Machine Learning Engineer Nanodegree

Capstone Project

Toxic Comment Classification Challenge

Ishan Soni
December 26th, 2018

I. Definition

Project Overview

My project draws upon the domain of document classification with an emphasis on the natural language processing component of document classification. Machine learning has been successfully used to classify documents by topic for several decades. However, machine learning techniques do not perform as well when performing sentiment analysis which requires the parsing of more complex language structures. This is where ideas from natural language processing must be applied. My aim with this project is to make progress towards solving a complex natural language processing problem using machine learning. I think that it would be very beneficial if computers could interpret and produce the same kind of natural language of which even young children are capable. This would allow machine learning to be applied to a wider variety of tasks than it is currently capable of solving. In particular, machine learning could be applied to many problems that are not well structured and for which there is not much training data.

This capstone project is based on the Toxic Comment Classification Challenge

Discussing things you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) are working on tools to help improve online conversation. One area of focus is the study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion). So far they've built a range of publicly available models served through the Perspective API, including toxicity. But the current models still make errors, and they don't allow users to select which types of toxicity they're interested in finding (e.g. some platforms may be fine with profanity, but not with other types of toxic content).

With more people joining social media than ever before, it becomes imperative that this problem is solved. Classifying toxic comments (obscene, threat, insult, identity-based hate) will be the core of this project.

Problem Statement

In this project, I will build a multi-headed model that will be capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate from a given comment. I'll be using a dataset of comments from Wikipedia's talk page edits which have been labeled by human raters for toxic behavior. Improvements to the current models will hopefully help online discussion become more productive and respectful.

This is a supervised multi-class classification problem and as such different measurable evaluation metrics can be applied. I'll be using the AUC-ROC metric to measure the efficiency of my models. Given a comment, the solution will be a machine learning model that receives as input the comment and outputs a class probabilty for every toxicity type [*toxicity_type*] : toxic, severe_toxic, obscene, threat, insult, identity_hate.

<u>Example 1</u>
Input Text : *You, sir, are my hero!*
Output will be : toxic : 0.1 , obscence : 0.0, ..
The output states that the algorithm thinks that there is a 0.1 probability or 10% chance that the given text is toxic. Similarly the algorithm thinks that there is 0 probability or 0% chance that this text is obscene.

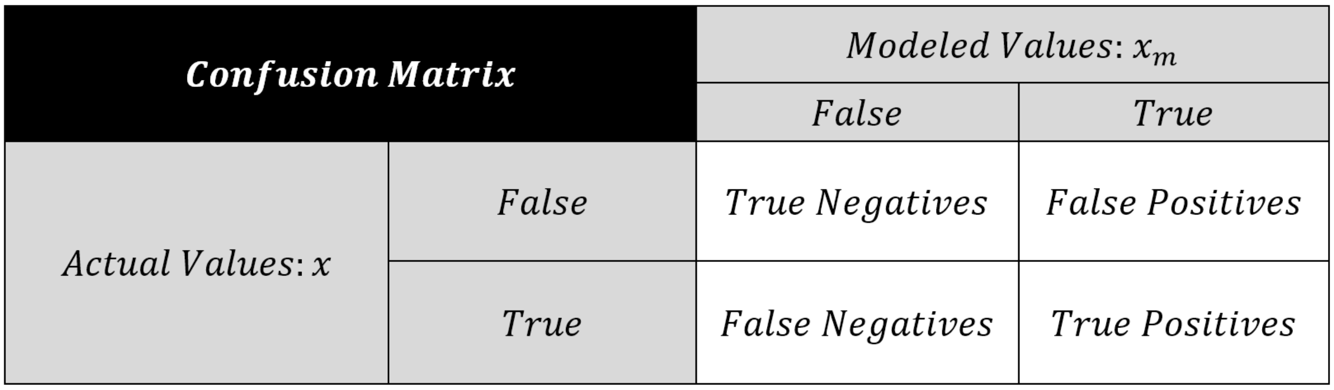
<u>Example 2</u>
Input Text : *Oh my, you are so fuckable!*
Output will be : toxic : 0.1 , obscence : 0.8, ..
The output states that the algorithm thinks that there is a 0.1 probability or 10% chance that the given text is toxic. Similarly the algorithm thinks that there is 0.8 probability or 80% chance that this text is obscene. These class probabilities will enable us to make many smart decisions. Such a comment should be fine on a platform that tolerates obscenity. Such platforms can use this class probability and mark this post as *NSFW*. Other platforms that don't tolerate obscenity can choose to delete such comments automatically.

Generally, if the class probability for a particular type of toxicity is > 0.5, we will label it as a toxic comment with that toxicity type.

Metrics

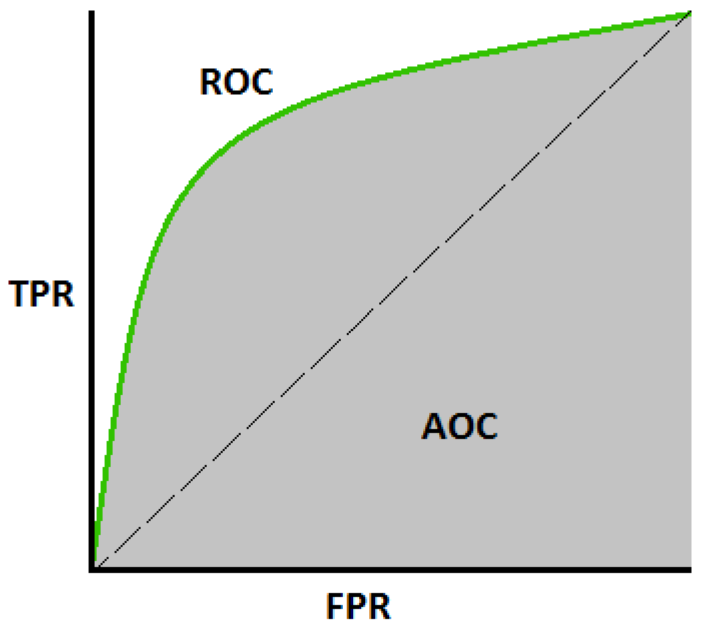
There are many different metrics that could be used to evaluate the solution, given that this is a multi-class classification problem. The most common metric is accuracy. We can create a confusion matrix for each toxicity type and study the algorithm's accuracy for that toxicity type.

Generally, if the class probability for a particular type of toxicity is > 0.5, we can label it as a toxic comment with that toxicity type. Therefore we can easily build a confusion matrix for each toxicity class.



Confusion Matrix		*Modeled Values*: x_m	
		False	*True*
Actual Values: x	*False*	*True Negatives*	*False Positives*
	True	*False Negatives*	*True Positives*

But since we are more interested in probabilities and we have defined our solution as : *Given a comment, the solution will be a machine learning model that receives as input the comment and outputs a class probabilty for every toxicity type*, an evaluation metric like the **AUC - ROC Curve (Area Under the Receiver Operating Characteristics)** makes more sense. AUC - ROC curve is a performance measurement for classification problems at various thresholds settings. ROC (Receiver Operating Characteristics) is a probability curve and its AUC(Area Under Curve) represents the degree or measure of separability. It tells us how much a given model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. The ROC curve is plotted with TPR(True Positive Rate) against the FPR(False Positive Rate) where TPR is on the y-axis and FPR is on the the x-axis.

