Toxic Comment Classification
Identifying and Classifying Toxic Online Comments

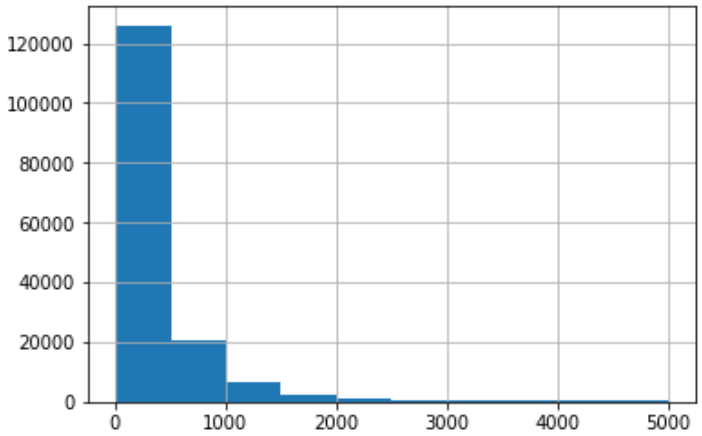
Problem Definition:
Moderating content is hard, this is especially the case for anonymous message boards which can bring the worst out in people with no repercussions for negative behavior. The toxic content that gets past moderation damages the health of the community, and drives away potential members. In fact, according to a study done by the Pew Research Center more than a quarter of Americans have chosen to not post something online after seeing the harassment of others. Additionally, four in ten internet users are victims of online harassment with varying degrees of severity. So with that problem in mind, I jumped t the opportunity to take part in a Kaggle competition that tries to address this pervasive issue. Within the context of the Kaggle challenge, the problem is, given a comment from Wikipedia's talk page edits, can you give a probability that it is toxic, severely toxic, obscene, threatening, insulting, hate speech, or none of the above? The probability outputs are not mutually exclusive, and a comment that is threatening, can also be insulting. Solving this problem may give moderators another tool to combat antisocial behavior and limit the amount of users that are turned off from contributing.

Brief Data Description:
The data I'll be working with has been provided by Kaggle in its Toxic Comment Classification Challenge, which contains comments from Wikipedia's talk page edits, as well as 6 binary target columns of toxic, severely toxic, obscene, threatening, insulting, hate speech.

The dataset contains 159,571 comments some of which are normal, and others which are antisocial. There are no missing data or incorrect data in this dataset, however we may need to do some cleaning of each comment. The average comment length is 394 characters, with a std deviation of 590 characters, minimum of 6 and max of 5000. The actual distribution of the comments is right skewed (positive skewness), with the majority of the comments falling 6 and 500 in length.

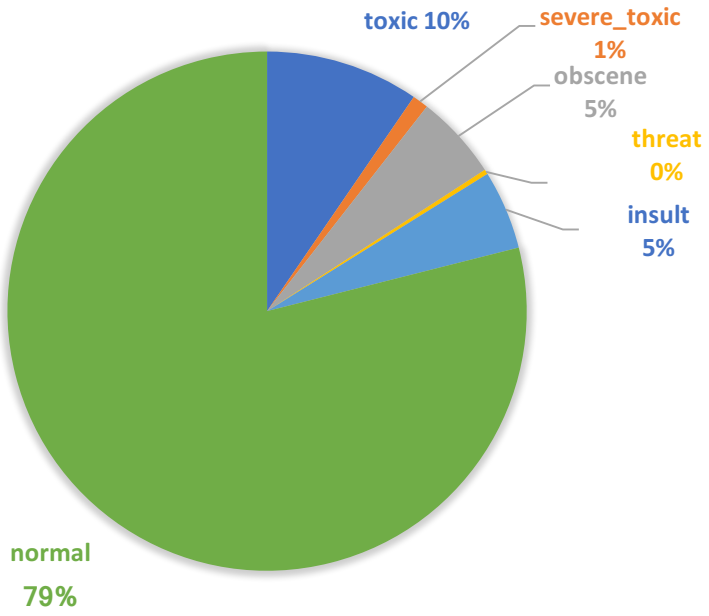
```
lengths = df.comment_text.str.len()
lengths.hist()
lengths.mean(), lengths.std(), lengths.max(), lengths.min()
```

```
(394.0732213246768, 590.7202819048923, 5000, 6)
```



Looking at the class breakdown, we see its pretty unbalanced in its distribution, which is important to keep in mind as we train our models. For example, Threat and Severe_Toxic have very low sample size, especially when compared to other tags and the total dataset.

