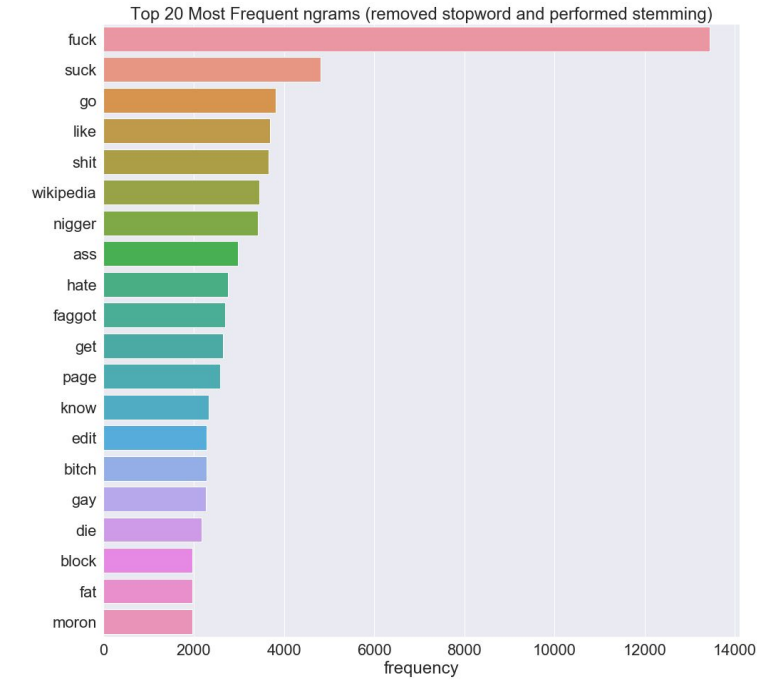
Appendix

Full list of features definition from Kaggle training and testing dataset

FEATURE	DATA TYPE	DESCRIPTION
id	Character	Unique ID of the Accident
comment_text	Character	Raw wikipedia comment posted by users
toxic	Number	1 if comment contains toxic content else 0
severe_toxic	Number	1 if comment contains severely toxic content else 0
obscene	Number	1 if comment contains obscene content else 0
threat	Number	1 if comment contains threat content else 0
insult	Number	1 if comment contains insult content else 0
identity_hate	Number	1 if comment contains identity hate content else 0