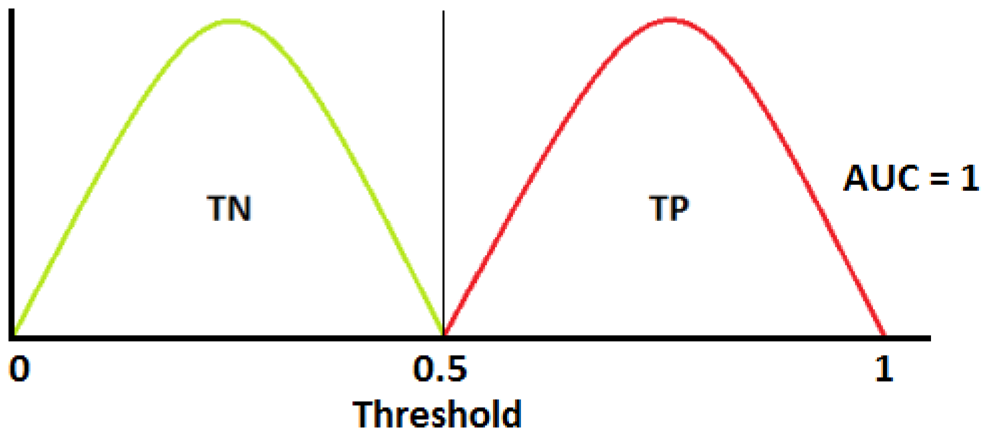


$$\text{TPR /Recall / Sensitivity} = \frac{\text{TP}}{\text{TP + FN}}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN + FP}}$$

$$\text{FPR} = 1 - \text{Specificity} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

An excellent model has AUC near to 1 which means it has a good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means our model has no class separation capacity whatsoever.



Since our models will output probabilities, the AUC-ROC which is based on probability distributions will prove to be the correct metric. We will be using the AUC-ROC score while we cross validate our models.

II. Analysis

Data Exploration

I'll be using a dataset of comments from Wikipedia's talk page edits which have been labeled by human raters for toxic behavior.

The types of toxicity are:

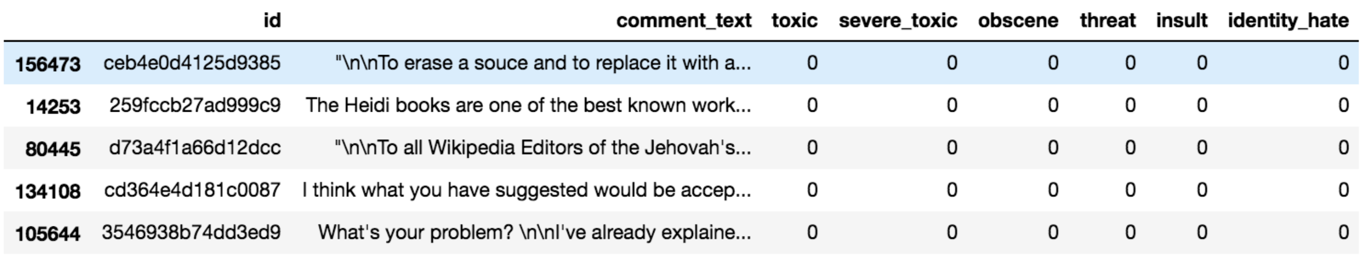
- toxic
- severe_toxic
- obscene
- threat
- insult
- identity_hate

Both the train and test datasets are in the form of a csv and are stored in the data directory. The train dataset has *159571* observations and the test dataset has *153164* observations. Each train observation corresponds to a different comment. There are a total of 8 variables/columns in this dataset and they are described below :

Variable	Data Type/ Values it can take	Description
id	object/string	Unique identifier
comment_text	string	The comment text that is rated/td>
toxic	int64/Boolean. The possible values are 0 and 1	Specifies whether the given comment is toxic or not.
severe_toxic	int64/Boolean. The possible values are 0 and 1	Specifies whether the given comment is severly toxic or not.
obscene	int64/Boolean. The possible values are 0 and 1	Specifies whether the given comment is obscene/vulgar.
threat	int64/Boolean. The possible values are 0 and 1	Specifies whether the given comment is a threat.
insult	int64/Boolean. The possible values are 0 and 1	Specifies whether the given comment is an insult.
identity_hate	int64/Boolean. The possible values are 0 and 1	Specifies whether the given comment is identity hate.

The train and test dataset *do not* contain any null values.

Sample Data :



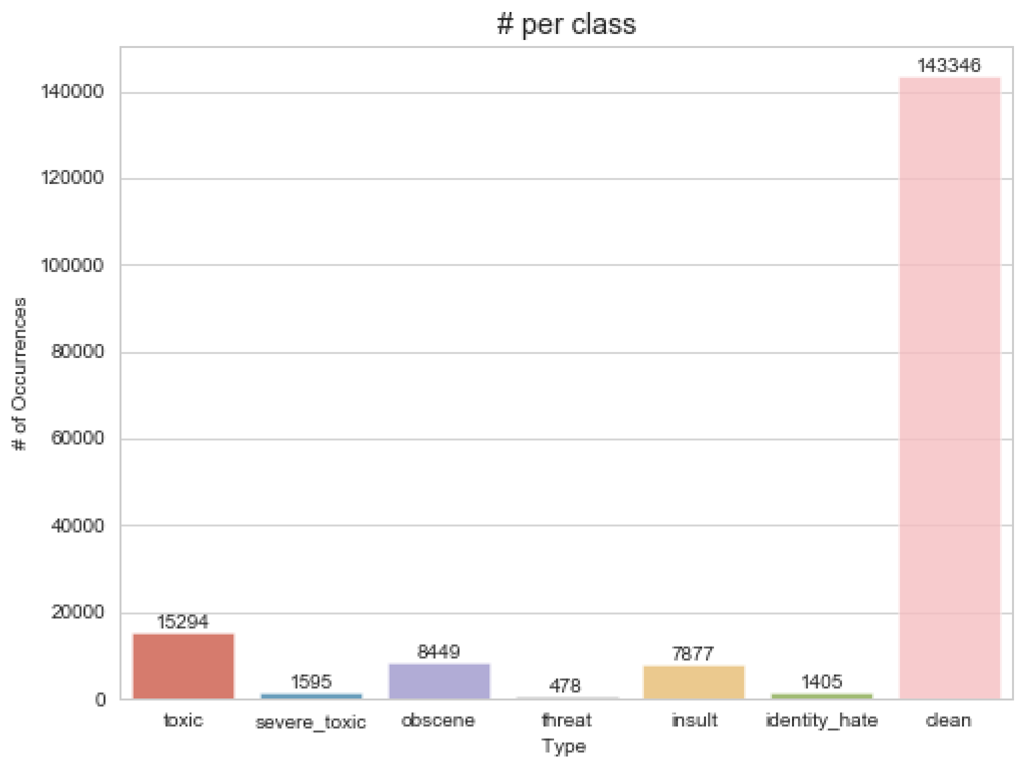
	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
156473	ceb4e0d4125d9385	"\n\nTo erase a souce and to replace it with a...	0	0	0	0	0	0
14253	259fccb27ad999c9	The Heidi books are one of the best known work...	0	0	0	0	0	0
80445	d73a4f1a66d12dcc	"\n\nTo all Wikipedia Editors of the Jehovah's...	0	0	0	0	0	0
134108	cd364e4d181c0087	I think what you have suggested would be accep...	0	0	0	0	0	0
105644	3546938b74dd3ed9	What's your problem? \n\nI've already explaine...	0	0	0	0	0	0