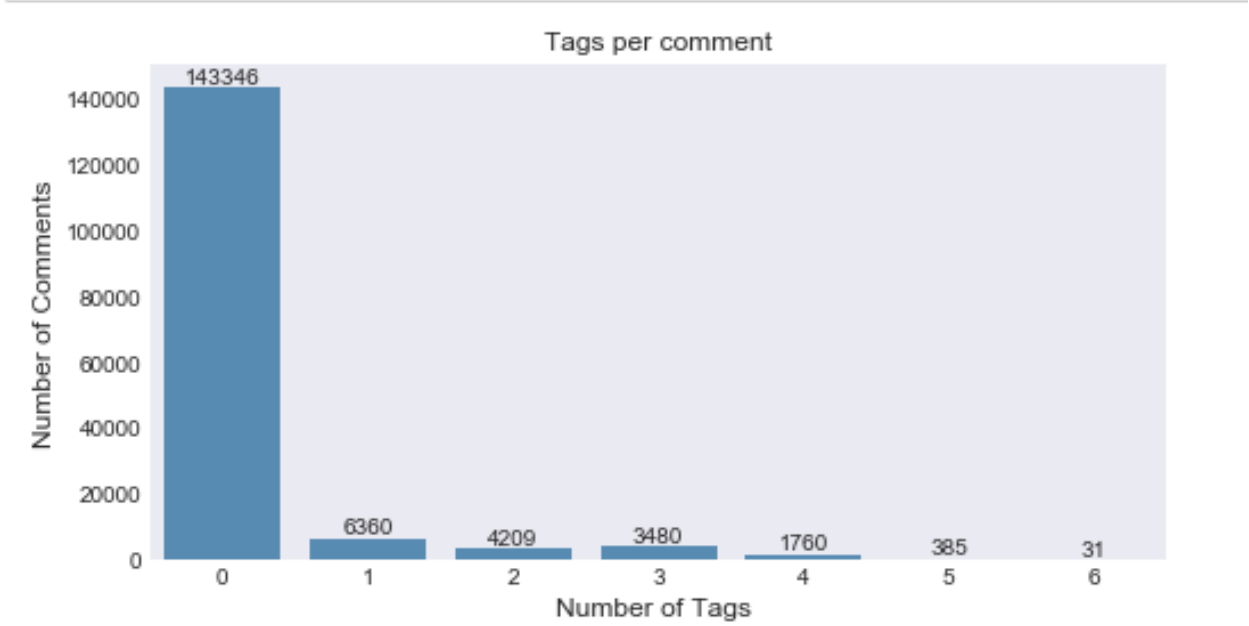
toxic	15257
severe_toxic	1592
obscene	8427
threat	477
insult	7857
normal	125961
Total	159571



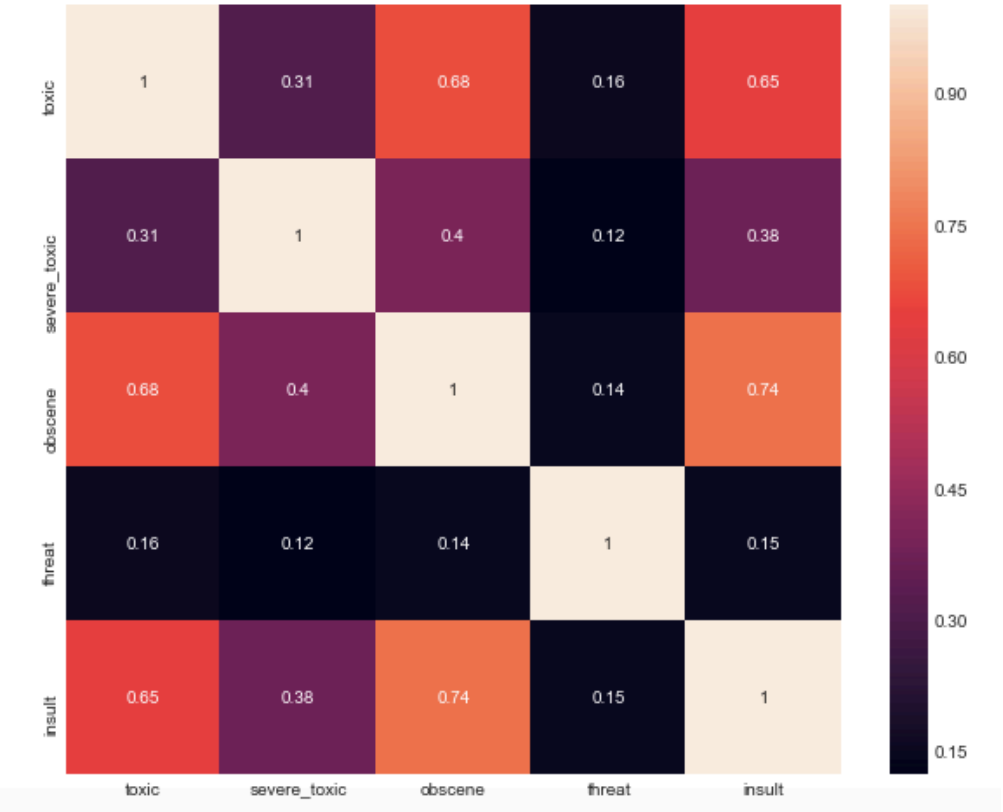
Here are some examples of the different types of comments that can occur. By this small example, we can already see that number of all caps words may be indicative of toxicity.

