Natural Language Processing - Sentiment Analysis

*CAUTION!!! You may find the content *offensive*!*

Problem Statement

The possibility of being verbally abused and harassed by strangers online is not a great motivator for people to engage and learn from one another's perspectives.

As a result, social networks and media platforms are challenged with effectively facilitating open conversations. This results in many such online communities preventing users from expressing their opinions, giving feedback etc. on their platform.

This appears to be a big problem and is being actively addressed and researched by companies like Alphabet.

- Conversation AI
    - Collaborative research effort exploring Machine Learning as a tool for better discussions online
- Jigsaw
    - Incubator within Alphabet building technology to tackle global security challenges
- Perspective API
    - Publicly available models (including toxicity) served through the Perspective API

Goals

- Perform exploratory data analysis (EDA) to understand texts being classified as ***"toxic"***, ***"severe-toxic"***, ***"threatening"***, ***"insulting"***, ***"hateful" and/or "obscene"*** in nature
- Create a set of Machine Learning NLP models to classify text as one or more of the above classes

Dataset

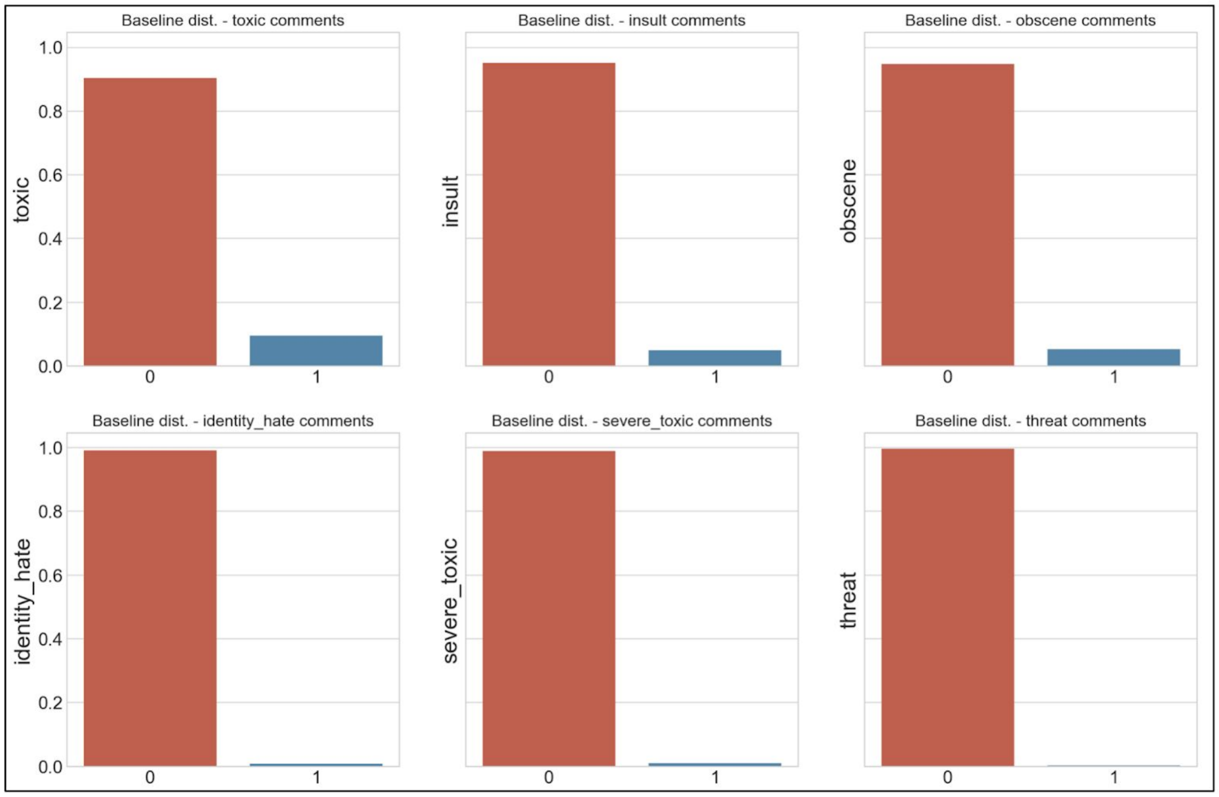
- Source: https://conversationai.github.io/
    - It is important to note that dataset has been hand-labeled and, as far as I know, there doesn't seem to be any reference material that tells us why a particular comment was labeled ***"toxic"***, ***"severe-toxic"***, ***"threatening"***, ***"insulting"***, ***"hateful" and/or "obscene"*** in nature
- Size:
    - Data: 159,571 rows
- Nulls:
    - The dataset ***does not*** have any null values/rows

Exploratory Data Analysis (EDA)

This section outlines my methodology to 1) gain high level understanding of the data, and 2) gain insights that are not obvious by looking at raw data.

