Washington State University

Toxic Comment Classification

Alex Tuck

Computer Science 315

Professor Jana Doppa

April 28, 2019

Introduction

All over the internet, people communicate using the written word more than anything else. On websites like Facebook, Twitter, and various discussion boards, people write posts to share their thoughts with others. In the past, a forum moderator would be assigned the task of policing forums and censoring inappropriate posts to maintain order and promote healthy discussion within the comments. Today, sites like Reddit commonly have thousands of comments on different posts, and new posts are constantly being added, making manual moderation a full-time job. With the rise in popularity of online communities, automated post moderation is tool to be considered for maintaining order online.

The goal of this project is to develop a comment sentiment analyzer that will be able to sort through comment sections and classify comments into categories 'toxic, 'severe toxic', 'obscene', 'insult', 'threat', and 'identity hate'. I want to determine how effective an automated moderator is in searching for inappropriate comments when compared to a human moderator. Are there key characteristics that distinguish one comment sentiment from another? Can an automated moderator do so well that human oversight is unnecessary? Or, should a human be validating the results?

There are several difficulties that come with this task. Language is open to interpretation and subtleties, meaning two comments with the same labels could have very different sentence structures and vocabulary. This leads me to the next challenge: the dataset I used was hand-labelled by several volunteers (Conversation AI Team, 2018). A comment labeled as 'identity hate' by one volunteer may not have been labeled similarly by another, leading to the dataset not being totally consistent. The comments were also produced by humans, meaning spelling errors and other sources of noise are present in the data. Additionally, most comments are civil and appropriate, and inappropriate comments are a small minority of the data. Categories of inappropriate comments also have varying breadth. For example, it is relatively easy to classify a comment as 'identity hate' off the presence of racial slurs. On the other hand, the criteria for classifying a comment as 'severe toxic' is broad and not so clear, leading to differences in how volunteers may have labelled the data.

I approached classifying comment sentiment by first trying to standardize the comments by removing repeated characters, punctuation, and digits, and removing suffixes from words. Next, I built vocabularies of the most frequent words in each sentiment category to be used as binary features. I trained six binary classifiers on the data, one for each sentiment category. Classifiers were trained to identify comments in a sentiment category against comments not belonging to any sentiment category, otherwise known as 'clean' comments.

At the end of the process I had two sets of six binary classifiers, one each for 'toxic, 'severe toxic', 'obscene', 'insult', 'threat', and 'identity hate' comments. One set of binary classifiers was made using linear support vector machines, and the other set was made using random forest classifiers. The average recall and precision of the support vector machines was

70.5% and 41.2%, respectively. Random forest classifiers had an average recall of 77.4% and average precision of 35.7%. However, both types of classifier for 'threat' and 'severe toxic' severely underperformed in the precision category. Precision of other categories was typically between 50% and 60%.

Data Mining Task

Massive amounts of text data are being generated on social media sites every moment, and manually validating the sentiment of each post or comment is an impossible task. Most posts are harmless, but a minority of posts are examples of hate speech, insults, or threats that do not contribute to overall discussion and health of an online community. The need for an automated moderator to wade through the hordes of good posts in search of the bad ones is apparent. Can a computer be taught to identify inappropriate comments as well as a human moderator could? Can it differentiate between the different sentiments of inappropriate comments?

To solve this problem, I used the Toxic Comment Classification Challenge dataset (Conversation AI Team, 2018). This dataset consists of over 150,000 comments scraped from Wikipedia pages, and labels regarding the sentiment of those comments. Labels are binary vectors with entries corresponding to the sentiment categories 'toxic', 'severe toxic', 'obscene', 'threat', 'insult', and 'identity hate'. My goal was to build an ensemble of binary classifiers that would make a prediction on the sentiment of a new, unseen comment.

The dataset provided several challenges to solving the task. Comments are organic in nature and are prone to large quantities of human error, such as spelling mistakes. It is not rare to see a comment with lots of spammed characters. Is it better to ignore spammed characters, or to try to glean information from them? There are also several examples of Wikipedia editor's usernames in the data. What is the right way to deal with usernames? And, what is the overall best way to clean comments so a model can be trained on them, but information is not lost? Model selection is another challenge to solving the task. Is there a model better suited to my needs I should try? Or, is it better to optimize a model that gave adequate initial results?

Technical Approach

My approach to solving the toxic comment classification problem can be broken down to three main steps.

