# **Unintended Bias in Toxicity Classification**

# 

# Capstone 2 Project for Data science Career Track

# 

Natural Language Processing is a complex field which is hypothesised to be part of AI-complete set of problems, implying that the difficulty of these computational problems is equivalent to that of solving the central artificial intelligence problem of making computers as intelligent as people. With over 90% of data ever generated being produced in the last 2 years and with a great proportion being human generated unstructured text there is an ever increasing need to advance the field of Natural Language Processing.

Recent UK Government proposal to have measures to regulate social media companies over harmful content, including "substantial" fines and the ability to block services that do not stick to the rules is an example of the regulamentary need to better manage the content that is being generated by users.

Other initiatives like ​Riot Games​' work aimed to predict and reform toxic player behaviour during games is another example of this effort to understand the content being generated by users and moderate toxic content.

However, as highlighted by the Kaggle competition ​Jigsaw unintended bias in toxicity classification​, existing models suffer from unintended bias where models might predict high likelihood of toxicity for content containing certain words (e.g. "gay") even when those comments were not actually toxic (such as "I am a gay woman"), leaving machine only classification models still sub-standard.

The outcome of our analysis is the type of algorithm that companies will use to define what is free speech and what shouldn't be tolerated in a discussion. This challenge actually starts with how the training dataset was produced: Multiple people (annotators) read thousands of comments and defined if those comments were offensive or not. Where is the trick? They disagreed in many of them. Having tools that are able to flag up toxic content without suffering from unintended bias is of paramount importance to preserve Internet's fairness and freedom of speech.

## **Dataset**

At the end of 2017 the Civil Comments platform shut down and chose make their ~2m public comments from their platform available in a lasting open archive so that researchers could understand and improve civility in online conversations for years to come. Jigsaw sponsored this effort and extended annotation of this data by human raters for various toxic conversational attributes.

In the data supplied for this competition, the text of the individual comment is found in the comment\_text column. Each comment in Train has a toxicity label (**target**), and models should predict the **target toxicity** for the Test data. This attribute (and all others) are fractional values which represent the fraction of human raters who believed the attribute applied to the given comment.

For evaluation, test set examples with target >= 0.5 will be considered to be in the positive class (toxic).

The data also has several additional toxicity subtype attributes. Models do not need to predict these attributes for the competition, they are included as an additional avenue for research. Subtype attributes are:

* Severe\_toxicity
* Obscene
* Threat
* Insult
* Identity\_attack
* sexual\_explicit

## 

## **Exploratory Data Analysis**

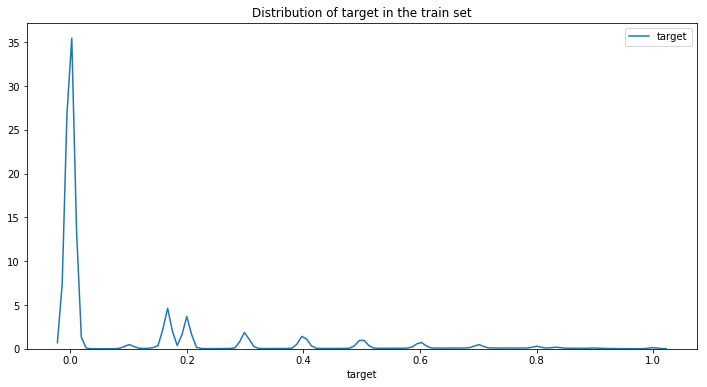
On available dataset basic exploratory data analysis was done on missing value and looking for outliers or false data, where we observed column with missing value had most of column missing thus amputing those value was not a logical option. Based on that observation it was decided to see distribution of those column and their relation with target so that based on those observation we can create some feature. After looking at all distribution and relation it was concluded that all these features were not proving almost no information, nor any significant relationships were observed. Thus comment column was observed in depth to observe any relation , new features were created so that any useful feature can be extracted from the comment column and finally it was concluded that only comment column will be used for prediction and all created features or other available features will be dropped.