Class distribution

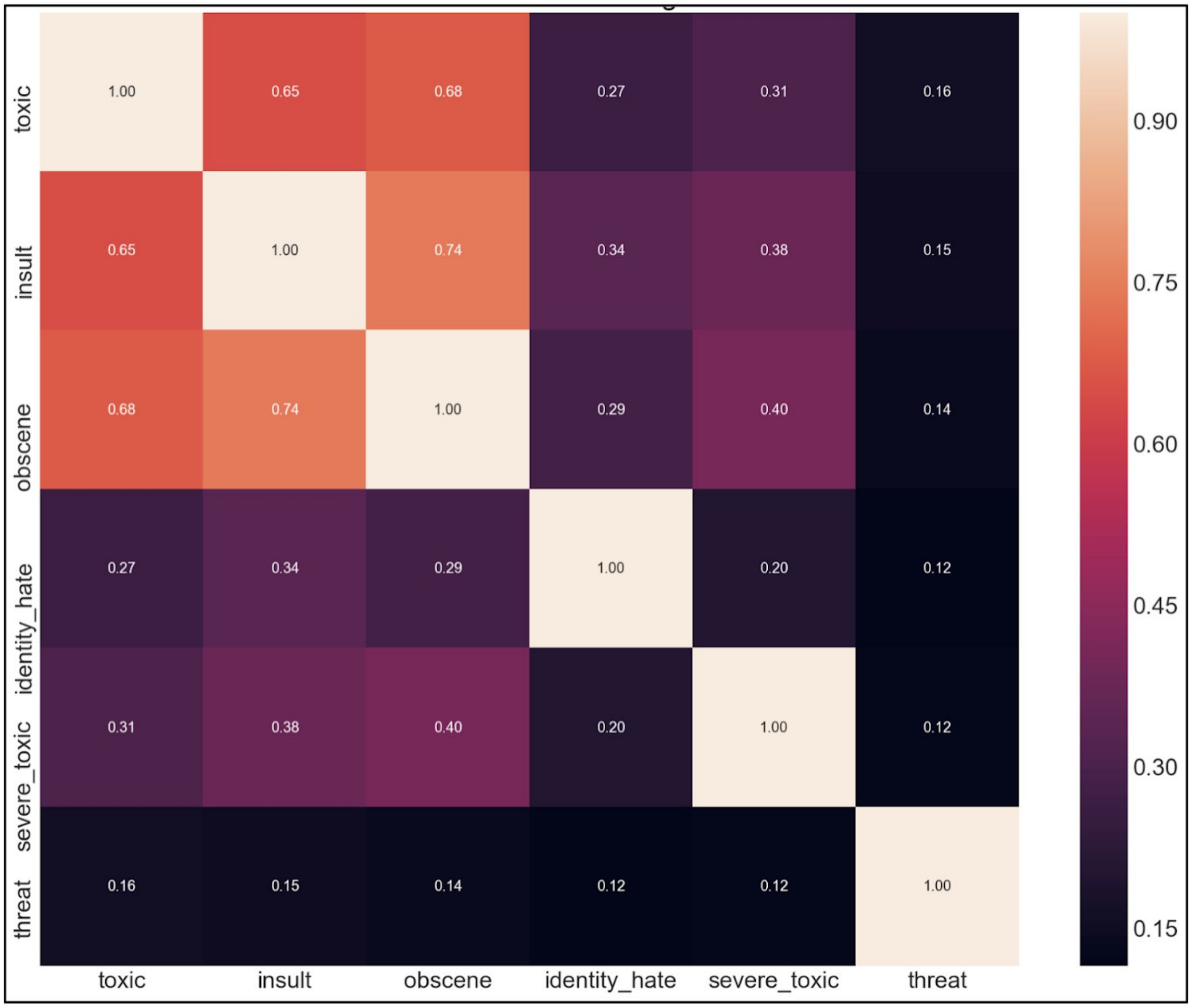
These set of plots show us how the six classes are distributed. For example, 90% of comments are toxic and 10% of comments are not toxic. This means that the classes are very imbalanced and the baseline accuracy is very high across the board.



Therefore, techniques such as ***oversampling-undersampling***, ***bias correction***, ***stratified cross-validation***, ***changing classification threshold***, *and* ***purposefully optimizing for evaluation matrix*** will be beneficial during model creation and evaluation process to ensure models' performance is above baseline accuracy.

