Toxic comment classification by using ML Algorithms:

The dataset is taken from Kaggle. The dataset has six labels that represent subcategories of toxicity, but the project is going to focus on a seventh label that represents the general toxicity of the comments

# Problem statement:

The goal is to create a classifier model that can predict if input text is inappropriate (toxic).

1. Explore the dataset to get a better picture of how the labels are distributed, how they correlate with each other, and what defines toxic or clean comments.

2. Create a baseline score with a simple logistic regression classifier.

3. Explore the effectiveness of multiple machine learning approaches and select the best for this problem.

4. Select the best model and tune the parameters to maximize performance.

5. Build the final model with the best performing algorithm and parameters and test it on a holdout subset of the data.

# Metrics:

Unfortunately for the problem, but fortunately for the Wikipedia community, toxic comments are rare. Just over 10% of this dataset is labeled as toxic, but some of the subcategories are extremely rare making up less than 1% of the data.

Because of this imbalance, accuracy is a practically useless metric for evaluating classifiers for this problem.

The Kaggle challenge based on this dataset uses ROC/AUC, or the area under a receiver operating characteristic curve, to evaluate submissions. This is a very generous metric for the challenge, as axes for the curve represent recall (a.k.a. sensitivity), the ratio of positive predictions to all samples with that label, and specificity, the ratio of negative predictions to all negative samples. This metric would work well if the positive and negative labels were relatively even, but in our case, where one label represents less than a third of a percent of the data, it’s too easy to get a high score even with hardly any true-positive predictions.

Instead, I propose using an F1 Score, which severely penalizes models that just predict everything as either positive or negative with an imbalanced dataset.

The F1 score is a harmonic average between precision and recall. This combines the strengths of precision and recall while balancing out their weaknesses, creating a score that can fairly evaluate models regardless of dataset imbalance. My justification for focusing on any\_label as the target is that distinctions between the specific labels are relatively ambiguous, and that there is greater value focusing on general toxicity of a comment to more reliably flag it for review. This will reduce the workload of moderators who will ultimately be making the final call, and the specific category more relates to the consequences for the commenter rather than whether or not the comment should be deleted.

# Analysis : Data exploration:

This dataset contains 159,571 comments from Wikipedia. The data consists of one input feature, the string data for the comments, and six labels for different categories of toxic comments: toxic, severe\_toxic, obscene, threat, insult, and identity\_hate.

Here it is showing how the labels are distributed throughout the dataset, including overlapping data. As you can see in the breakdown, while most comments with other labels are also toxic, not all of them are. Only “severe\_toxic” is clearly a subcategory of “toxic.” And it’s not close enough to be a labeling error. This suggests that “toxic” is not a catch-all label, but rather a subcategory in itself with a large amount of overlap. Because of this, I’m going to create a seventh label called “any\_label” to represent overall toxicity of a comment. From here on in, I’m going to refer to any labeled comments as toxic, and the specific “toxic” label (along with other labels) in quotation marks.

Only 39% of the toxic comments have only one label, and the majority have some sort of overlap. I believe that because of this, it will be much more difficult to train a classifier on specific labels than whether or not they are toxic.

16225 out of 159571 comments, or 10.17%, are classified as some category of toxic.

# Breakdown of Category Overlap:

1595 severe\_toxic comments. (1.00% of all data.)

• 1595 or 100.00% were also toxic.

• 1517 or 95.11% were also obscene.

• 112 or 7.02% were also threat.

• 1371 or 85.96% were also insult.

• 313 or 19.62% were also identity\_hate.

1405 identity\_hate comments. (0.88% of all data.)

• 1302 or 92.67% were also toxic.

• 313 or 22.28% were also severe\_toxic.

• 1032 or 73.45% were also obscene.

• 98 or 6.98% were also threat.

• 1160 or 82.56% were also insult.

15294 toxic comments. (9.58% of all data.)

• 1595 or 10.43% were also severe\_toxic.

• 7926 or 51.82% were also obscene

. • 449 or 2.94% were also threat.