Normal	Toxic	Severe_Toxic	Obscene	Threat	Insult	Identity_hate
Radial symmetry Several now extinct lineages included in the Echinodermata were bilateral such as Homostelea, or even asymmetrical such as Cothurnocystis (Stylophora). -	Bye! Don't look, come or think of comming back! Tosser.	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	I NEVER FUCKING MADE THIS MOTHER FUCKING ARTICLE I JUST FUCKING EDITED IT AND THAT WAS A FUCKING LONG TIME AGO FUCKING ASSHOLES! PISSING ME OFF @	Hi! I am back again! Last warning! Stop undoing my edits or die!	HELLO You disgrace to humanity. Stop wasting valuable Internet resources (your killing the fucking planet) and throw out your PC. PS it's like 3am what the fark kinda life do you have on Wikipedia 24/7. A VERY SAD AND LONELY ONE LOL. Life's too short find something better to do	Hey Roy. Go fukk yourself you gay bich. Ya u heard me. Pussy

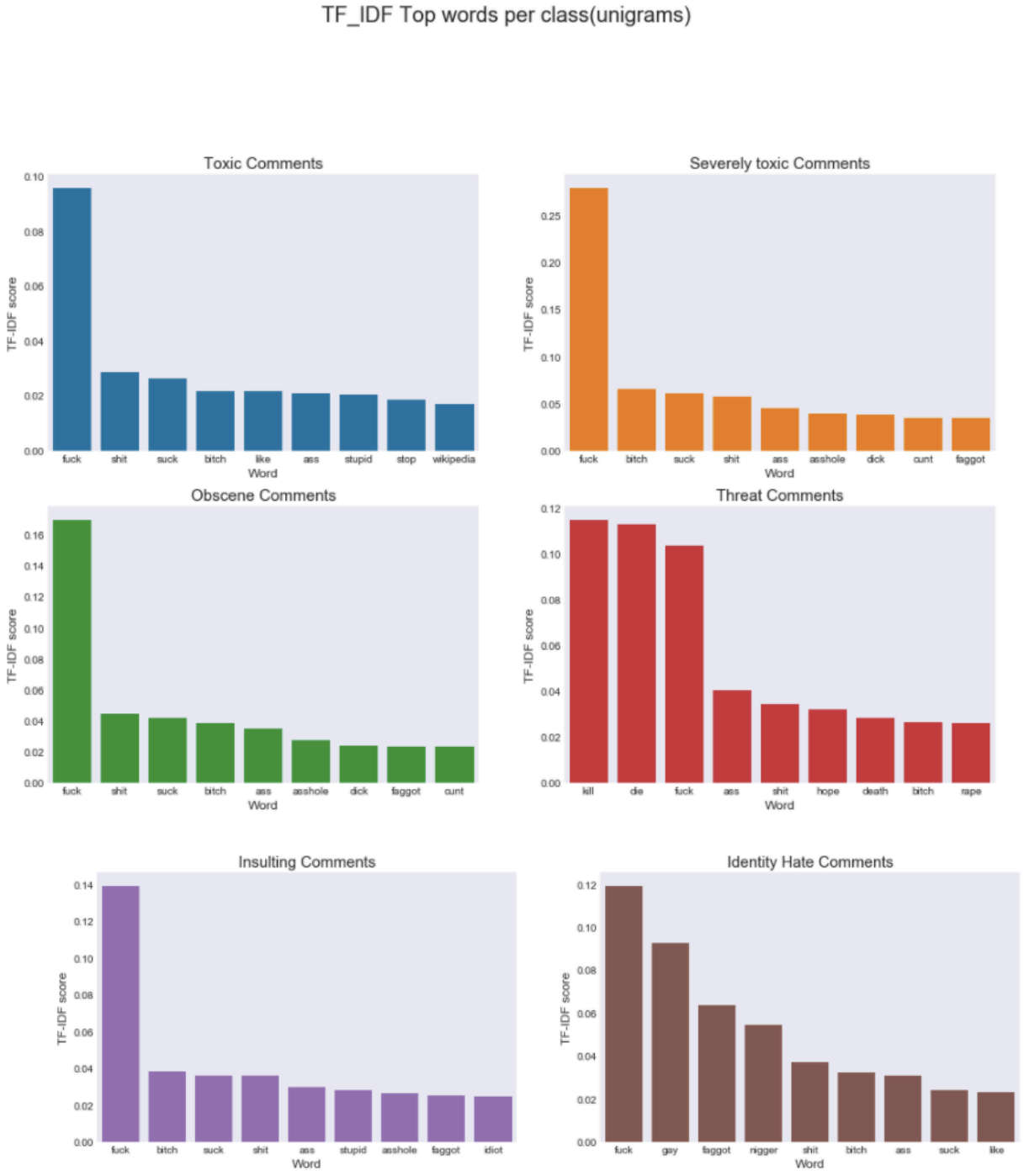
In this graph we can see that while the majority of comments have zero tags, some comments appear to have multiple tags per comment.



Looking a little closer, we can see from this correlation matrix that there are certain classes that are closely correlated with one another. For example, toxic, insulting and obscene comments seem to be fairly correlated.



Taking a look the top words by class (based on TF_IDF), its evident that a lot of words that appear in one class, are likely to appear in another class, which will be an interesting quirk in the data to exploit, and again further drives home the point that comments can have multiple different tags.



The word cloud visualization demonstrates the uniqueness of this dataset, for example words like page, fact, reference, reliable source, section give the impression that this dataset is most likely coming from Wikipedia, which would be useful to know if we hadn't already known prior to the competition where the data was coming from.

Describe the Challenges:
The challenges that will need to be dealt with is poor spelling and intentional misspelling to avoid filters. Locating and identifying a list of bad words will also be useful in creating a new feature (i.e. does this comment have a bad word in it? How many?) thankfully there are many resources online that can be referenced.

Being able to detect sarcasm will also be important, however as of right now there doesn't seem to be a very good way of doing this, as a result it may simply be a challenge that goes unaddressed.

There is also the challenge of an obvious class imbalance, with only 477 threatening comments out of a total of 159,571 it will be difficult to be able to build an accurate classifier without either creating synthetic comments or finding other external datasets containing threatening comments.

Finally an issue that can't really be addressed is the fact that the tagging of the 6 categories was done via crowdsourcing, so it may not be 100% accurate, since something that one crowdsourcing user finds toxic, might be completely benign to another.

Methodology:

Data Gathering

Data gathering included scouring the web for additional datasets beyond the one provided by the Kaggle competition. I was also able to find a list of banned words, along with a naughty word list compiled by Google which may prove useful in feature engineering (i.e. does this comment include a banned word?). Another dataset I made use of were English stop words, along with a dictionary of apostrophe words mapped to their non apostrophe counterparts (i.e. Didn't -> Did Not).

Data Exploration

Exploring the data involved using Python and Excel primarily. I started off by figuring out what the class composition looked like with Excel, which was a simple SUM() function that added all the 1's in the target columns. I was then able to use this data to visualize a pie chart showing the breakdown of normal v.s. toxic data. I then went on to identify the mean, std, max, min and distribution of comments to get a better understanding of how these comments were structured which was done in Python.