Top Unigrams/Bigrams and Wordcloud of each Toxic Category

Toxic



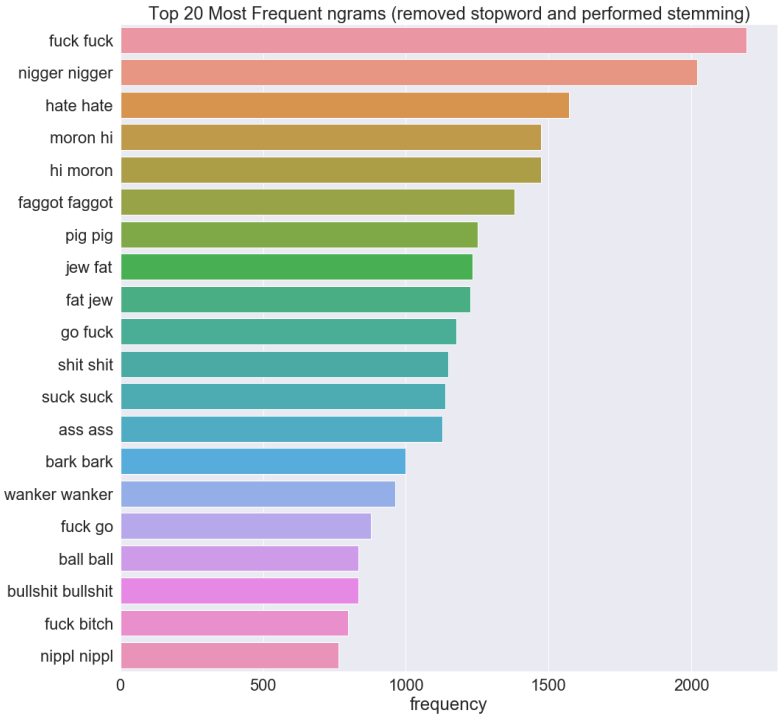
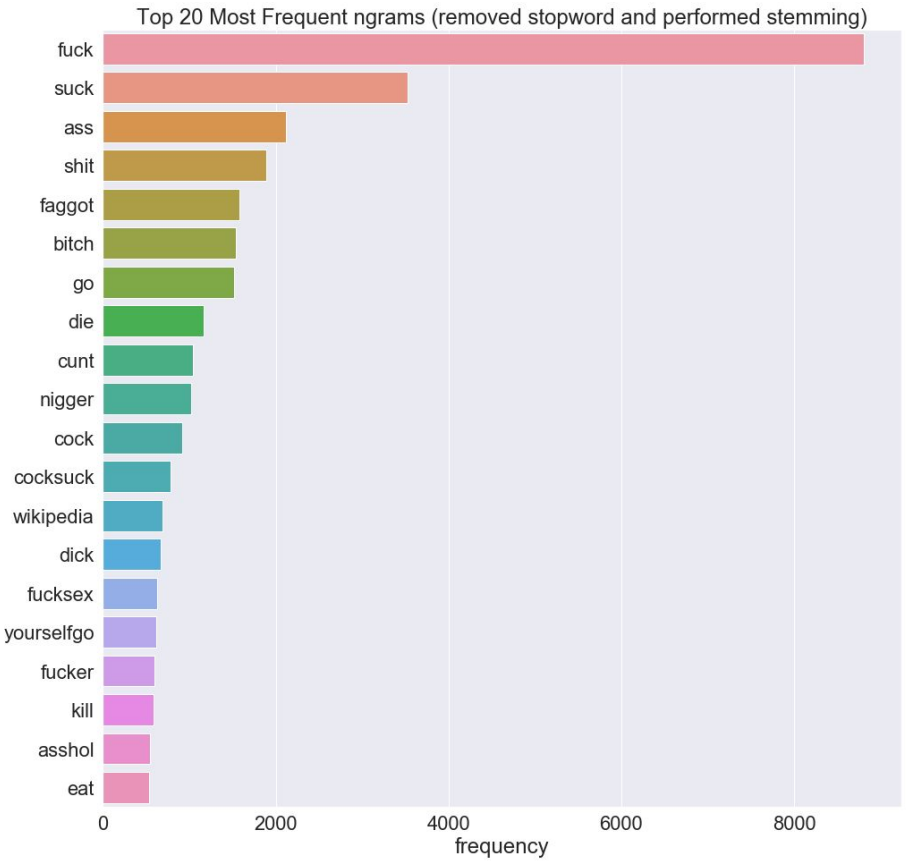