• 7344 or 48.02% were also insult.

• 1302 or 8.51% were also identity\_hate.

7877 insult comments. (4.94% of all data.)

• 7344 or 93.23% were also toxic.

• 1371 or 17.41% were also severe\_toxic.

• 6155 or 78.14% were also obscene.

• 307 or 3.90% were also threat.

• 1160 or 14.73% were also identity\_hate.

478 threat comments. (0.30% of all data.)

• 449 or 93.93% were also toxic.

• 112 or 23.43% were also severe\_toxic.

• 301 or 62.97% were also obscene.

• 307 or 64.23% were also insult.

• 98 or 20.50% were also identity\_hate.

8449 obscene comments. (5.29% of all data.)

• 7926 or 93.81% were also toxic.

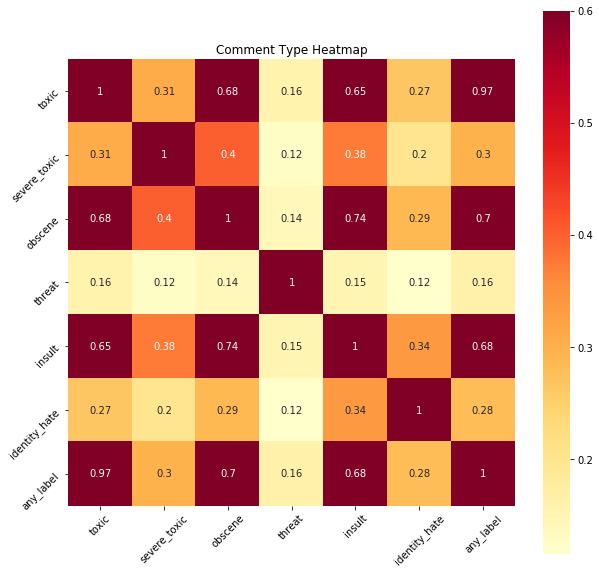
• 1517 or 17.95% were also severe\_toxic.

• 301 or 3.56% were also threat.

• 6155 or 72.85% were also insult.

• 1032 or 12.21% were also identity\_hate.