- Preprocessing and data cleaning
- Feature selection
- Model training and optimization

Preprocessing and data cleaning

Because the comments are organic, they need to be cleaned as much as possible before they can be used to train a classifier. My approach to cleaning a single comment was to use

several functionalities from the Natural Language Toolkit (Bird, 2009). NLTK provides a regular expression tokenizer and an implementation of Porter Stemmer, an algorithm for removing common endings to English words. Using Porter Stemmer and the regular expression tokenizer, I hoped to clean words in a way such that two words with different ending would be counted as the same word. I also removed any occurrences of punctuation and digits and limited the number of subsequent repetition of letters to two. If there is an English word with three characters is a row, it is an exception to the rule.

Algorithm 1: Comment cleaning

```
tokenizer=RegExpTokenizer('\w[a-z]{1,20}\w')  #Capture chars only, limit token length
pattern="(.)\1{2,}" #Regular expression to find 2 or more repeated characters

FOR EACH comment:
    Remove digits and punctuation from comment
    tokens = tokenizer(comment)
    cleaned_comment = ' '

    FOR EACH t IN tokens:
        w = pattern.sub("\1\1", t) #Replace occurrence of 2+ chars for 2 chars
        s = PorterStemmer(w)
        cleaned_comment.append(s)

    RETURN cleaned_comment
```

Each comment is processed according to the above. The result is a new text string without digits or punctuation, character repetition is limited to two, and words are stemmed. The cleaned comments are added to a new column in the data. Both the test set and train set were preprocessed in this manner.

After preprocessing the comments, I made a new sentiment category called 'clean'. A comment is labelled 'clean' if it does not belong to any sentiment category.

Feature Selection

Now that the comments have been somewhat standardized, I need to select features. Intuitively, I thought using the *N* most frequent words in a category as features to train on was a good approach. This would capture key words that appear most often in a sentiment category. There are several issues with this approach. The first is how to define frequent. If frequency of a word is the raw count of how many times it occurred in a sentiment category, comments with repeated words will float to the top. Here is an example of such a comment:

…User:NHRHS2010 is a homo like mitt romney is.
User:NHRHS2010 is a homo like mitt romney is.
User:NHRHS2010 is a homo like mitt romney is….

This particular comment repeats for several hundred lines, and results in the words 'mitt' and 'romney' having high frequency in the 'identity hate' category, which this comment belongs to. To overcome this obstacle, I decided to count the occurrence of each word in a comment only once. Each occurrence is divided by the number of comments in the category. The end result is the probability a word is found in any given comment from the category. This way, words that are common among all comments in a category rise to the top.

The second issue with using raw frequency is that common English words that do not add relevant information are also captured. I took two approaches to avoid this. Scikit-learn provides a list of common English words, known as stop words (Scikit-learn, 2011). I stemmed the stop words and did not count occurrences of stemmed stop words. The other approach was to count word frequencies in the 'clean' comments. To make the vocabulary for a sentiment category, I contrasted the set of *N* most frequent words in the sentiment category with the *N* most frequent words in the 'clean' comments to find the most frequent words unique to the sentiment category.

Algorithm 2: Find word frequencies

```
n_examples = LENGTH(sentiment_category)

FOR stemmed_comment IN sentiment_category:
        Stemmed_tokens = tokenize(stemmed_comment)
        Stemmed_tokens = set(stemmed_tokens)#Use set so words only occur once

        FOR tok IN stemmed_tokens:
                IF tok NOT IN stemmed_stop_words:
                        word_frequency[tok] += 1/n_examples #Increment count
```

Algorithm 3: Generate vocabulary using N most frequent words

```
clean_frequency = find_word_frequencies(category = 'clean')
SORT clean_frequency BY most frequent words
clean_frequency = set(clean_frequency[0:N])#Make a set of N most frequent words

FOR c IN sentiment_categories:
    c_frequency = find_word_frequencies(category = c)
    SORT c_frequency BY most frequent words
    c_frequency = set(c_frequency[0:N])
    unique_to_c = c_frequency – clean_frequency #Words in c but not clean
    SAVE unique_to_c AS c_features
```

In practice, I used N=300 words. See *Table 1* in the appendix for resulting vocabularies. I used these vocabularies as binary features to train the classifiers. Vocabularies range from 122 to 182 words.