Figure 51: Top Ngram Counts

Figure 52: WordCloud

Severe Toxic



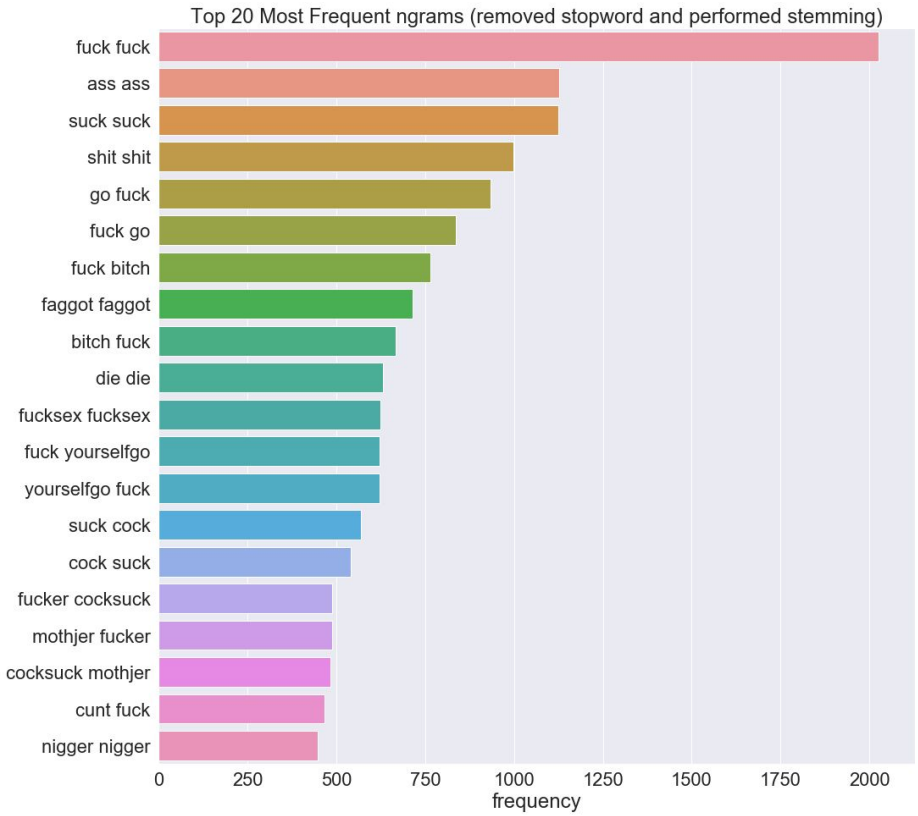


Figure 53: Top Ngram Counts