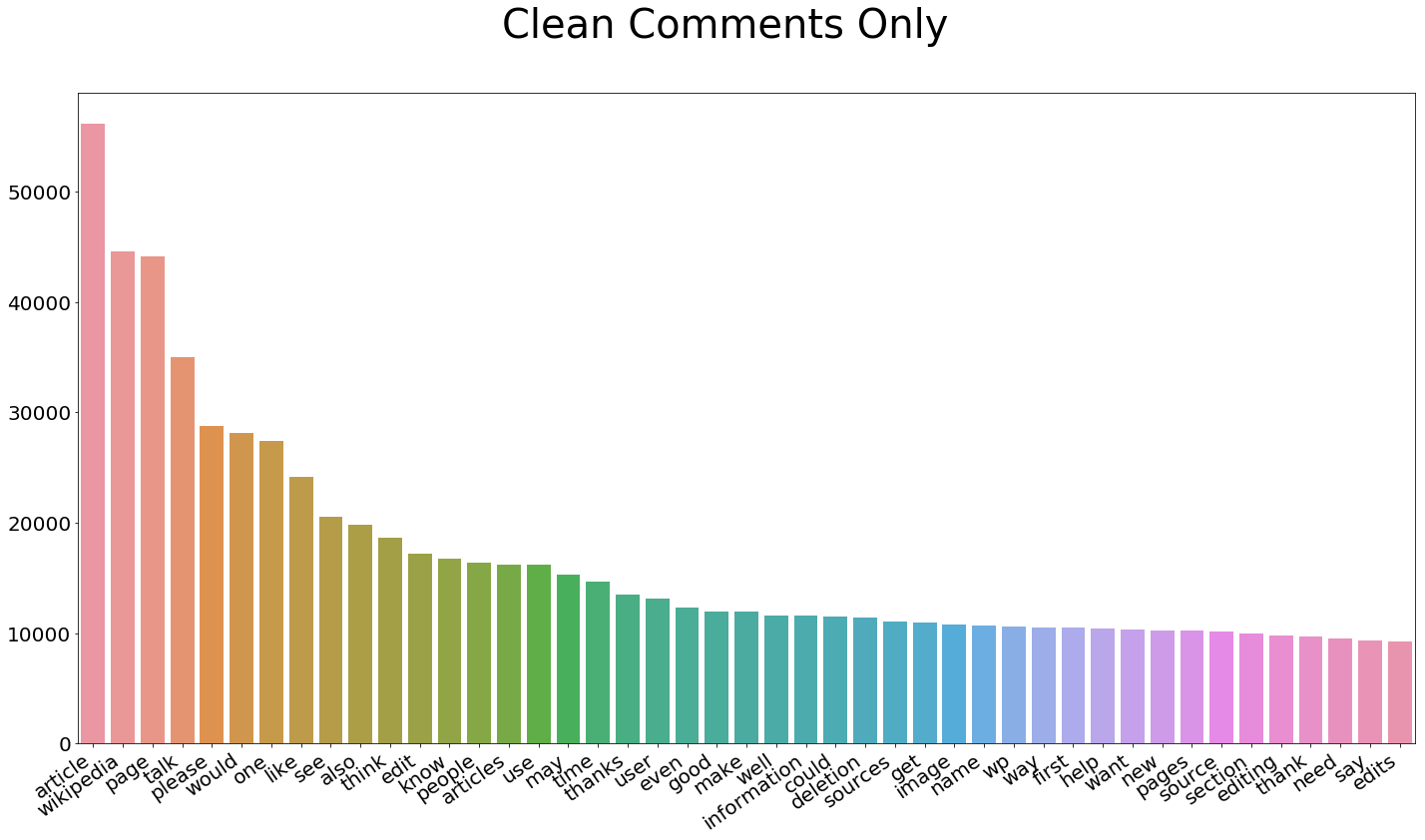
The correlation matrix below provides more insight into these overlapping categories. Threats are not likely to be severely toxic, nor are they likely to be racist or homophobic. But insults are often obscene, and identity hate really doesn’t have much overlap at all. I believe the categories with significant overlap will be more difficult to predict, as they’ll have similar contributing features, but “identity\_hate” will have more unique attributes and be easier to predict.