Class correlations

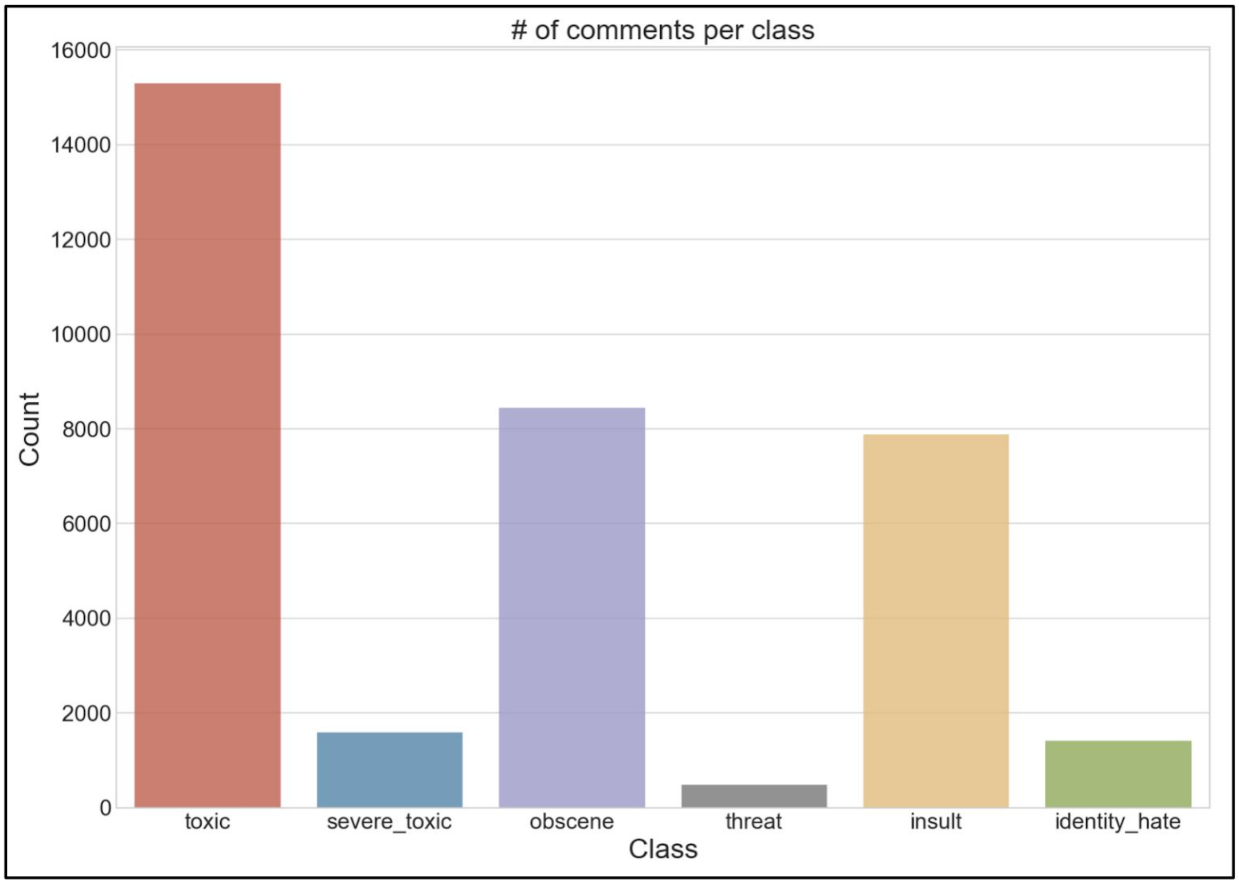
The reason behind creating this correlation matrix is to see if a given class can be leveraged as a feature when creating models for one of the other classes.



Interestingly there seems be high correlation between ***obscene*** & ***toxic*** and ***obscene*** & ***insulting*** but not between ***toxic*** and ***server_toxic***.