Figure 54: WordCloud

Obscene

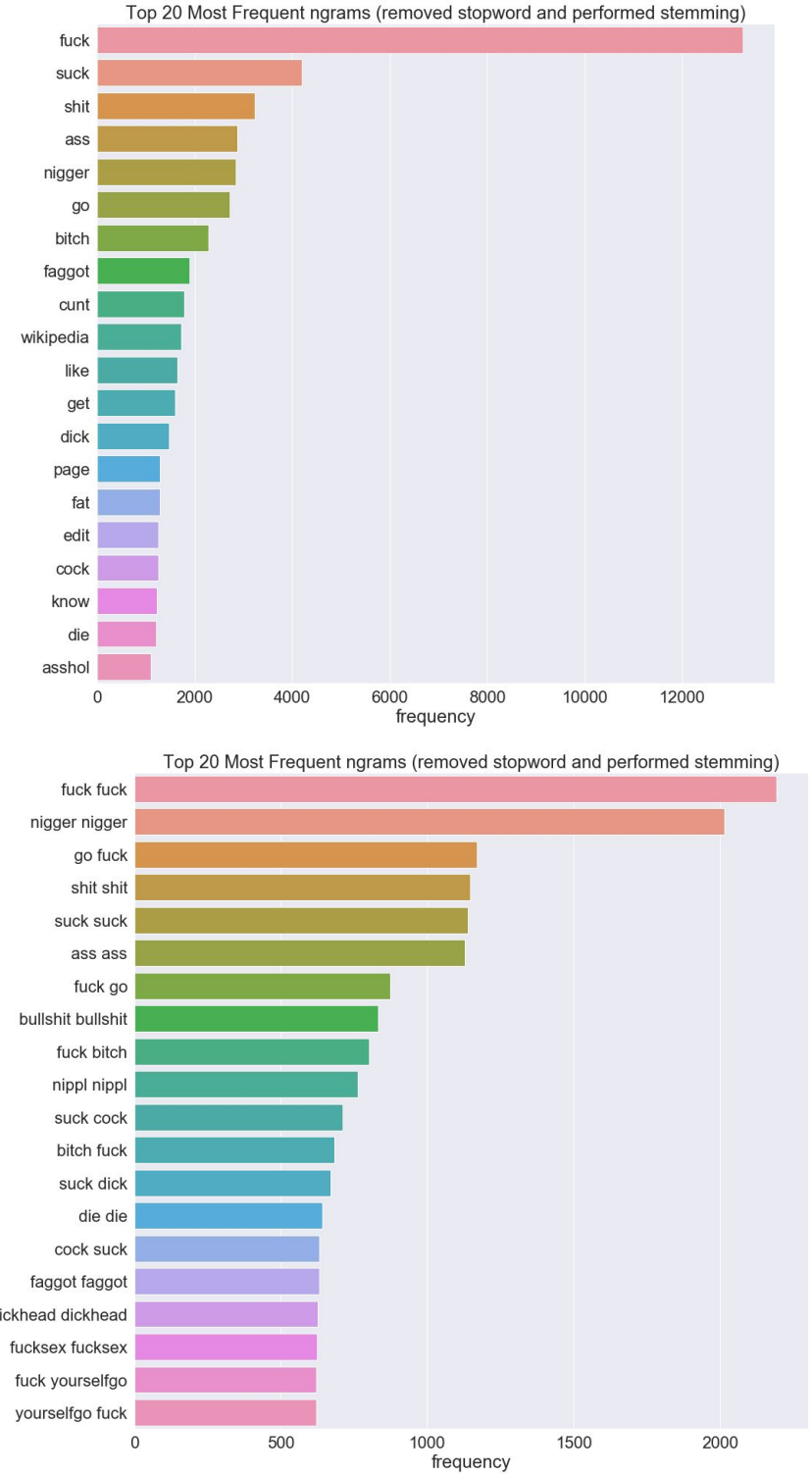
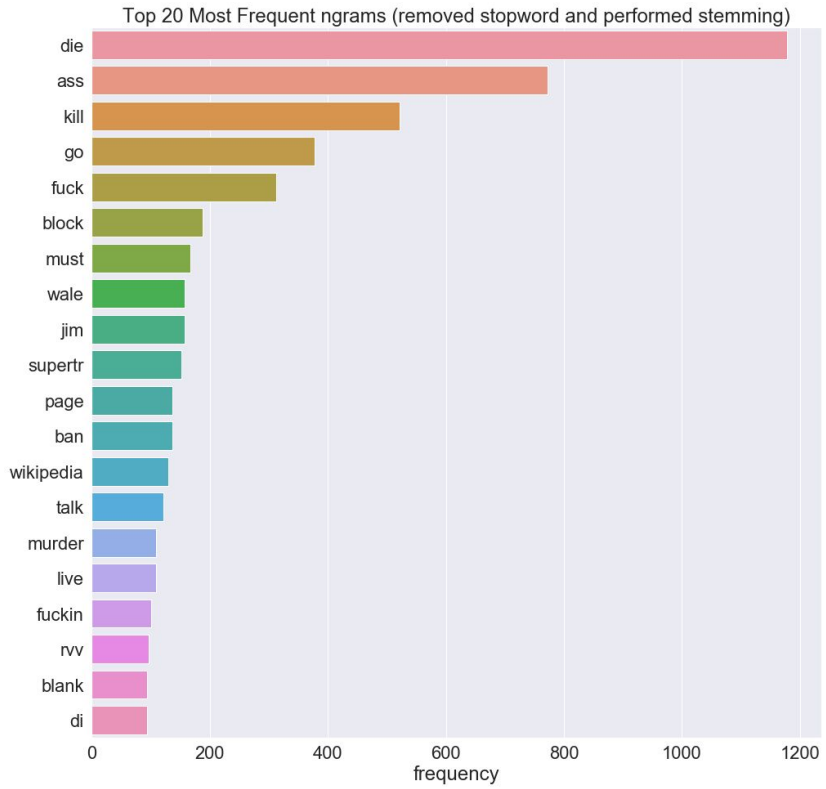