Since each comment's classification isn't mutually exclusive, I also thought it may be interesting to see what a heatmap would look like of the classifications, and see if there were any overlaps/correlations. Looking at this heatmap toxicity was highly correlated with insulting (0.65) and obscene (0.68).

I also explored given a certain toxic classification, what was the top word based on TF-IDF which helped me identify words that were common among different classifications, as well as certain keywords that were specific to only one classification

Another visualization technique I explored was word clouds, which Python has an external library for. I was able to explore what the common words were in the corpus of text with a simple visualization of a word cloud. Finally I identified 7 different comments that had different classifications (including normal) to get a better understanding of what each of these comments looked like.

Data Preparation

In terms of data preparation, I started by doing some feature engineering. Specifically, I created 5 new columns, which counted the number of uppercase and unique words, as well as the number of stop words, punctuation, and bad words from my external bad words dataset. I did this with a simple lambda function which iterated over each comment and counted how many times what I was searching for occurred.

After this was done, the next step was to clean the comments, which I started by applying a mapping function to each comment. First, I made sure to convert each comment to lowercase, and proceeded to remove new character lines. I was also made aware of potential leaky features in the form of ip and usernames, so I made sure to remove those as well. I then proceeded to tokenize the comment and search through an apostrophe dictionary that I found online which converted words like wasn't into was not. Additionally, I made sure to remove any words that were deemed stop words, based off of the nltk.corpus library of stop words.

After my comments were cleaned, I then had to transform them into vectors with the help of TfidVectorizer so that my machine learning algorithms would be able to take them as input. Since I would be combining all these features, I turned the word_features sparse matrix into a pandas dataframe so that I could then proceed to combine all these features into one big pandas datafarame. This set of features along with my target column, would be passed into a train_test_split function, allowing me to extract my test and train features and target values which will be used in the model building step.

Model Building

Before attempting to build any models, I decided on a set of evaluation metrics which would help distinguish poor models with better performing ones. My initial thought was to use accuracy, but I realized this would be a poor evaluation metric given that a model that misclassified everything as not toxic might perform well given skewed distribution of target classes. Thus, I settled on Recall, Specificity, Precision and most importantly the Matthews Correlation Coefficient, which provides a value between -1 and +1, with a +1 representing a perfect prediction, 0 an average random prediction, and -1 an inverse prediction, as my four evaluation metrics.

For model building, I decided to first implement a baseline model without any data preparation to see what my accuracy would look like. For my baseline model, I used a Multinomial Naïve Bayes classifier specifically for each target column, classifying them as a target value or not.

With this baseline, I could now judge the effectiveness of my data preparation step. Since I had already done all the hard work of cleaning and feature engineering in the previous section, it was a simple matter of doing a train_test_split on this prepared dataset and passing it into the MNB classifier. Aside from specificity, which relates to the classifier's ability to identify negative results (i.e. is it misclassifying positives as negatives?), it performed better across the board for all the evaluation metrics I had outlined previously.

Confident that my data preparation and feature engineering steps had improved my results, I proceeded to test a Logistic Regression model, which didn't seem to perform better on any evaluation metric with either the baseline or the MNB with data preparation.

The third and final model that I tested was a SGDClassifier which performed better than the baseline, however failed to surpass either the logistic regression or Multinomial Naïve Bayes models in any significant way.

Given these results, I proceeded to rank and evaluate each model based on each evaluation metric. Ultimately the model that performed the best was the Multinomial Naïve Bayes model with data preparation, which had a MCC score of 0.474 (0.07 higher than Logistic Regression), scored the highest on Precision, and was close behind all the other models in its Recall and Specificity.

Evaluation:

Recall	**Toxic**	**Severe_toxic**	**Obscene**	**Threat**	**Insult**	**Identity_hate**	**Average**
Multinomial Naive Bayes (Base)	91.00%	99.00%	95.00%	100.00%	95.00%	99.00%	96.50%
Multinomial Naive Bayes (With Data Preparation)	94.00%	98.00%	96.00%	100.00%	96.00%	99.00%	97.17%
Logistic Regression (With Data Preparation)	95.00%	99.00%	97.00%	100.00%	97.00%	99.00%	97.83%
SGDClassifier (With Data Preparation)	93.00%	99.00%	97.00%	100.00%	96.00%	99.00%	97.33%