Model Training and Optimization

The models I decided to use were support vector machine with a linear kernel and random forest classifier (Scikit-learn, 2011) I chose these models because we learned about them in Prof. Jana Doppa's data mining class, and I wanted to see how well they were able to classify toxic comments. They are also relatively fast to train on large numbers of examples.

The training data for each sentiment category was the set of all comments in that category, as well as all comments in the 'clean' category. I chose to split the data in this manner because many comments belong to more than one sentiment category, and this would provide the clearest boundary between categories. The end result is a binary classifier that specifies a comment as 'clean' or belonging to a sentiment category.

To find an optimal classifier random forest classifier and support vector classifier for each category, I ran three-fold grid search cross validation for each model/category pair. Given a space of parameters for a model, grid search looks for the combination of parameters that maximizes a given scoring function. I chose 'weighted precision' as my scoring function. Weighted precision calculates a precision score that accounts for imbalances in the dataset. I was most interested in precision because I do not want a classifier to mistake 'clean' comments for inappropriate comments.

Table 2: Grid search parameter space

Linear SVM	**Random Forest Classifier**
Penalty : [‘l1’,’l2’]	Criterion : [‘gini’, ‘entropy’]
Tol : [0.1, 0.001, 0.00001, 0.0000001]	Min samples split : [2 to 38, by 4]
C: [1 to 10, by 1]	Min samples leaf : [1 to 17, by 3]
Fit intercept : [True, False]	Max features : [None, ‘auto’]
Class weight : [None, ‘balanced’]	Class weight : [None, ‘balanced’]

The parameter space for grid search was chosen arbitrarily. I did not know which parameters would increase performance, and given my limited computing resources, I was not able to test all combinations of parameters I would like to. Suffice it to say there are probably better models outside of the search space. The optimal models are listed below.

Table 3: Optimal linear SVM classifiers

Linear SVM	**Optimal Settings**
Toxic	'C': 1, 'class_weight': None, 'fit_intercept': True, 'penalty': 'l2', 'tol': 0.001
Severe toxic	'C': 9, 'class_weight': None, 'fit_intercept': True, 'penalty': 'l1', 'tol': 1e-05
Obscene	'C': 3, 'class_weight': None, 'fit_intercept': True, 'penalty': 'l2', 'tol': 0.1
Threat	'C': 1, 'class_weight': 'balanced', 'fit_intercept': False, 'penalty': 'l2', 'tol': 0.001
Insult	'C': 4, 'class_weight': None, 'fit_intercept': True, 'penalty': 'l2', 'tol': 0.001
Identity hate	'C': 9, 'class_weight': None, 'fit_intercept': True, 'penalty': 'l1', 'tol': 0.1

Table 4: Optimal random forest classifiers

RFC	**Optimal Settings**
Toxic	'class_weight': None, 'criterion': 'gini', 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 30
Severe toxic	'class_weight': 'balanced', 'criterion': 'entropy', 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 10
Obscene	'class_weight': None, 'criterion': 'entropy', 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 10
Threat	'class_weight': 'balanced', 'criterion': 'gini', 'max_features': 'auto', 'min_samples_leaf': 13, 'min_samples_split': 10
Insult	'class_weight': None, 'criterion': 'gini', 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 14
Identity hate	'class_weight': None, 'criterion': 'entropy', 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 26

Evaluation Methodology

The data for this project came from the Toxic Comment Classification Challenge on Kaggle.com (Conversation AI Team, 2018). The dataset consists of comments scraped from Wikipedia and is split into a training set and a testing set. The training set has 159,554 comments, and the testing set has 153,164 comments, of which 63,529 are labelled. Comments are labelled by their sentiment: 'toxic', 'severe toxic', 'obscene', 'threat', 'insult', and 'identity hate'. Inappropriate comments have a low incidence rate: only 16,220 comments of the train data belong to at least one of the six categories. These 16,220 comments are even further subdivided when considering individual categories. Because of the organic nature of the dataset, several challenges arise from working with it. Spelling mistakes are common, as well repeated characters or words. Usernames are occasionally mentioned in the comments, and they may contain non-alphabetical characters. Additionally, the comments were hand labelled by a team of volunteers (Conversation AI Team, 2018). The categories are up to interpretation, and volunteers may have differing opinions as to what characterizes a toxic comment.