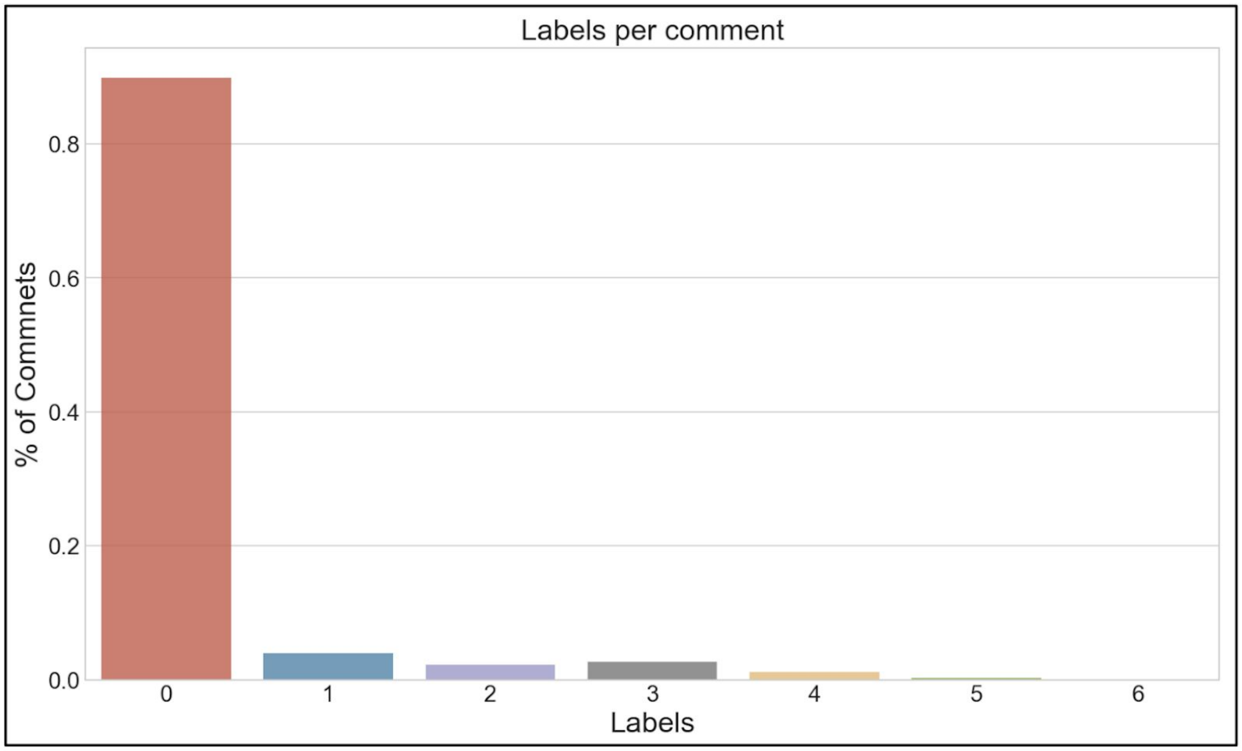
Comments per class

It is clear that majority of the labeled classes are “toxic”



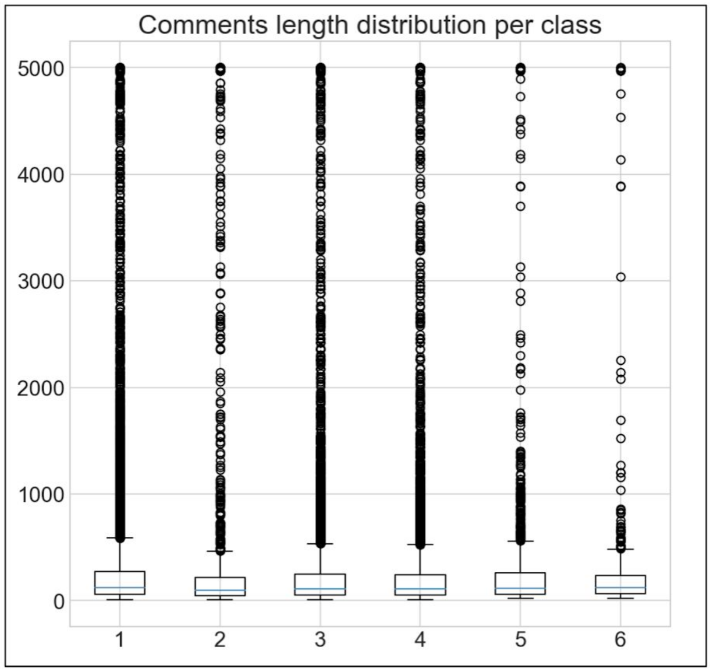
Labels per comment

This is insightful not because it tells us that majority of the comments are not labeled (0) but, more importantly, there are comments that have been assigned more than one label. This indicates that six (6) models will need to be created.



Comments length distribution

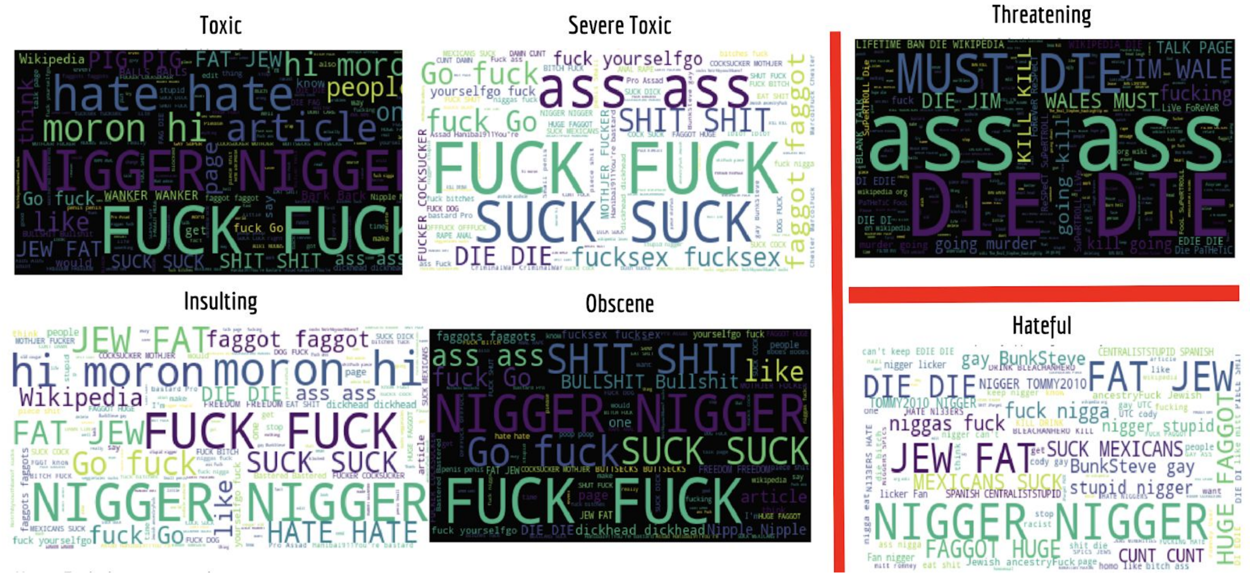
Note the difference in the avg length of comments between ***toxic*** and ***severe_toxic*** classes.