Figure 55: Top Ngram Counts

Figure 56: WordCloud

Threat



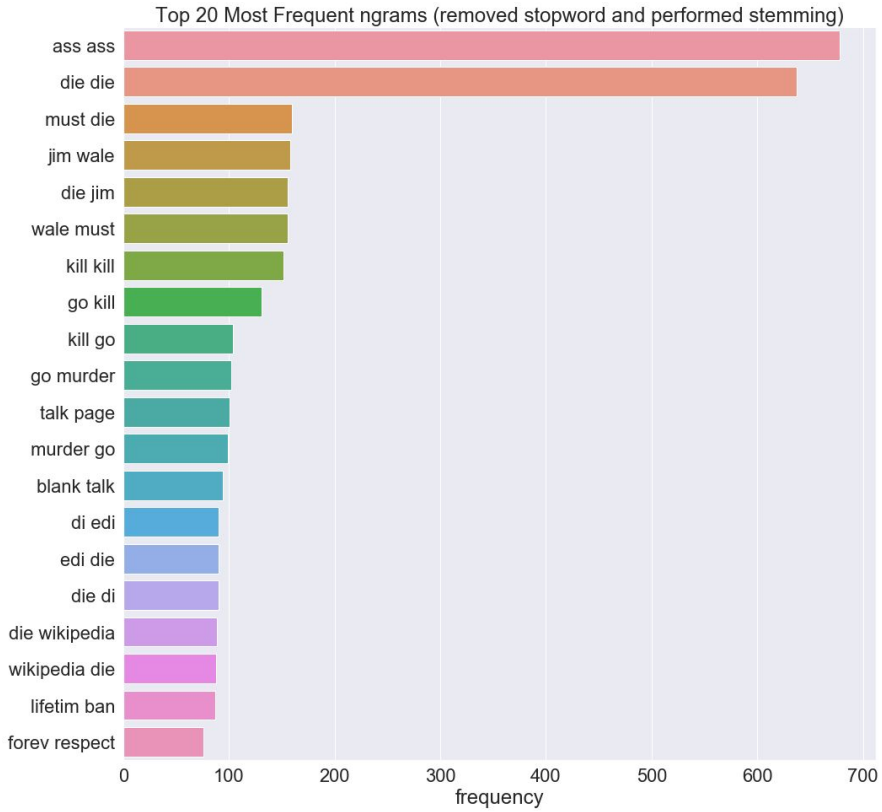


Figure 57: Top Ngram Counts