Table 5: Train and test comment breakdown

Dataset	Total	Clean	Toxic	Severe Toxic	Obscene	Threat	Insult	Identity Hate
Train	159554	143334	15289	1564	8445	478	7875	1403
Test	63529	57289	6087	367	3688	211	3424	712

With the characteristics of the dataset in mind, I knew a general accuracy score was a poor metric to evaluate classifiers by because of the low incidence of inappropriate comments. If all comments were classified as appropriate, the model would achieve very high accuracy, but would be doing its job badly. I decided to use precision, or the proportion of correctly identified positives, as my metric for evaluating models. A high precision means the classifier does not mistakenly classify an appropriate comment as inappropriate. If a comment moderator bot were to be deployed, high precision is a desirable quality to have because it will not mistakenly punish users who make appropriate comments.

Results and Discussion

After training each of grid search's optimal models on the entirety of the training data, and testing on the entirety of the testing data, I got the following results.

Table 6: Precision and recall of optimal models from grid search

Category	Model	Train Precision	Train Recall	Test Precision	Test Recall
identity_hate	rf	0.936	0.665	0.411	0.585
identity_hate	svm	0.877	0.556	0.407	0.567
insult	rf	0.906	0.781	0.507	0.709
insult	svm	0.890	0.675	0.561	0.651
obscene	rf	0.931	0.852	0.557	0.777
obscene	svm	0.915	0.705	0.597	0.640
severe_toxic	rf	0.363	0.961	0.088	0.953
severe_toxic	svm	0.929	0.826	0.299	0.855
threat	rf	0.046	0.930	0.032	0.933
threat	svm	0.040	0.951	0.026	0.928
toxic	rf	0.882	0.649	0.548	0.690
toxic	svm	0.865	0.546	0.583	0.594

One of my goals for this project was to optimize precision with respect to inappropriate comments because of their low incidence rate. For most sentiment categories, test precision is approximately 50%. I think this is a good proportion considering that in the best case 'clean' vs. another category, 'toxic' comments are outnumbered almost 10:1. However, 'threat' and 'severe toxic' have significantly lower precision on test data. This is likely because they are the least represented categories in the testing data and are severely outnumbered by 'clean' comments. There are 271 'clean' comments for every 'threat', and 156 'clean' comments for every 'severe toxic'.

Overfitting seems to be an issue across all categories. I am thrilled that some models were able to achieve high precision and recall on the training data, but that did not always translate to testing data. However, severe decreases usually came in the precision category. Recall remains relatively consistent between testing and training data, and in some cases, such as both models of 'toxic', even increases. While the models cannot always tell the difference between inappropriate comments and 'clean' comments, as evidenced by the sometimes-subpar precision, there is a silver lining in the fact that many classifiers do not mistake inappropriate comments for 'clean' comments, as shown by many recall rates being above 70%.

It's a toss up between which model is better. In some cases, such as 'identity hate', random forest and linear SVM perform very similarly on testing data. In other cases, such as 'toxic', random forest has a 10% better recall rate, but SVM has 5% better precision. This could be because of the limited search parameters I used for grid search. Its possible that model settings exist outside of the search space that drastically outperform the current models. Better

models for ‘threat’ and ‘severe toxic’ may be found outside of the search space. For this reason, grid search is both a blessing and a curse. It returned optimal models for my specified settings, but searching for better optima over a larger parameter space would take way to long for the scope of this project.

Lessons Learned

Quality feature selection is key. One of my initial approaches to the problem was to use the *N* most common words in a sentiment category as features for a binary classifier. This approach performed poorly for a number of reasons.

- It did not effectively separate words common to the sentiment category and words common to the ‘clean’ comments
- It did not clean the words, so different word tenses were treated differently
- Repeated words in comments were each counted as an occurrence of the word, so a word very frequent to one comment would be frequent to the category

After I refined and cleaned the features, the results I got from the classifier were instantly improved.

Be aware of the quirks of the dataset. Going hand-in-hand with feature selection, being aware of the characteristics of the data can help you select better features through domain knowledge. Through working with the toxic comment dataset, I discovered some of the sources of noise in the data, such as spammed characters or words. If I put more time into exploratory data analysis, I could have identified them sooner and selected better features.

Model optimization is a time-consuming process. In Random Forest Classifiers (and many others), the space of possible models is combinatorial, with parameters like max depth, minimum samples to split a node, minimum samples on a leaf, number of estimators, etc. The optimal models returned by grid search are local optima according to the parameter space I specified. A better, global optimum may exist for each classifier, but searching for it is unreasonable given my computing resources.