	Class	Avg Len
1	toxic	295
2	sever_toxic	453
3	obscene	286
4	insult	277
5	identity_hate	308
6	threat	307

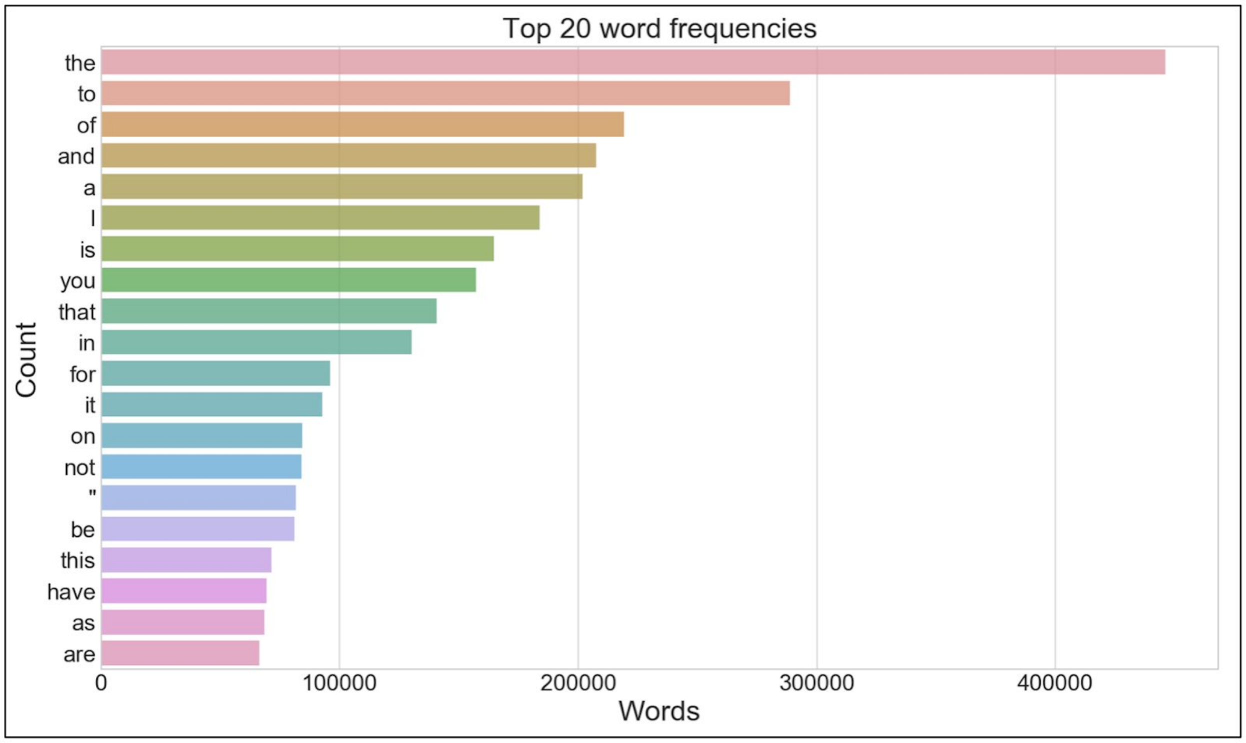
Frequently appearing words per class

This chart displays words that frequently appear in each of the classes. (Note: excluding stop words.) As one would imagine, most of the (curse) words are common among ***toxic*, *severe toxic*, *insulting and obscene*** comments and there's a clear distinction between words more common in threatening and hateful comments.



Top 20 word frequencies across all classes

This chart makes it clear that removing stop words during feature engineering will be necessary so most frequently appearing words that provide little to no information are removed from the features list during model creation.