### **Missing value observation**

After looking at the available data it was observed that most features based on race, gender, ethnicity or religion were missing and many of the available features without missing values like religion or feature describing emotion were filled with value as zero thus based on observation it was decided that more analysis needs to be done on all features, by combining them as a category of features.

### **Target variable Distribution**

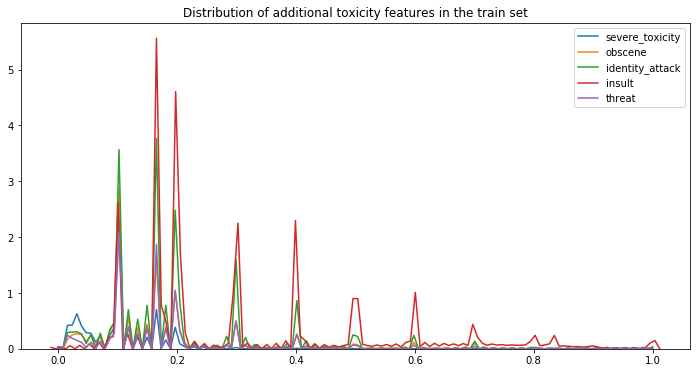
As we are predicting target variable it is important to understand the distribution of the target variable.For instance here we are going to convert the target variable to a category with target value above 0.5 as toxic otherwise as nono toxic, any information regarding uneven distribution as well as distribution near decision boundary i.e here values near 0.5 are important to note as if most observations are near 0.5 than more in-depth analysis needs to be done on those comments.



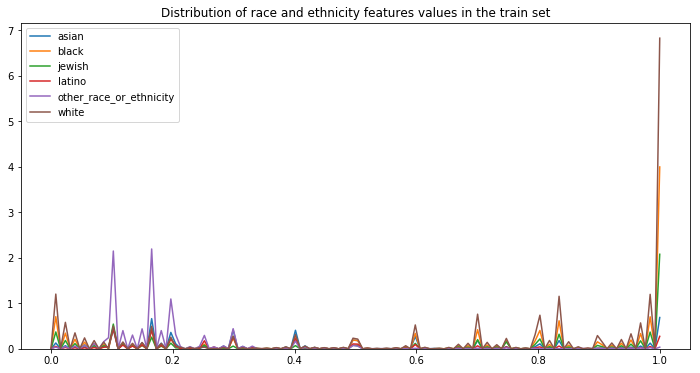
After looking at distribution it is clear that most comments are non toxic and also we have not big chunk of data neat decision boundary thus we can move ahead with 0.5 as our decision boundary.

### **Feature Distribution:**

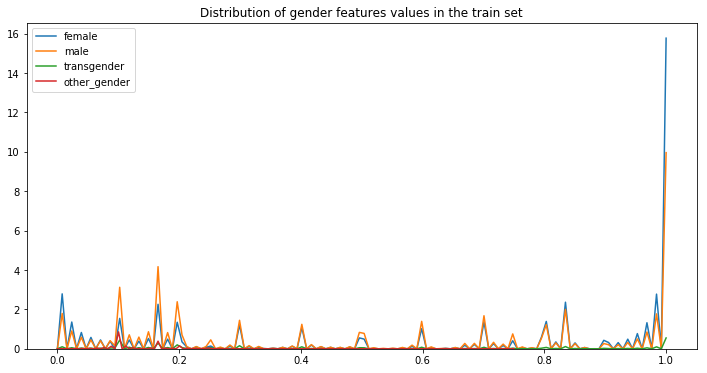
After looking at missing value we had decided to observe features after combining them as a category with respect to target variable so that new features can be created, thus we plotted distribution of these features as a group based on race, gender, sexual orientation, religion and disability with respect to target variable.