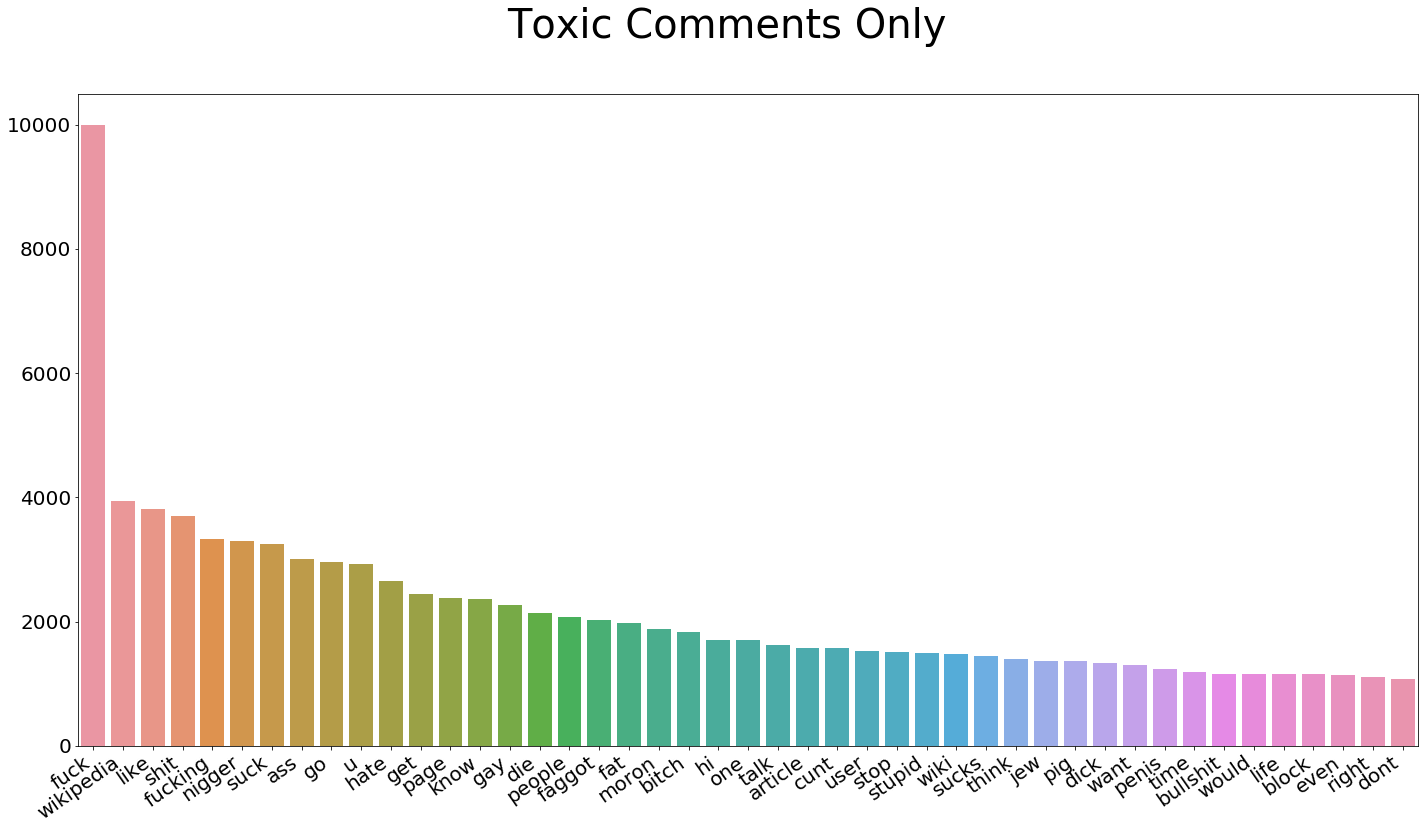
# Toxic samples:

“your a cocksucker u can't do anything to me”

what does a clean Wikipedia comment look like? I used a tokenizer with the standard stopwords to get the overall count of individual words and plotted the top 45.

Fig 4: Non-Toxic Comment Word Frequency

Article, page, please, think, edit, etc. The highest frequency words are about what you would expect from people discussing Wikipedia page edits and policy. Now let’s look at what a bad comment looks like. Can you find the top two words from the clean comments? “Wikipedia” surprisingly comes in second, and “article” falls back quite a few spaces. The difference in the highest frequency vocabulary is stark.

 Fig 5: Toxic Comment Word Frequency

In addition to the words themselves, I’ve extracted some other attributes of the comments that show contrast between toxic and clean comments.

* Capitalization

Toxic comments are more likely to be either in all caps or have no capitalization at all. In an average clean comment, 5% of the characters are capital letters. In toxic comments, that number jumps to 14%.

* Comment Length

On average, clean comments are about a third longer than toxic comments. The average character count for clean comments is 404, while toxic comments average 303 characters.

* Word Length

This one is a little closer, but there is a difference. Toxic comments average 4.1 characters per word, where clean comments average 4.4

* Question Marks

My thought here is that more legitimate posts might have more question marks. But that assumption was wrong, as toxic comments have 50% more question marks per comment than clean comments. 0.6 versus 0.4.

• Exclamation Marks

I made the opposite assumption about exclamation marks, and that paid off! Toxic comments have an average of 3.5 exclamation marks, while clean comments only have 0.3. This could be a very useful feature.

# Algorithms and techniques:

As a natural language processing problem, is a classification task that involves high dimensionality data. I will vectorize the data and test multiple classification algorithms.