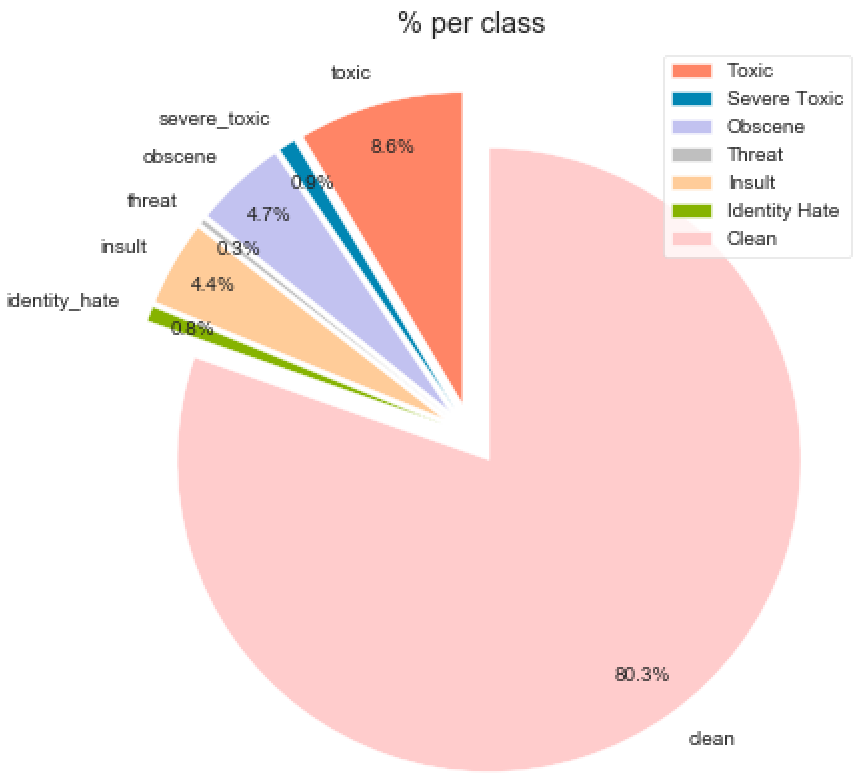
The distribution of different toxicity types are as follows :

Class/Toxicity Type	Percentage (%)
Clean	80.3
Toxic	8.6
Severe Toxic	0.9
Obscene	4.7
Threat	0.3
Insult	4.4
Identity hate	0.8

Exploratory Visualization

The below images shows the distribution of toxicity type in our train dataset.



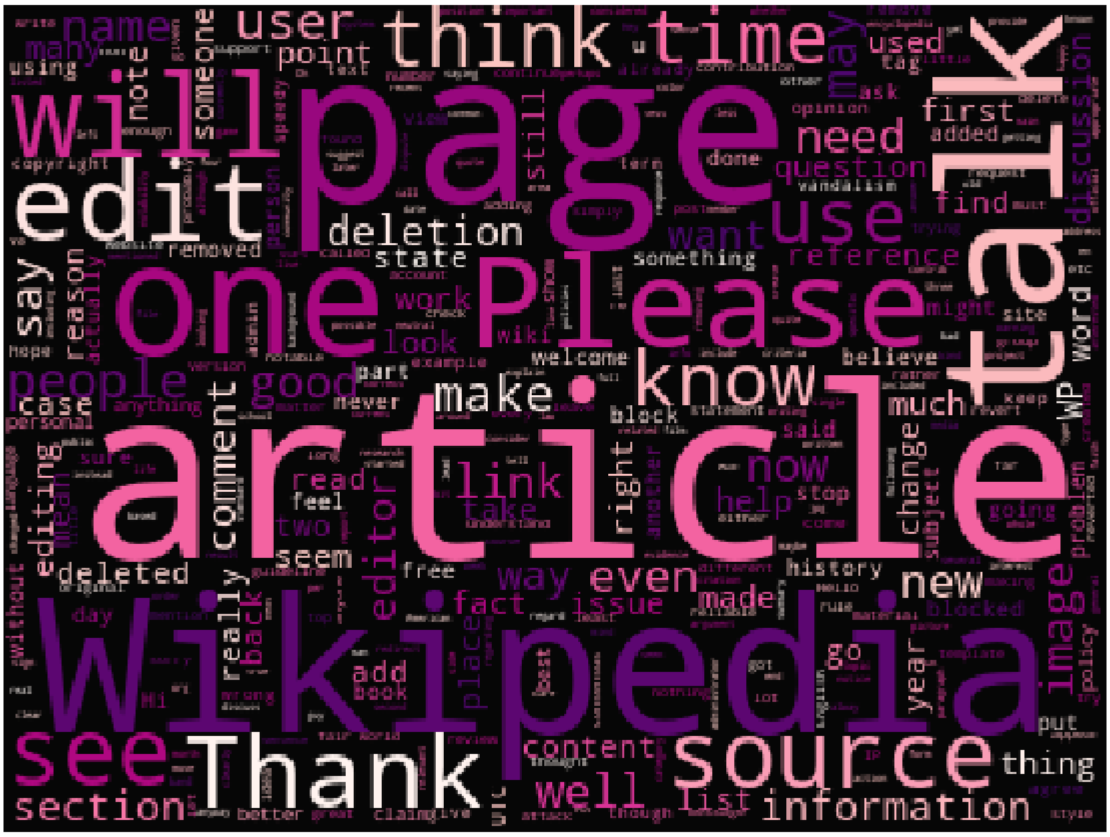


The above figures support our decision of building a multiheaded classification model. The distribution of toxicity types is very uneven. The toxicity is not evenly spread out across classes.

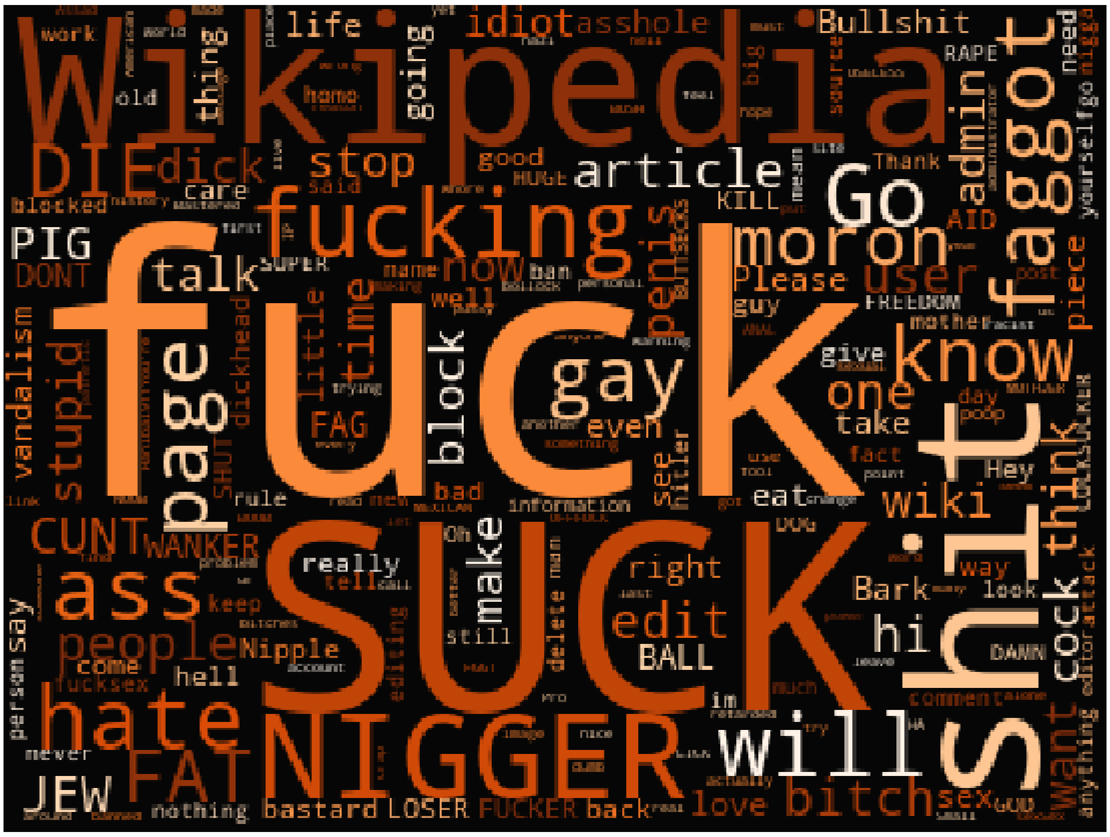
Hence we might face class imbalance problems.