Specificity	**Toxic**	**Severe_toxic**	**Obscene**	**Threat**	**Insult**	**Identity_hate**	**Average**
Multinomial Naive Bayes (Base)	99.96%	100.00%	99.98%	100.00%	99.98%	100.00%	99.99%
Multinomial Naive Bayes (With Data Preparation)	94.08%	98.85%	97.37%	99.88%	96.97%	99.12%	97.71%
Logistic Regression (With Data Preparation)	99.84%	99.91%	99.87%	100.00%	99.58%	100.00%	99.87%
SGDClassifier (With Data Preparation)	99.97%	100.00%	99.92%	100.00%	99.68%	100.00%	99.93%

Precision	**Toxic**	**Severe_toxic**	**Obscene**	**Threat**	**Insult**	**Identity_hate**	**Average**
Multinomial Naive Bayes (Base)	92.00%	98.00%	95.00%	100.00%	95.00%	98.00%	96.33%
Multinomial Naive Bayes (With Data Preparation)	95.00%	99.00%	97.00%	99.00%	97.00%	99.00%	97.67%

Logistic Regression (With Data Preparation)	95.00%	99.00%	97.00%	99.00%	96.00%	99.00%	97.50%
SGDClassifier (With Data Preparation)	94.00%	98.00%	97.00%	99.00%	96.00%	98.00%	97.00%

Matthews Correlation Coefficient	**Toxic**	**Severe_toxic**	**Obscene**	**Threat**	**Insult**	**Identity_hate**	**Average**
Multinomial Naive Bayes (Base)	0.328	-0.006	0.236	0.046	0.143	-0.0006	0.1244
Multinomial Naive Bayes (With Data Preparation)	0.693	0.466	0.7	0.04	0.64	0.31	0.4748333333
Logistic Regression (With Data Preparation)	0.66	0.302	0.692	0	0.56	0.192	0.401
SGDClassifier (With Data Preparation)	0.52	0	0.622	0	0.4648	0	0.2678

From the results its clear that the Multinomial Naïve Bayes with data preparation is the best model among the ones that I tested. This is based off of its higher Matthews Correlation Coefficient, and the relatively similar performance among other dimensions such as Precision, Specificity, and Recall.

Conclusions:

Working with the dataset provided by Kaggle it was clear that there still needs to be a lot of work done in the domain of toxicity detection. From my own personal experience of being part of online communities, toxic behavior can be a huge deterrent to developing strong cohesive communities and machine learning promises to be an interesting tool to moderate toxicity and address these challenges.

Looking at the previous section on Evaluation, its evident that the data preparation and feature engineering section yielded improvements in our metrics across the board (aside from Specificity which curiously the base model had performed the best in). This really drives home the point that often times with Kaggle challenges (As in the real word), the model selection is only a minor step in the process of fine tuning performance, and in fact it’s the data cleaning, feature

engineering and external datasets that will more often than not yield the highest increase in performance.

It was also clear from the challenge that class distribution is an important dimension to think about when building models. In the case of this challenge, there were very few examples of threatening and severely toxic comments, which made the task of classifying future instances of these comments a lot more complicated.

I also extended this challenge further by thinking about how it could be deployed in a business setting. The first step would be to select the best performing model (Multinomial Naïve Bayes with Data preparation in this case), pickle it, and turn it into a REST API with the help of Flask and Heroku. Once this is done, you can now perform Post requests to this API that will return a probability that a comment is toxic. If its beyond a certain threshold, then it will be screened for additional moderation (or perhaps outright rejected), which could take the form of either volunteer moderators or Mechanical Turk workers. False negatives that sneak through the filter could be detected through a user reporting mechanism that flags comments for additional screening after they've crossed a certain threshold for user reports. Meanwhile the model can be continuously learning through online learning and improving its prediction capabilities. Deploying such a service would ease the burden from moderators so they can focus on non-automatable tasks instead.

Future Work:

If others were to pickup from where I left off, there'd be some interesting work in exploring how to detect sarcasm (something I'm sure many PhD's and postdocs are currently working on).

I'd also say it'd be interesting to explore other models to see if they would yield any increase in performance. Additionally, stacking could be explored as a way to combine several weak learners into a stronger learner.

If capital was available, perhaps outsourcing to Mechanical Turk to classify additional Wikipedia comments would be helpful, maybe even going so far as generating additional comments for tags that have few samples available (i.e. Threat comments).