Define your own version of success. Building a successful classifier is relative to the dataset. In the toxic comment dataset, the incidence rate of inappropriate comments is low compared to appropriate comments. Building a classifier with precision and recall is ideal, but are there other, easier goals you can achieve? When I thought about how I wanted this classifier to perform, one of my preferred characteristics was to have a high rate of precision when classifying inappropriate comments. I did not want acceptable comments to be mistakenly classified, and I was willing to let some inappropriate comments slip through to achieve this.

In hindsight, I wish I had refined my features more. Using key words frequent to a sentiment category works, but it does not capture the sentiment of combining words into

phrases. Misspelled words are also common in the data, and I did not know where to start with fixing that. I also wish I experimented more with using different levels of training data. I used all of it when training the classifiers, and this may have led to overfitting. The last thing I wish I did differently was use randomized search cross validation to select better models (Scikit-learn, 2011). By using randomized search, I could have gotten a better notion of promising parameters to use for the models.

Bibliography

Bird, Steven, Edward Loper and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

Conversation AI Team. "Toxic Comment Classification Challenge." *Kaggle*, 2018, www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview.

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

Appendix

Table 1: Sentiment vocabularies

Toxic count=122	**Severe toxic** count=175	**Obscene** count=139	**Insult** count=132	**Threat** count=182	**Identity hate** count=161
youll	youll	youll	youll	youll	heard
suck	heard	suck	self	heard	islam
prick	self	prick	suck	islam	cultur
dude	fcking	dude	prick	undo	suck
dumb	touch	dumb	dude	self	hitler
final	suck	final	bet	suck	prick
nonsens	dad	wont	dumb	hitler	dude
wont	prick	wrote	wont	prick	bet
wrote	dude	ugli	wrote	dumb	dumb
loser	bet	loser	ugli	final	wont
bother	dumb	hole	loser	wont	small
bunch	wont	bother	hole	wrote	ugli
eat	wrote	bunch	bother	small	speak
die	small	eat	brain	ugli	jewish
fat	ugli	unblock	bunch	jewish	loser
power	loser	die	eat	loser	monkey
serious	brain	fat	die	round	bunch
ass	hole	dog	fat	punch	eat
friend	bunch	power	mouth	hole	babi
kill	eat	serious	dog	pit	die
boy	unblock	ass	power	brain	fat
watch	die	friend	douch	wit	dog
liar	fat	kill	serious	threaten	power
gonna	mouth	nerd	ass	eat	defend
damn	dog	boy	friend	fat	serious
crap	power	watch	kill	die	ass
god	serious	lick	nerd	swear	friend
bitch	ass	dirti	boy	mouth	kill
comput	friend	gonna	watch	dog	nerd
piec	kill	damn	lick	maggot	chink
black	nerd	crap	liar	slowli	boy
girl	rot	god	dirti	hunt	watch
hell	boy	bitch	gonna	serious	lick
hard	watch	comput	damn	jack	destroy
cock	lick	piec	crap	ass	gonna
stick	dirti	black	god	soon	antisemit
face	gonna	girl	bitch	friend	dirti

wasnt	damn	hell	comput	kill	damn
arent	crap	hard	piec	gross	crap
stuff	god	scum	black	rot	woman
enjoy	bitch	cock	girl	boy	god
guess	cum	stick	hell	watch	bitch
pathet	comput	face	hard	gonna	piec
hey	piec	arent	scum	destroy	black
busi	ars	turn	cock	damn	girl
yeah	laid	wasnt	stick	crap	hell
hate	girl	stuff	face	god	hard
human	hell	enjoy	arent	bitch	scum
bastard	shove	guess	turn	piec	chines
prove	hard	pathet	wasnt	splatter	cock
listen	lover	hey	stuff	hell	face
fuckin	king	busi	enjoy	shove	turn
asshol	cock	yeah	guess	shot	race
lie	stick	hate	pathet	hard	stuff
jew	face	bastard	hey	heart	guess
faggot	arent	listen	busi	scum	evil
kid	fuke	fuckin	yeah	ground	pathet
truth	stuff	pig	hate	cock	hey
wast	fall	asshol	human	face	yeah
abus	guess	hous	bastard	send	hate
piss	pathet	lie	listen	neck	human
white	hey	jew	fuckin	turn	today
told	busi	faggot	pig	throat	bastard
game	yeah	kid	asshol	cancer	listen
death	hate	truth	lie	pathet	fuckin
harass	human	wast	jew	hey	muslim
sad	ruin	abus	faggot	yeah	pig
love	fli	piss	kid	intent	beat
hand	bastard	white	truth	hate	asshol
ahead	listen	told	wast	human	lie
sick	muslim	game	abus	hang	jew
nazi	fuckin	death	piss	bastard	america
homosexu	pig	dickhead	white	listen	faggot
dick	asshol	harass	told	fuckin	kid
mother	hous	sad	game	pig	truth
away	lie	love	death	beat	wast
youv	jew	hand	dickhead	asshol	piss
school	faggot	ahead	wouldnt	blood	white
stupid	kid	mom	harass	hous	wait
pussi	truth	sick	sad	jew	death