Lets look at some common words for different toxicity types

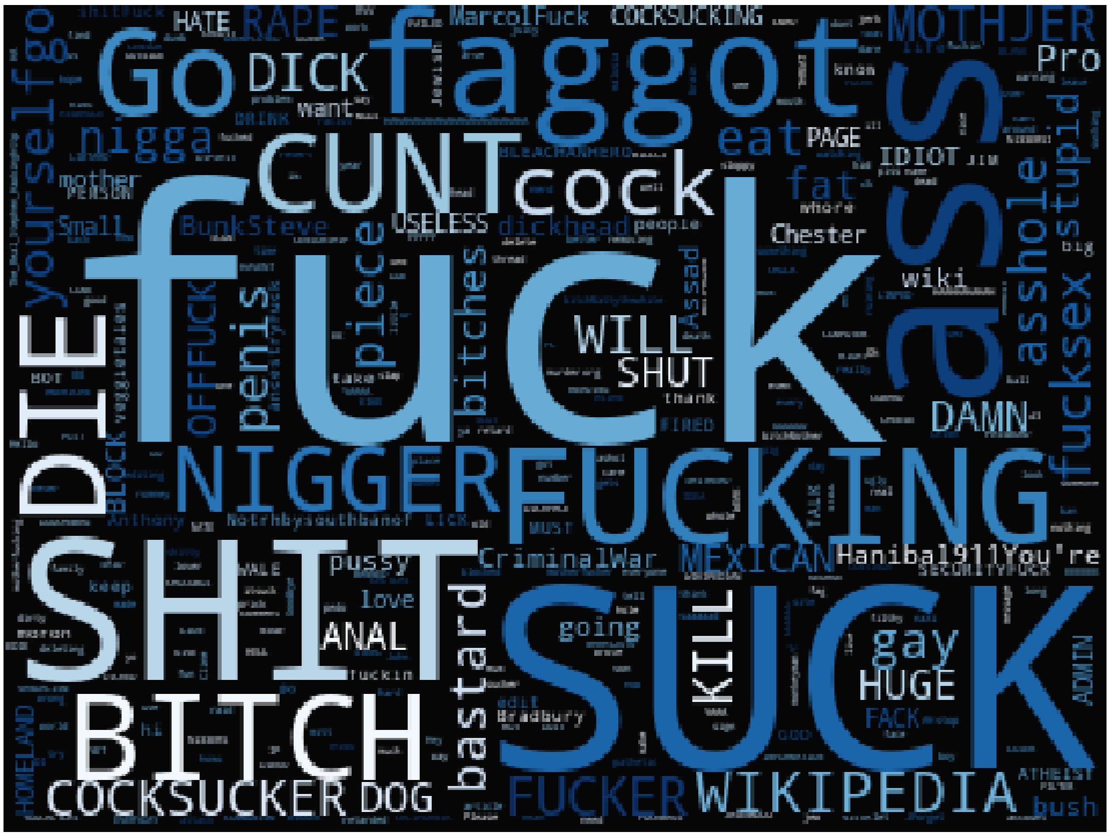
1. Common words in clean comments :



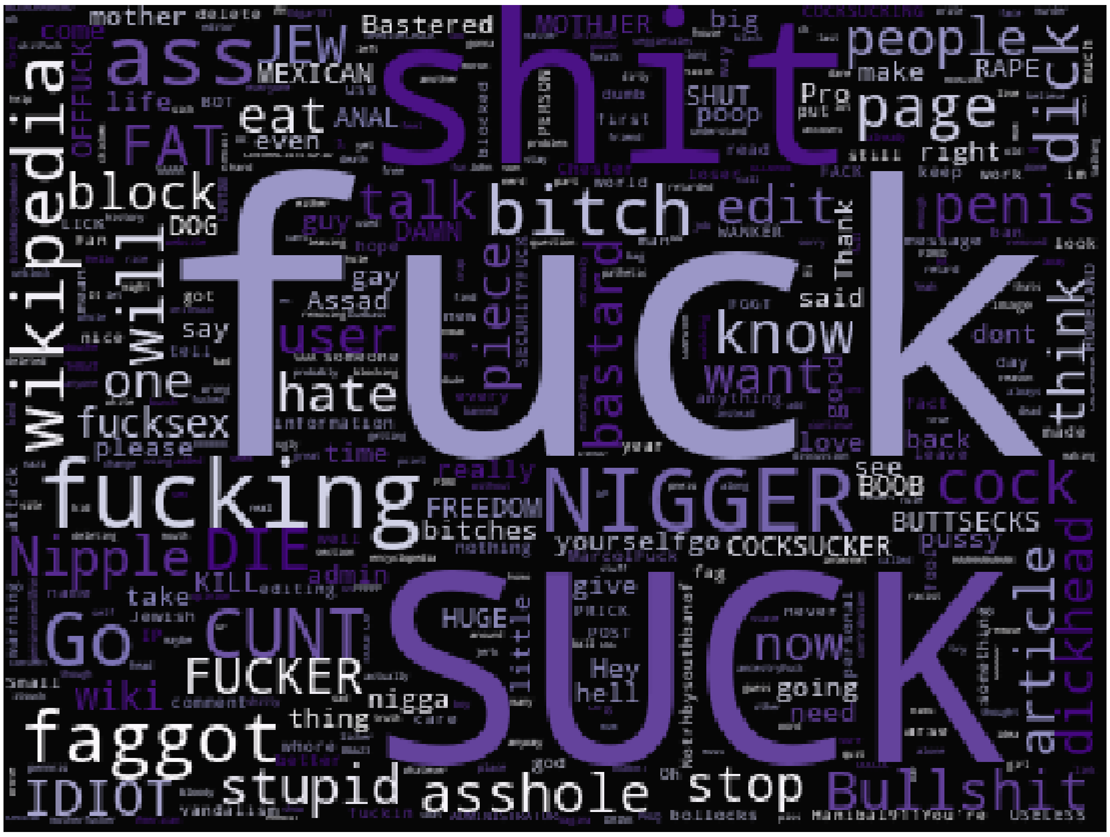
2. Common words in toxic comments :