Frequently appearing words across all classes

This word cloud shows the most frequently appearing words (*excluding stop words*) across all comments.

one
article
also
think
talk
page
even
people
fact
know
well
say
work
section
editor
way
time
mean
thing
seem
see
since

Picture worth 1000 words… really!

This picture is really worth a 1000 words because it tells us that using **TFidfVectorizer** instead of **CountVectorizer** to create features during model creation will be a better choice.

Here’s why: **CountVectorizer** only counts frequencies and assigns equal weights to all including these top words/features that basically have no “meaning” with respect to our target classes. With **TFidfVectorizer** the score increases proportionally to count, but it is also offset by the frequency of the word in the entire corpus -- as a result, more “meaningful” / “rare” words are assigned higher weights.

Feature Engineering

These are the numeric features created and applied dynamically using FeatureUnion technique during model fitting.

- Sentences count
- Word count
- Unique word count
- Stop words count

Feature Transformation

I followed the below process to transform and pre-process all comment texts:

- Noise Removal
    - Html, Other markup
    - Line breaks
- Normalization
    - To lowercase
    - Replace contractions (y'all -> you all)
    - Remove punctuation
    - Remove non-ASCII
    - Remove stop words (nltk.corpus)
    - Lemmatize verbs
- Vectorization using **TFidfVectorizer**

Machine Learning and Modeling

Baseline Models using DummyClassifier

Before creating complex models, I decided to quickly create simple, baseline models using sci-kit's learns **DummyClassifier** to compare with other classifiers. I set DummyClassifier's ***strategy*** hyperparameter to "stratified" so it generated predictions by respecting the training set's class distribution. Here are the results:

Class	**Accuracy**	**ROC AUC**	**False Negatives**
toxic	0.82	0.50	3,470
severe_toxic	0.98	0.50	392
obscene	0.90	0.49	2005
insult	0.90	0.49	1,879
identity_hate	0.98	0.49	347
threat	0.99	0.49	118

Note that as expected the accuracies are high but ROC AUC is right at .50% and the number of False Negatives are relatively high across the board as well.

Model Evaluation and Class Imbalance

Options available to address the class imbalance issue:

- Bias correction
- Oversampling / undersampling
- Weighting observations
- Stratified cross-validation
- Changing classification threshold
- Purposefully optimizing evaluation metrics

I chose to optimize for evaluation metrics to achieve balance between ***max sensitivity*, *min false positives and accuracy***

- Sensitivity
    - When actual value is positive, how often is the prediction correct
- False Negatives
    - A truly “toxic” comment is predicted to be “non-toxic”

Model Selection -- MultinomialNB

- Suitable for classification with discrete features -- including fractional counts such as ***tf-idf***
- Hyperparameter ***class_prior***
    - Specify prior probabilities of the classes -- to address class imbalance
        - Tune for max **Sensitivity** and min **False Negatives**