I will vectorize the text data using the term frequency – inverse document frequency (tf-idf) statistic.

This technique takes into account not only the frequency of words or character n-grams in the text, it also takes into account the relevancy of those tokens across the dataset as a whole. The inverse document frequency reduces the weight of common tokens while boosting the weight of more unique tokens. I will establish a benchmark for performance with the top 10,000 words, and the number of tokens and the mix of words and character n-grams will be a parameter to tune for higher performance later on. I will also create a number of engineered features containing various attributes of the comment text, such as average word length, capitalization, and number of exclamation points. I will run the benchmark test without these features and experiment with them to optimize the solution. With the benchmark vectorization and features, I will experiment with multiple algorithms with default parameters to determine the most effective approach to the problem. The models I will use are

* Logistic Regresssion (Benchmark)

• Multinomial Naive Bayes

• Support Vector Machine

• Support Vector Machine with Naive Bayes Features

• Light GBM Recurrent neural networks work well on this problem and top

# Baseline:

The baseline model is a Logistic Regression model fit to tf-idf vectorized comment text with using only words for tokens, limited to 10,000 features. The target value is any\_label, but I also want to track the model’s performance on specific categories. But I do think that it’s important to note that my personal difficulty in perceiving the difference in specific categories when looking through comments is reflected in the model’s ability to specifically predict severe\_toxic, threat, and identity\_hate labels.

The cross-validated F1 scores for each label break down as follows:

toxic score:0.7192

severe\_toxic score: 0.3224

obscene score: 0.7452

threat score: 0.2069

insult score: 0.6277

identity\_hate score: 0.2772

any\_label score: 0.7299

I also tested a baseline with the engineered features included.

toxic score: 0.7235

severe\_toxic score: 0.3475

obscene score: 0.7440

threat score: 0.2029

insult score: 0.6274

identity\_hate score: 0.2768

any\_label score: 0.7328

# Data Preprocessing:

**Cleaning** :

The dataset is relatively clean. There is a minor data leak, IP addresses appended to some comments. For various reasons, mainly that this data may slightly compromise the model’s ability to generalize to new data, I’ve used a regular expression to strip all IP addresses from comments.

**Feature Engineering:**

During the exploratory data analysis, I found that many attributes of comments outside of the words themselves may be useful in predicting whether they are toxic. The features I added to the dataset are:

• Comment length in characters

• Percent of letters in a comment that are capitalized

• Average length of words in a comment

• Number of exclamation marks in a comment

• Number of question marks in a comment

**Vectorization:**

As discussed previously, I am using a term frequency – inverse document frequency (tf-idf) statistic to vectorize text. The number of features and presence of character n-grams is a parameter to tune for model optimization.

**Feature Scaling**:

The engineered features are normalized from 0.0 to 1.0. The tf-isf features are not scaled. Implementation Finding the Best Algorithm To compare the relative performances of each algorithm, I’m going to test them on the same preprocessed data as the benchmark, a tf-idf vectorized data with 10,000 features. The target is any\_label, and the scores are F1 scores with five-fold cross validation.

• Logistic Regression: 0.7299

◦ With engineered features: 0.7328

◦ Engineered feature boost: 0.39%

• Multinomial Naive Bayes (NB): 0.6670 (0 predictions for threat category)

◦ With engineered features: 0.6734 (0 predictions for threat category)

◦ Engineered feature boost: 0.96%

• Support Vector Machine (SVM): 0.7703

◦ With engineered features: 0.7739

◦ Engineered feature boost: 0.47% boost

• Support Vector Machine with Naive Bayes Features (NB-SVM): 0.7804

◦ With engineered features: 0.7842

◦ Engineered feature boost: 0.49%

• LightGBM\*: 0.7470

◦ With engineered features: 0.7573

◦ Engineered feature boost: 1.38%

Not surprisingly, the engineered features gave the tree based model a larger performance boost than other models. The performance gain for other algorithms is modest, but it does come at an extremely minimal cost in terms of processing resources and training time so I would consider them worthwhile. In terms of overall performance, NB-SVM is the strongest performer and SVM comes in at a close second. The naive bayes features gave SVM a 1.31% performance boost at a cost of about five times the training time. Around thirty-five seconds versus a little over three minutes, but the difference is minimal at prediction time and the gain is significant.