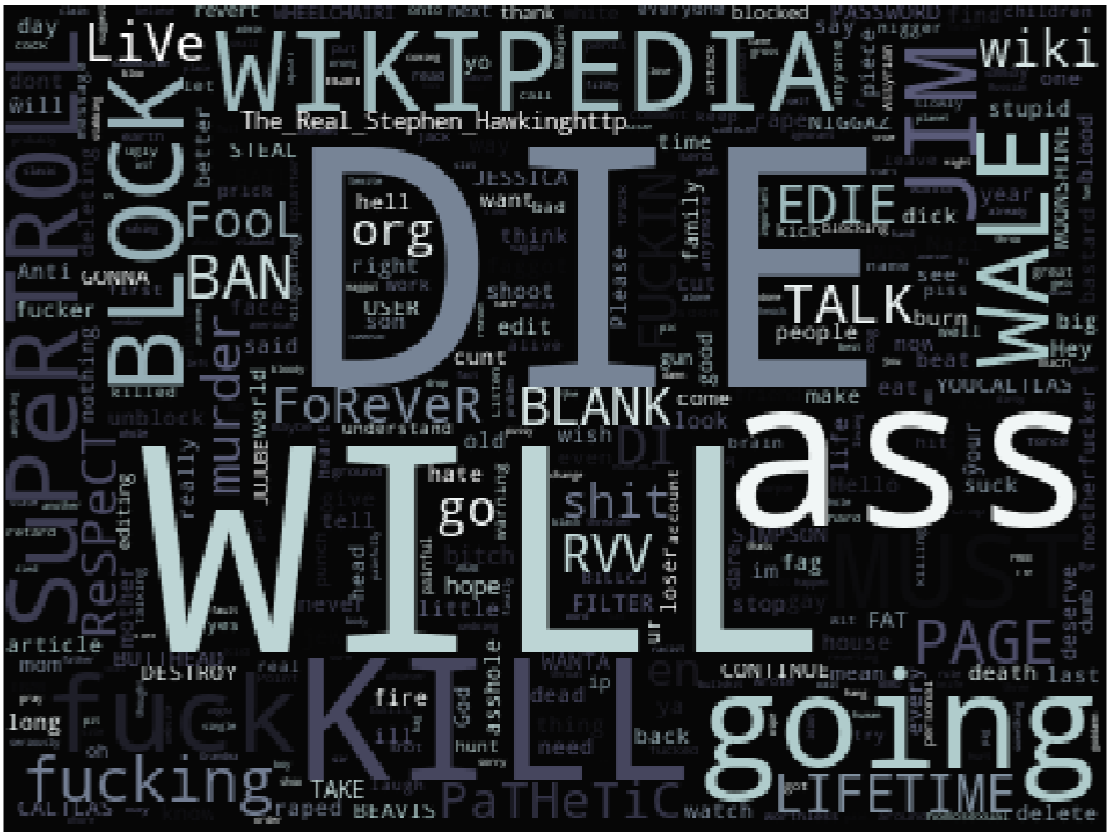
3. Common words in severe toxic comments :



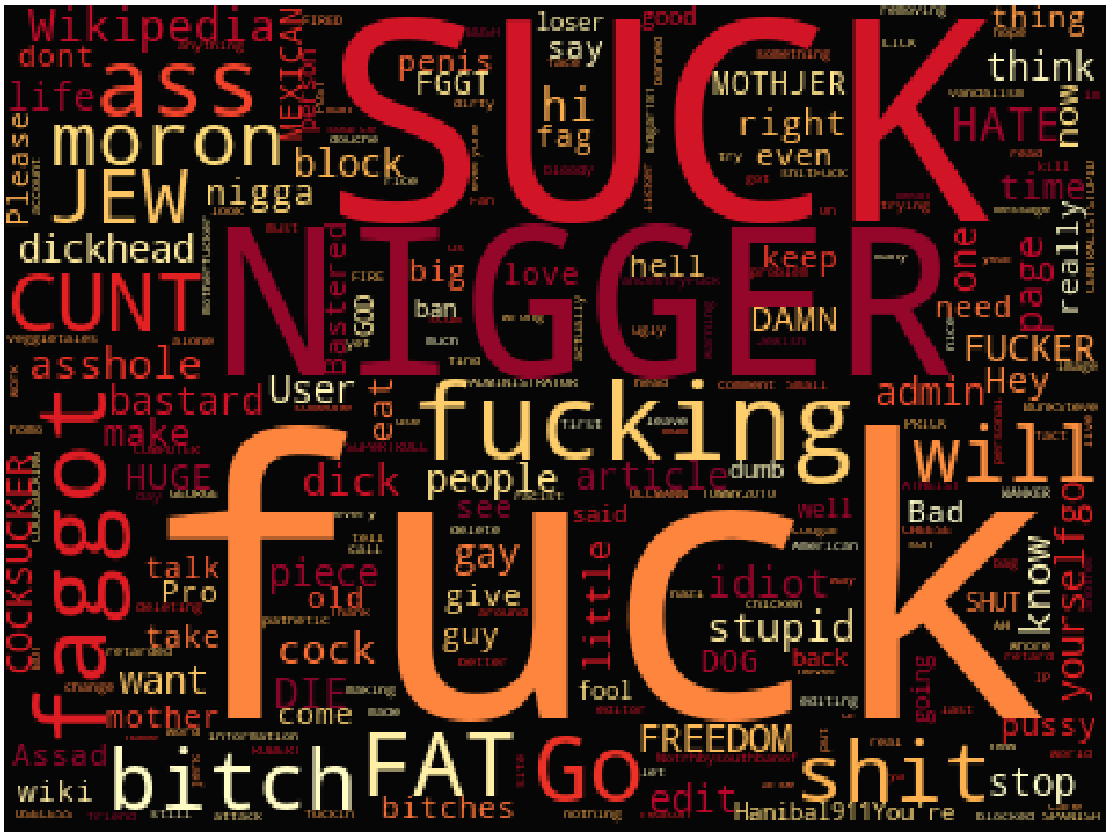
4. Common words in obscene comments :