obvious	hit	nazi	love	faggot	children
troll	fucken	homosexu	hand	kid	arab
act	shoot	dick	ahead	slit	sad
job	wast	mother	mom	forev	european
fucker	abus	away	sick	hit	love
motherfuck	geek	youv	nazi	shoot	mom
shit	piss	school	homosexu	abus	sick
nice	white	dumbass	dick	pain	nazi
sex	death	whore	mother	piss	dick
racist	dickhead	stupid	away	white	homosexu
big	sad	pussi	youv	member	mother
dead	goddamn	obvious	school	ago	away
cunt	worthless	wanna	dumbass	wait	school
moron	love	troll	whore	death	lesbian
internet	punk	act	homo	children	whore
rape	ahead	job	stupid	threat	cut
fun	mom	cocksuck	pussi	bloodi	terrorist
ball	sick	fucker	obvious	wouldnt	homo
peni	blow	bag	troll	harass	anim
bullshit	homosexu	motherfuck	act	sad	stupid
head	nazi	shit	job	aliv	twat
nigger	fail	nice	cocksuck	goddamn	slave
dare	mother	kiss	fucker	knife	german
joke	away	sex	bag	worthless	pussi
deserv	dick	racist	motherfuck	love	pedophil
idiot	youv	big	shit	forc	cuz
jerk	school	dead	nice	track	christian
insult	dumbass	cunt	racist	hand	wanna
deal	whore	moron	big	ahead	religion
shut	cut	internet	dead	corps	huge
fuck	homo	rape	cunt	mom	act
son	wipe	fun	moron	nazi	job
total	fck	ball	internet	blow	filthi
fool	fuk	peni	rape	homosexu	cocksuck
fag	stupid	bullshit	fun	dick	bag
man	butt	head	ball	mother	fucker
sit	pussi	nigger	peni	away	motherfuck
run	troll	joke	bullshit	youv	shit
wow	wanna	dare	head	whore	turk
gay	act	deserv	nigger	cut	sex
retard	vandalis	idiot	joke	bodi	racist
lol	job	nigga	dare	drown	aint
	filthi	burn	deserv	wipe	big

	cocksuck	jerk	idiot	stupid	dead
	bag	insult	nigga	gun	cunt
	fucker	deal	burn	twat	men
	motherfuck	wtf	jerk	pussi	rape
	shit	shut	shut	pray	slut
	nice	fuck	fuck	wanna	came
	kiss	son	son	filthi	ball
	sex	total	total	fucker	peni
	racist	fool	fool	bag	indian
	aint	fag	freak	hurt	head
	wanker	man	fag	motherfuck	nigger
	big	sit	famili	earth	dare
	dead	wow	man	shit	deserv
	cunt	gay	run	nice	idiot
	moron	retard	sit	sex	nigga
	internet	lol	wow	racist	theyr
	rape		gay	aint	cool
	slut		retard	wanker	asian
	fun		lol	stab	burn
	ball			big	sucker
	peni			dead	murder
	bullshit			cunt	shut
	head			internet	fuck
	nigger			rape	poor
	dare			protect	disgust
	deserv			gut	throw
	wife			came	jesu
	idiot			ball	son
	nigga			singl	women
	anal			peni	fool
	hour			shall	fag
	burn			head	famili
	sucker			nigger	man
	wank			dare	queer
	jerk			deserv	russian
	shut			planet	gay
	fuck			idiot	retard
	son			hour	lol
	total			burn	
	laugh			jerk	
	mum			murder	
	tit			shut	
	useless			kick	

	fag freak famili man sit gay retard shitti lol			fuck disgust throw son till laugh fool fag man famili sit horribl gay retard supertrl knock	