LightGBM also showed strong results with some tuning, but there is an issue with the model that prevents it from being a practical solution unless it’s performance is significantly higher than other models. Like all treebased models, dense data is a firm requirement. Refining the text vectorization strategy is an important aspect of optimizing a natural language processing solution. Linear regression and support vector machines can run on sparse matrices which reduces their memory footprint, but LightGBM will see memory grow too fast converting sparse matrices into dense ones.

# Refinement of the NB-SVM Model:

Step 1: Optimize tf-idf vectorizer

Experiment with word and character level ngrams and maximum feature counts. All other parameters are fixed, and refinement steps will all use five-fold cross validation.

Word Ngram -----Word Max Features ---Char Ngram-- Char Max Features ---F1 Score

1 ----10,000---- 0 ----0----- 0.7842

1 ----30,000 ----0---- 0--- 0.7926

1-2---- 30,000--- 0 ----0---- 0.7877

2-6---- 30,000---- 0 -----0----- 0.4042

0 ---0--- 3-7---- 20,000 ----0.7742

1---- 5,000 ----3-7 ----5,000---- 0.7962

1-2 ----15,000 ----3-7 -----5,000 ----0.7985

1-2---- 20,000 ----3-7----- 10,000 ---0.8005

1-2--- 20,000---- 3-5---- 10,000---- 0.8015

1-3---- 20,000----- 3-5---- 10,000 ---0.8013

The ideal seems to be a 20,000 word ngrams in range 1-3 and 10,000 char ngrams in range 3-5, in a tie with the same settings but a 1-2 word ngram. An added benefit of these parameters is that this is a relatively fast vector to calculate.

Step 2: Tune NB feature weight

The NB feature transformer class has a parameter ‘epsilon’ that controls the influence of the feature level probabilities on the input features. The default parameter is 1.0.

NB Weight --- F1 Score

0.1--- 0.7966

0.2 ----0.7989

0.3--- 0.8003

0.4 ---0.8001

0.5 ----0.8008

0.6 ---0.8013

0.7 --0.8007

0.8--- 0.8020

0.9 ---0.8019

1.0 ---0.8015

The weight of the features transformed by Naive Bayes doesn’t have much effect of the model’s performance.

Step 3: SVM Parameter Tuning

Parameter 1:

Kernel SVMs have a number of parameter options. The root option is the kernel, and other options stem from there. So far, we’ve been using a linear kernel. A linear kernel fits best to datasets where the the dataset is linearly separable, and because this model vastly outperformed a well-tuned boosted tree model (LightGBM) I assume that it has a reasonable level of linear separation.

Parameter 2:

Penalty for error term Going with the linear kernel, there is only one major parameter to tune: C. This is the coefficient for the L2 penalization. The default is 1.0, and I will test 0.5, 0.7, and 0.9

C ----- F1 Score

0.3 ----0.8049

0.5----- 0.8054

0.8----- 0.8032

1.0------ 0.8015

Setting C to 0.5 instead of the default 0.8015 gives a 0.51% performance increase.

# Results:

# Model Evaluation and Validation:

The model has been trained, tested, and optimized using training and test subsets of the data. I will use an unseen holdout subset of the data to evaluate the model.

The F1 Score on the holdout data is 0.8072.