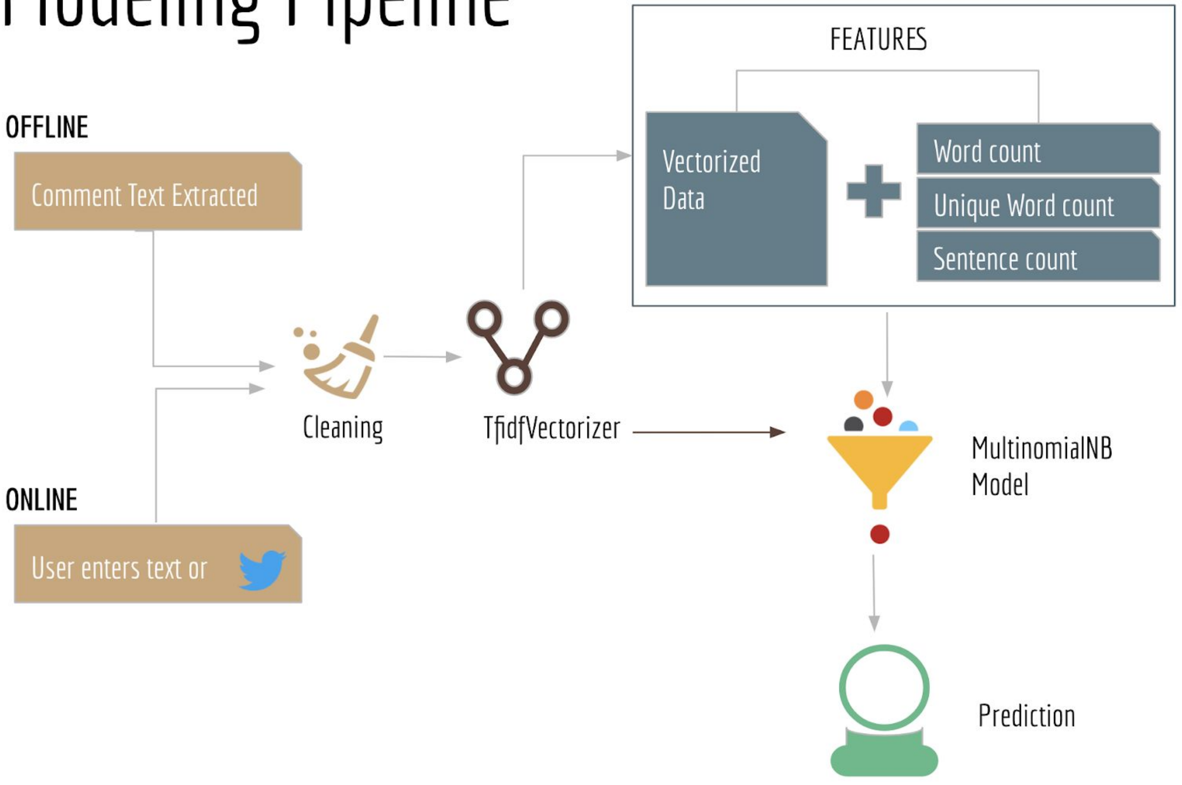
Modeling Pipeline

I used FeatureUnion and Pipeline objects to setup the modeling pipeline as follows to test features and tune ***class_prior*** as well as other hyperparameters:

- Feature extractors and transformers
    - Extract and return comment text from the dataframe
        - Vectorize using TfidfVectorizer

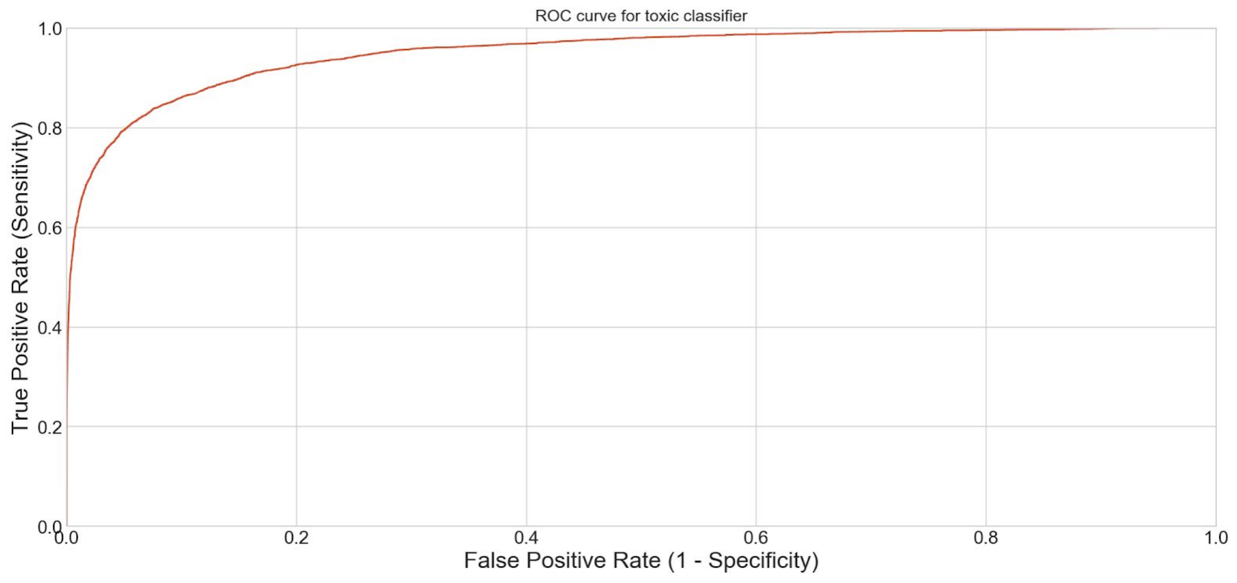
    - Calculate word count for each original/raw comment text and return it as a feature df
    - Calculate unique word count for each original/raw comment text and return it as a feature df
    - Calculate word count for each cleaned comment text and return it as a feature df
    - Calculate unique word count for each cleaned comment text and return it as a feature df
- GridSearch with 10-fold cross-validation
- Recorded the following metrics
    - Best GridSearchCV score
    - Holdout GridSearchCV Score
    - Null accuracy
    - Accuracy
    - ROC AUC Score
    - Confusion matrix (TN, FP, FN, TP)
    - Specificity
    - Precision
    - **Sensitivity**
    - **False Negatives**

Modeling Pipeline



Result Metrics (For “toxic” class)