Figure 58: WordCloud

Insult

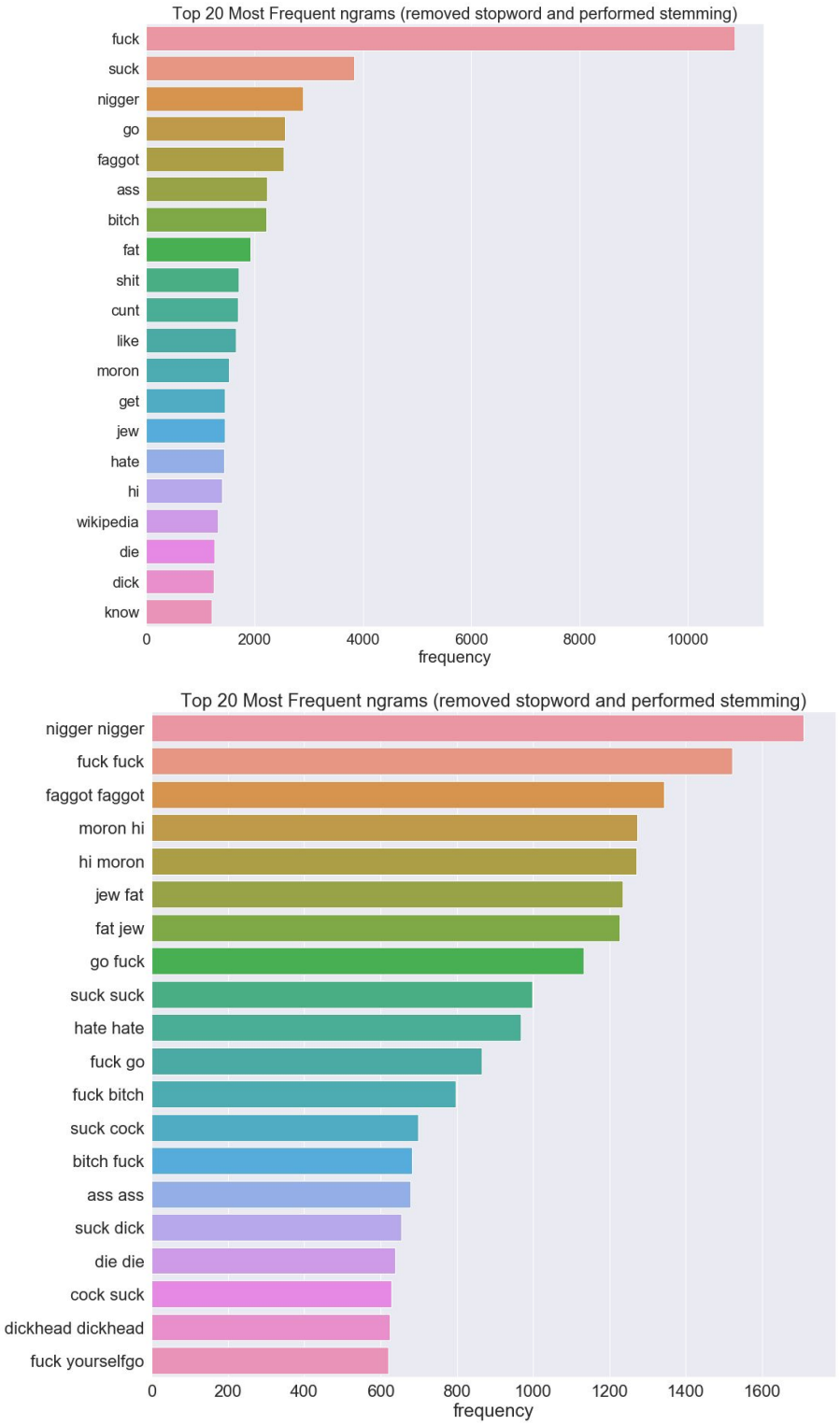
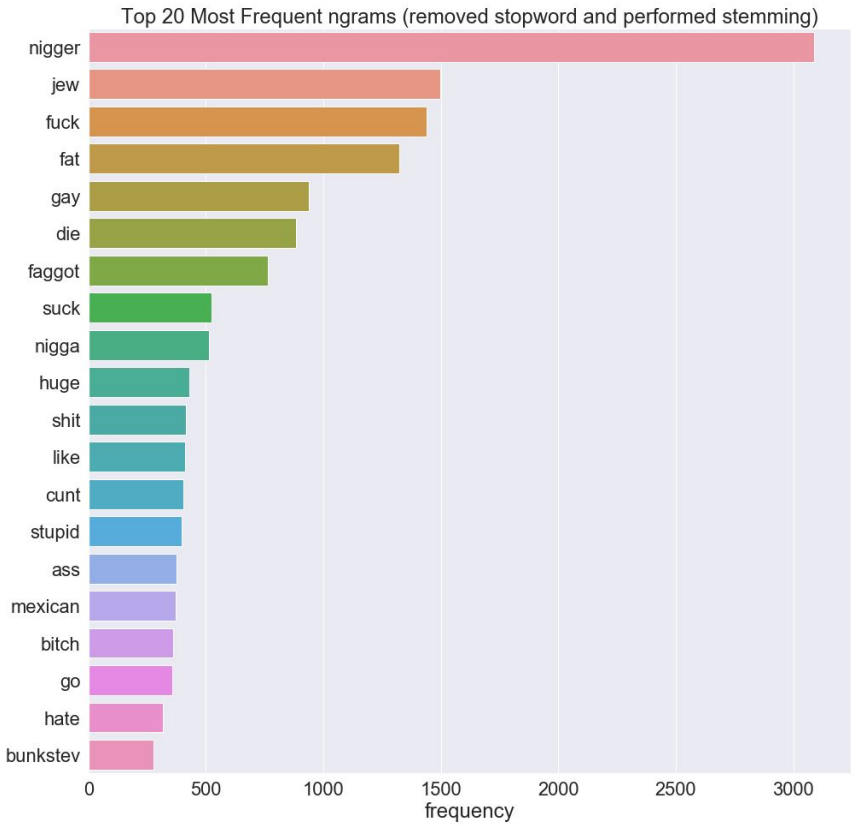