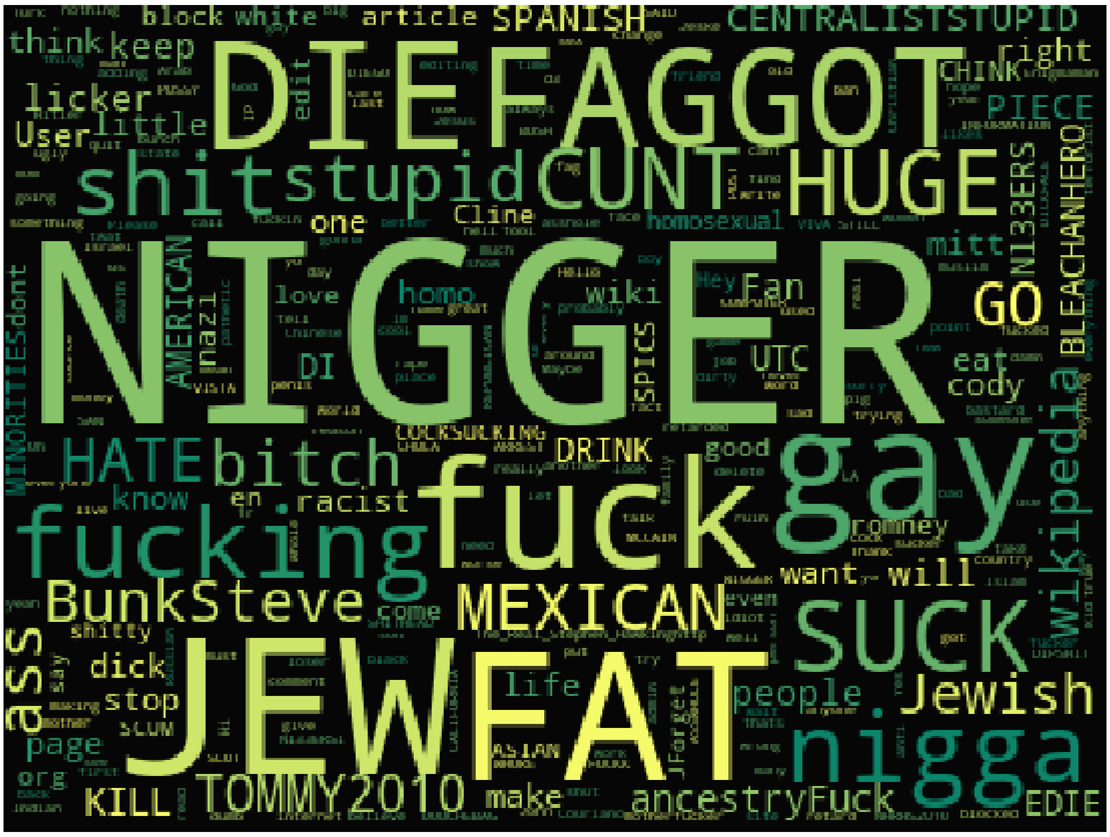
5. Common words in threat comments :



6. Common words in insult commnents :



7. Common words in identity hate comments :



From the above visualizations, we can see the most common words for each toxicity type. Since this dataset is taken from wikipedia talk's page edits, we can see that wikipedia is a common term in all toxicity types and can be removed since it doesn't help us at all in our analysis. Also, we can see a few usernames in the words too. We should also clean them.

Algorithms and Techniques

Since we have defined our solution as : *Given a comment, the solution will be a machine learning model that receives as input the comment and outputs a class probabilty for every toxicity type*, I will use machine learning algorithms that can output probabilities. I will be employing two algorithms to solve the problem : linear models and MultinomialNB (Naive Bayes classifier for multinomial models).

1. The Linear Logistic Regression model will form my benchmark. I will also use it for my actual model. Linear logistic regression is a good benchmark because it is simple and does not require the tuning of many hyperparameters. It is also fast to train and robust to noisy data. The expectation is that the more complex Logistic Regression model (after hyperparameter optimization and feature transformations) and MultinomialNB model should achieve superior performances.

2. Naive Bayes is a family of algorithms based on applying Bayes theorem with a strong(naive) assumption, that every feature is independent of the others, in order to predict the category of a given sample. They are probabilistic classifiers, therefore will calculate the probability of each category using Bayes theorem. The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts

such as tf-idf may also work. We do have other alternatives when coping with NLP problems, such as Support Vector Machine (SVM) and neural networks. However, the simple design of Naive Bayes classifiers make them very attractive for such classifiers. Moreover, they have been demonstrated to be fast, reliable and accurate in a number of applications of NLP.

Benchmark

The Linear Logistic Regression (LogisticRegression) from the sklearn.linear_model package will form my benchmark. Linear logistic regression is a good benchmark because it is simple and does not require the tuning of many hyperparameters. It is also fast to train and robust to noisy data.

This benchmark model is trained on a Bag of Words representation of our data.

The Bag Of Word representation was created during the sklearn.feature_extraction.text.CountVectorizer. It was instantiated using the following parameters:

Argument	Value	Explanation
analyzer	"word"	For creating word features
min_df	5	Only retain terms/words that appear in atleast 5 comments
ngram_range	(1, 1)	Create ngrams of 1 word each
max_features	10000	Only include the top 10000 words ordered by word frequency.

We train this model on our bag of words representation and use cross validation to ensure our model is not overfit. The average cross-validated AUC-ROC score we get for this benchmark model on our train data is around **0.94**. This benchmark model gives us a Kaggle score of **0.93**.

III. Methodology

Data Preprocessing

Before we predict probabilities for each toxicity class using our machine learning algorithms, we need to convert our raw comment text into a form that can be consumed by these algorithms. Below are the two vectorized forms I have used :

1. Bag of Words : In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, where the (frequency of) occurrence of each word is used as a feature for training a classifier. Bag-of-words

2. Tfidf : TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Tf-idf

Both of these vectorized representations are very sparse matrices. Therefore we need to reduce the number of features we have in these representations by reducing our vocabulary. We do this by removing words that are not relevant and by converting different words that convey the same meaning into a single same word.

As identified during Data Exploration, there were a lot of / n's and a lot of usernames in our comment text. We have removed these unnecessary words during our Data Cleaning phase.

```
train['comment_text'] = train['comment_text'].map(lambda x: re.sub('\\n','
',str(x)))

train['comment_text'] = train['comment_text'].map(lambda x: re.sub("\[\
[User.*",'',str(x)))
```

After this, we removed *stopwords* from our comment text and performed *stemming*. *Stopwords* are words like 'a', 'the', 'in' etc that occur very frequently in our data but do not contribute anything towards our analysis. *Stemming* is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form. Eg love, loved and loving mean the same thing and performing a stem operation on them will reduce all of them into the same root form 'lov'. This will further reduce our vocab and reduce the matrix sparsity of our bag of words/ ngrams representations.

The following function was used to remove stopwords and perform stemming :

```
import string
# nltk nlp library
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Stemmer
from nltk.stem import SnowballStemmer
s = SnowballStemmer("english")

def cleanComment(comment):
    comment = comment.translate(str.maketrans('', '', string.punctuation))
    words = [s.stem(w.lower()) for w in word_tokenize(comment) if
w.lower() not in stopwords.words("english")]

    return " ".join(words)
```

Implementation

I trained two models (Logistic Regression and MultinominalNB) on our new feature set which was created by combining word and char feature vectors.