# Justification:

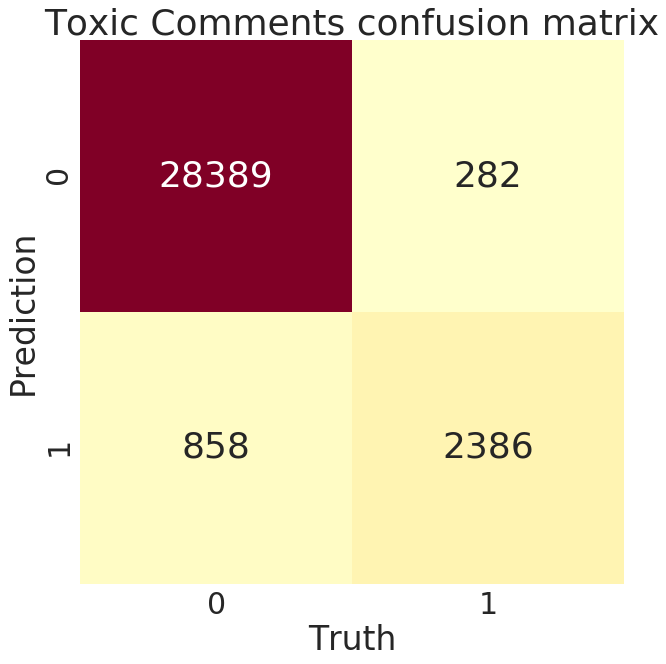
The final model offers a significant performance boost over the benchmark linear regression model, about 11%. So far we’ve been talking in the abstract about F1 Scores, but now let’s dig into the real world performance and what those numbers actually mean.

This model has 96% accuracy. Now on the surface, that sounds great. But since this is a highly imbalanced dataset, that doesn’t mean a lot. In fact, if I had just created a model that predicted “0” for every single item, it would get an accuracy of 90%.

The real metric of how well the model performed at predicting a toxic comment is recall. This model achieved a recall score of 0.74, which means that it correctly 74% of the actual toxic comments as toxic. That may seem low, but there’s a catch. If we predict every result as “1,” we’ll get 100% recall.

As discussed before, the F1 Score provides a target that helps a model find the nuance in an imbalanced dataset between catching the positive results without focusing on them to a point where the usefulness of the model suffers. A confusion matrix can illustrate the concept of balancing true positives and true negatives, as well as accuracy, recall, and precision.

Overall, I do believe that this model is robust enough for this application and it offers a large advantage over both the standard approach of human flagging for review (though I wouldn’t eliminate that as a feature) and an out-of-the-box model.



# Conclusion:

Reflection

The process for this project was as follows:

1. Analyze the problem and propose a useful solution.

2. Explore the dataset to get a better picture of how the labels are distributed, how they correlate with each other, and what defines toxic or clean comments.

3. Develop an objective that fits a practical use case and addresses the major class imbalance.

4. Create a baseline score with a simple logistic regression classifier.

5. Explore the effectiveness of multiple machine learning algorithms.

6. Select the best model based on a balance of performance and efficiency.

7. Refine the preprocessing strategies to optimize model performance.

8. Tune model parameters to maximize performance.

9. Build a the final model with the best performing algorithm and parameters and test it on a holdout subset of the data.

The final model offered about 11% performance gain over the initial benchmark model, which makes it an effective solution to the problem. Even more so considering that the current system in place was hand-labeling by users via a reporting function.

# Improvement:

I believe that there are a number of ways that the solution could be improved.

Recurrent neural networks, despite their increased overhead, could be a very effective solution if GPU resources are available for quick predictions.

Another great strategy could be mixed models, a sort of divide an conquer method where the problem is divided into multiple smaller, contextual problems. While the solution laid out here generalizes to the entire dataset, no one solution will be able to generalize perfectly to the diverse variety of inputs you’ll get from Internet users. By training models on different situations, like a model that’s only been trained on short or long comments, to only detect whether a comment is toxic when profanity is present, etc, and storing them in memory, you could use a simple decision tree to feel comments into the model that would be most effective. A few that I can think of are

• Short comments

• Long comments

• “Hot” threads where the rate of commenting is high and emotions may be high

• Comments with profanity (A general model might flag profanity as toxic, where a model trained only on comments where profanity is present may pick up on more nuance.)

• Comments by a user who has already been flagged as toxic in the past