Figure 59: Top Ngram Counts

Figure 60: WordCloud

Identity Hate



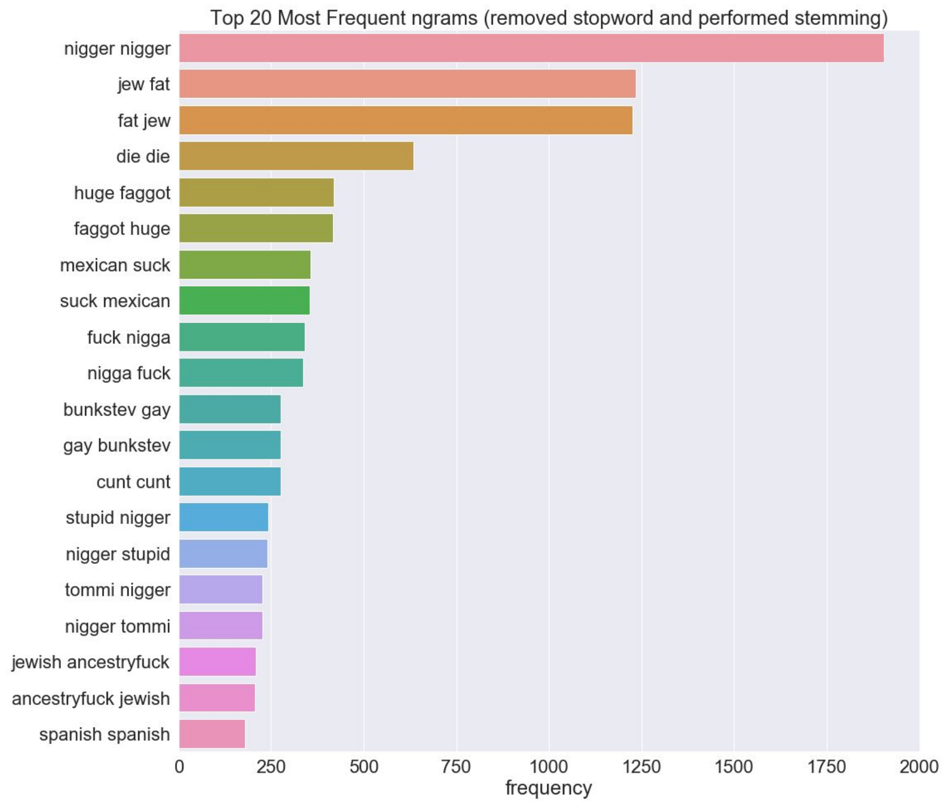


Figure 61: Top Ngram Counts

Figure 62: WordCloud

References

Duggan, M. (11 July 2017) Online Harassment 2017. *Pew Research Center.* Retrieved from https://www.pewresearch.org/internet/2017/07/11/online-harassment-2017/

Wang, Z., Sugaya, S. & Nguyen, D. P. T. (2019). Salary Prediction using Bidirectional-GRU-CNN Model. *Association for Natural Language Processing*. Retrieved from: https://www.anlp.jp/proceedings/annual_meeting/2019/pdf_dir/F3-1.pdf

Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., & Xu, B. (2016). Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. *The 26th International Conference on Computational Linguistics: Technical Papers*. Retrieved from: https://www.aclweb.org/anthology/C16-1329/