The word features were created using the TfidfVectorizer from the sklearn.feature_extraction.text package. This class creates a tf-idf vectorized format (as opposed to the Bag of Words we used for our baseline model). It was instantiated using the following parameters :

Argument	Value	Explanation
analyzer	"word"	For creating word features
min_df	5	Only retain terms/words that appear in atleast 5 comments
ngram_range	(1, 1)	Create ngrams of 1 word each
max_features	10000	Only include the top 10000 words ordered by word frequency.
strip_accents	"unicode"	Strip accented characters eg à
token_pattern	r'\w{1,}'	Features will only be word characters [a-zA-Z0-9_]

The char features were also created using the TfidfVectorizer from the sklearn.feature_extraction.text package. People often try to obfuscate bad words with additional characters. Using character n-grams can potentially detect those. It was instantiated using the following parameters :

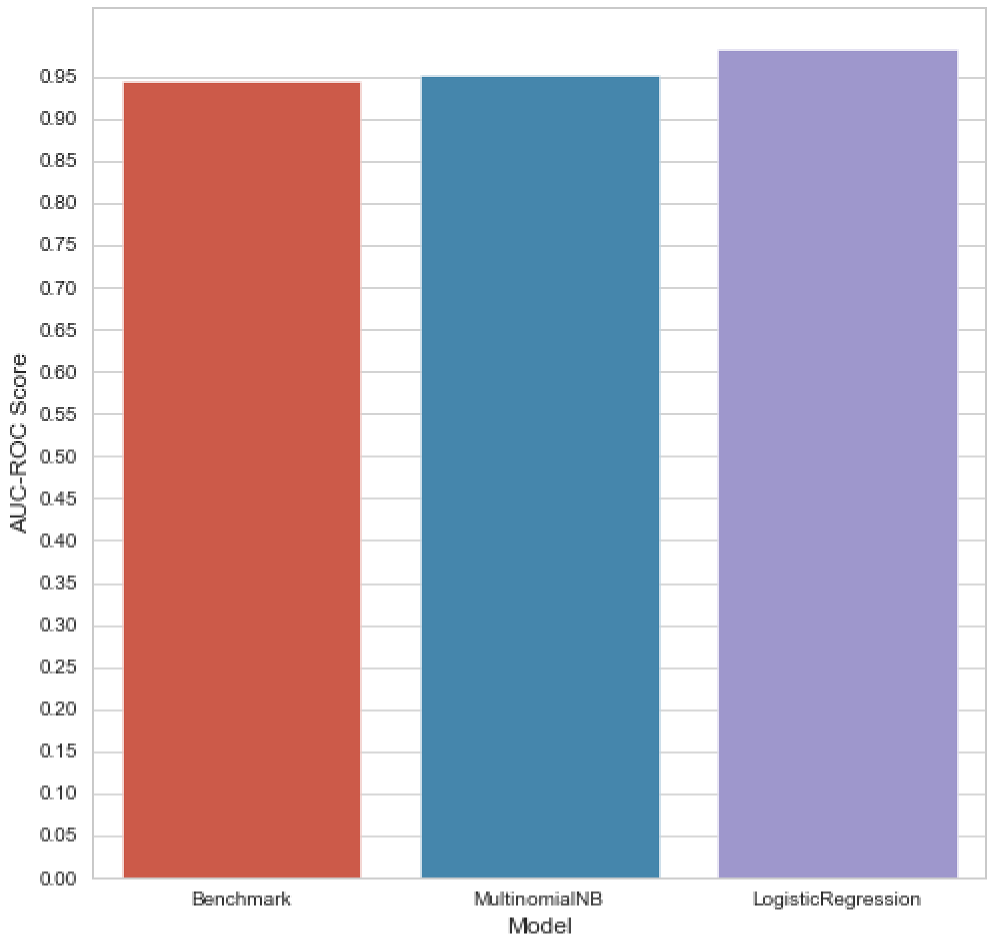
Argument	Value	Explanation
analyzer	"char"	For creating character features
ngram_range	(2, 6)	Create ngrams of 2-6 characters each
max_features	30000	Only include the top 30000 char n-grams ordered by frequency.
strip_accents	"unicode"	Strip accented characters eg à

These two feature vectors were combined using the *scipy.sparse.hstack* and then fed into our machine learning models.

1. *MultinominalNB* : The MultinominalNB model was developed using the sklearn package. The model was trained for each class individually using a for loop by fitting the new train feature set we created above and the individual toxicity class values. It was done using the fit() method. We then calculated the cross validated AUC-ROC score for each toxicity class. The AUC-ROC was then averaged over all toxicity classes to get the final score. The MultinominalNB model achieved an AUC-ROC score of **0.9522**. I also predicted the toxicity values for the test/holdout dataset using the model's predict() method and submitted the predictions to Kaggle. This model achieved a Kaggle score of **0.9487**.

2. *Logistic Regression* : The Logistic Regression model was again developed using the sklearn package. The model was trained for each class individually using a for loop by fitting the new train feature set we created above and the individual toxicity class values. It was done using the fit() method. We then calculated the cross validated AUC-ROC score for each toxicity class. The AUC-ROC was then averaged over all toxicity classes to get the final score. The Logistic Regression model achieved an AUC-ROC score of **0.982**. I also predicted the toxicity values for the test/holdout dataset using the model's predict() method and submitted the predictions to Kaggle. This model achieved a Kaggle score of **0.9778**.

Below is the score comparison of all our models :



Refinement

Our Logistic Regression model had the best AUC-ROC score. I then performed Grid Search on our Logistic Regression model to find the best set of hyperparameters given our feature vectors. I used the GridSeachCV from the sklearn.model_selection package to find the best hyperparameters.

Grid Search performs an exhaustive sweep through a manually specified subset of the hyperparameter space of a learning algorithm. The grid search algorithm was guided by the AUC-ROC performance metric, measured by cross-validation on the training set. The k value for cross validation was kept at 3.

The manually specified hyperparameters were :

1. C : Inverse of regularization strength. Smaller values specify stronger regularization. Values given [0.1, 0.5, 1.0]
2. solver : Algorithm to use in the optimization problem. Values given ['liblinear', 'sag']

The best hyperparameters came out to be C = 1 and solver = 'sag'. The final model created using the above hyperparameters has an average AUC-ROC of **0.983** and a Kaggle score of **0.9778**.

The improvement is almost negligible from the initial Logistic Regression model.

IV. Results

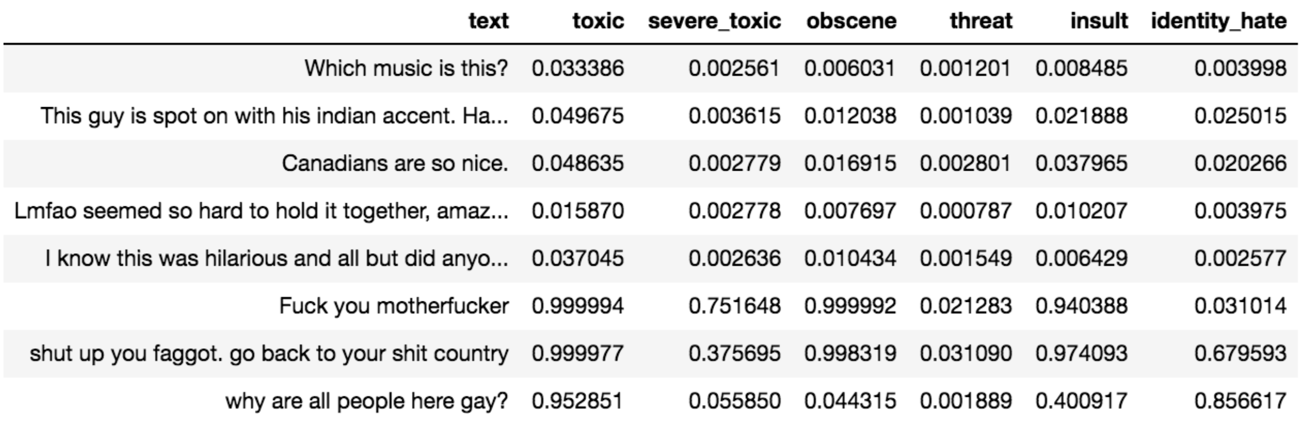
Model Evaluation and Validation

As indicated in the Implementation and Refinement section, the model with the best AUC-ROC was the *Logistic Regression model* with hyperparameters C = 1 and solver = 'sag'. C = 1 is often the best choice and solver = 'sag' came as no suprise since 'sag' is faster/better for larger datasets. Also since feature scaling wasn't an issue in our case since all features were nearly at the same scale, sag's fast convergence can be guaranteed.