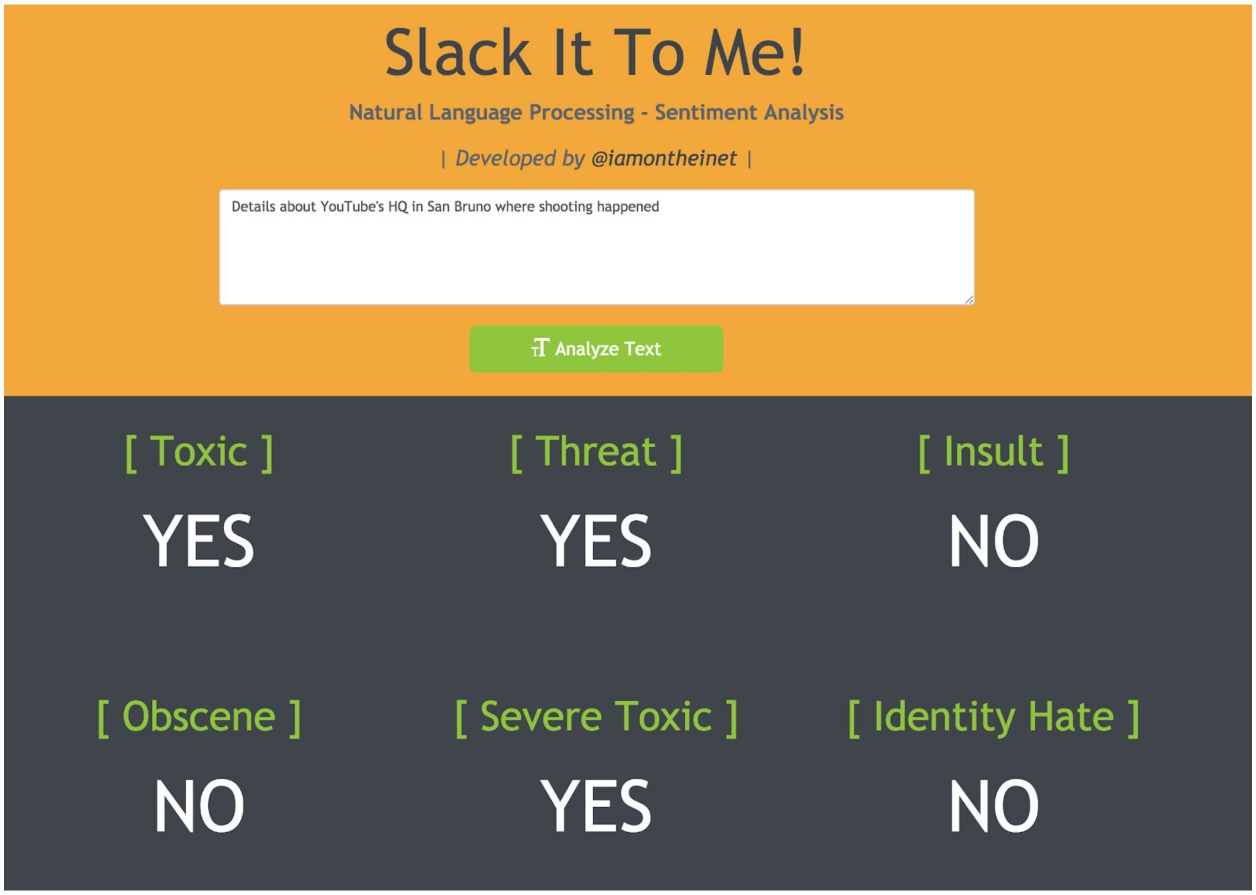
Class Priors (class 0, class 1)	**Sensitivity**	**False Negatives**	**Accuracy**
Default	0.54	1,731	0.95
[0.5, 0.5]	0.86	511	0.88
[0.4, 0.6]	0.91	327	0.82
[0.3, 0.7]	0.94	214	0.74
[0.1, 0.9]	0.98	55	0.48



Note: Change in ***class_prior*** impacts **Sensitivity, False Negatives** and **Accuracy.** Since in this case the goal was to find a (reasonable) balance between the three, I was able to achieve that goal by tuning ***class_prior*** at values **[0.4, 0.6]. (**A similar pipeline and hyperparameter tuning was applied to other classes.)

Web Application

Created web application (Slack It To Me!) where user can input free-form text to see if it classifies as "toxic", "severe toxic", "obscene", "threatening", "insulting", "hateful" in nature.



Slack It To Me!

Natural Language Processing - Sentiment Analysis

| *Developed by* @iamontheinet |

Details about YouTube's HQ in San Bruno where shooting happened

[Toxic]	[Threat]	[Insult]
YES	YES	NO
[Obscene]	**[Severe Toxic]**	**[Identity Hate]**
NO	YES	NO

Behind the scenes the web application loads persisted vectorizer and models (trained according to the above modeling pipeline and hyperparameter tuning). The tech stack includes Python, Flask, Bootstrap, jQuery and the application is hosted and deployed on Heroku.

Next Steps

- Assess other techniques to address class imbalance
- Implement other models, neural networks, word embeddings and tokenizer such as Word2Vec, GloVe, etc.