The model's robustness can be clearly seen using the Kaggle scores it has achieved. These scores are calculated against unseen/holdout data and looking at these scores, we can see that the model generalizes well to unseen data.

Our Basline Logistic Regression model had an AUC-ROC score of 0.94 and a Kaggle score of 0.93. Our MultinominalNB model had an AUC-ROC score of 0.9522 and a Kaggle score of 0.9487. Our final Logistic Regression model has an AUC-ROC score of 0.983 and a Kaggle score of 0.9778 which is clearly better than both the above models.

Also to further test out the robustness of our final model, I tested the final model on some unseen comments from youtube and I am pretty satisfied with them. Here are the results :



text	toxic	severe_toxic	obscene	threat	insult	identity_hate
Which music is this?	0.033386	0.002561	0.006031	0.001201	0.008485	0.003998
This guy is spot on with his indian accent. Ha...	0.049675	0.003615	0.012038	0.001039	0.021888	0.025015
Canadians are so nice.	0.048635	0.002779	0.016915	0.002801	0.037965	0.020266
Lmfao seemed so hard to hold it together, amaz...	0.015870	0.002778	0.007697	0.000787	0.010207	0.003975
I know this was hilarious and all but did anyo...	0.037045	0.002636	0.010434	0.001549	0.006429	0.002577
Fuck you motherfucker	0.999994	0.751648	0.999992	0.021283	0.940388	0.031014
shut up you faggot. go back to your shit country	0.999977	0.375695	0.998319	0.031090	0.974093	0.679593
why are all people here gay?	0.952851	0.055850	0.044315	0.001889	0.400917	0.856617

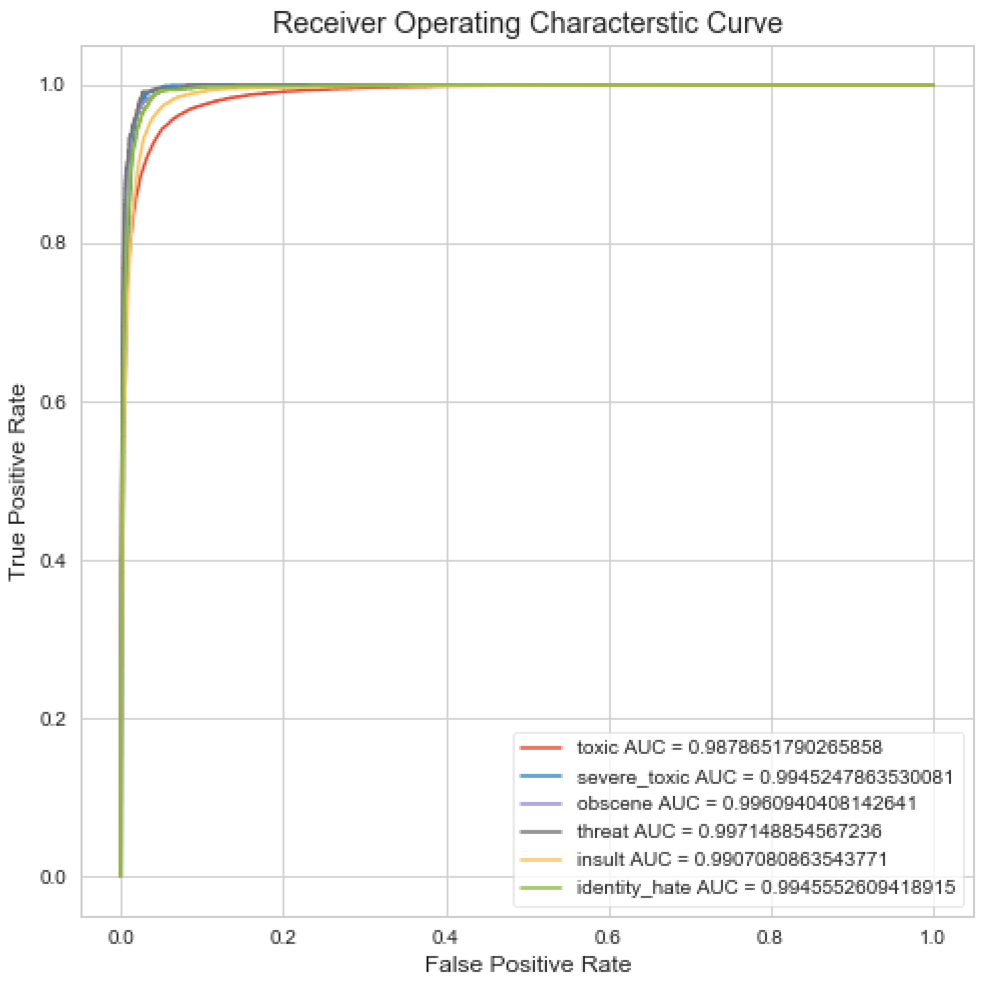
Justification

The final Logistic Regression model has an AUC-ROC score of *0.983* which is better then the benchmark model's AUC-ROC score of *0.94*. As indicated in the *Metrics* section, a higher AUC-ROC score means a better model. The Area under curve in the ROC graph for our final model is 0.983 while that of the benchmark model is 0.94. Our final model has better class separation qualities than our benchmark model.

V. Conclusion

Free-Form Visualization

The following chart displays the Receiver Operating Characterstic (ROC) curve for all the 6 toxicity classes for our final Logistic Regression model. Our final model performs pretty good for all the 6 toxicity classes despite the class imbalance problem we discussed earlier. Since the distribution of toxicity types was very uneven, I thought our model would do poorly on some classes like Threat and Severe Toxic, but it was not the case.



Reflection

The process used for this project can be summarized using the following steps :

1. An initial problem and relevant, public datasets were found.
2. The data was downloaded, examined, cleaned and converted into a vectorized representation that could be ingested by our classifiers.
3. A benchmark was created for our classifiers.
4. Classifiers were trained on our vectorized representation. The best among them was chosen.
5. The final classifer was refined using hyperparameter optimization.
6. The final tuned classifier was used to predict holdout and unseen random data.

The project aspect that I found the most difficult was step 5. Given that I had to build a multi headed model, I was not sure how to go about hyperparameter optimization. It's only after I stumbled on the OneVsRestClassifier, was I able to solve this problem.

The project aspect that I found the most interesting was the stacking of word and char features to create a new feature set that was then fed to the machine learning models. The use of char features was clever. People often try to obfuscate bad words with additional characters. Using character n-grams can potentially detect these.

Improvements

An improvement that I would like to make is to use a Neural Network and Dense Word Embeddings to do this task. An advantage of using neural networks other machine learning algorithms is that neural network can take word order into account. However, this means that the input data cannot be structured using a bag of words or tfidf model. Instead every comment text would be represented as a vector of a fixed length: these vector representations are also called sequences.

Another improvement that could be done is instead of using unigrams, we can use bigrams in our vectorized representations. This will increase our feature space but will also help our models to further distinguish between different toxicity types.

Yet another improvement that can be made is using lemmatization instead of stemming. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.