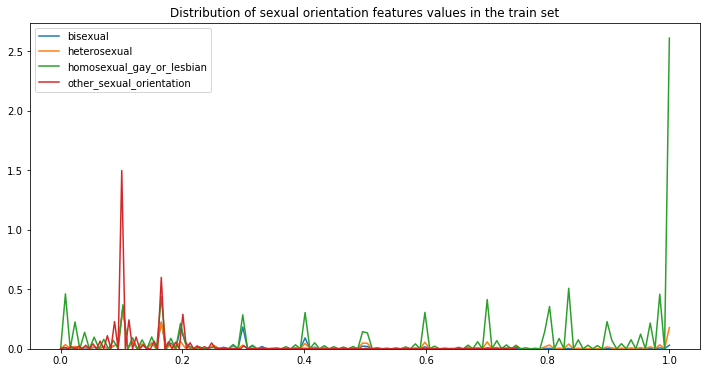
After looking at toxicity feature distribution it was clear that **type of comment does not have any direct relation with toxicity**, as we can see all types of comments can have toxic as well as non-toxic comments, reason of insult type having higher frequency in graph is more number of observations and those comments can be toxic as well as non toxic thus it was clear that using toxicity feature will not have any significant impact on the results



After looking at ethnicity based comments we did find some observation where other races or ethnicity category of people comments were non-toxic and white and black race had many toxic comments compared to non toxic. Based on that observation, we decided to create 3 feature one indicating others , one indicating white and black and third category as rest. After performing test on those features no significant results were observed and it was also important to note that the number of comments were pretty thin with data having information about these categories thus we decided to drop the columns

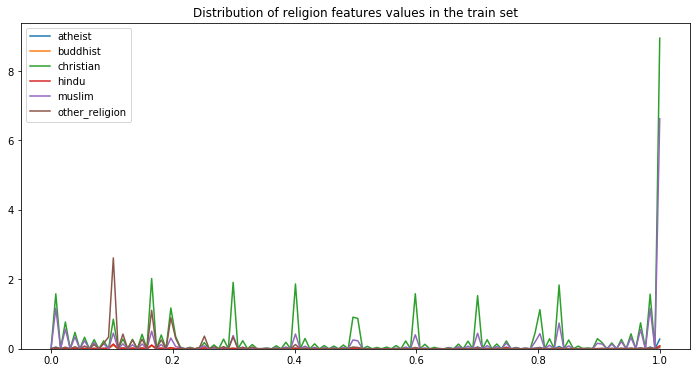


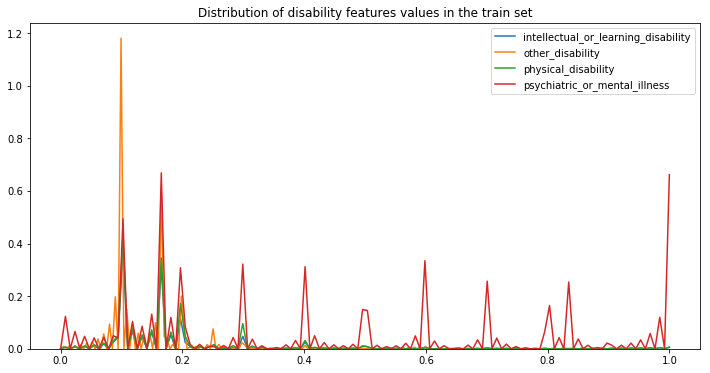
Here we did not have enough observation but based on the available data comments towards female had more probably being toxic compared to non-toxic



Homosexual commets were mostly toxic and other sexual orentation commetes were almost always non toixc but with only 1% of data were labelled based on Sexual orentation we decided not to use them for prediction.

It was also important to keep checking these features in future models as based on data observed it was very clear that it would had big impact on prediction.



Comments with christian and muslim has more probably being toxic than other religions, where as comments with hindus and buddist or other\_religion are usually non toxic which indicated to use these features but even after combining these features we did not have enough number of observations to make a firm decision to use them thus we ended up deciding to drop these columns

Here comments targeting mentally ill people has more chance to be toxic but similar to sexual orientation type we did not have enough observation thus only option is to keep these features as to be looked at in future iteration of modelling.

**OBSERVATION**

Overall after looking at above distribution plots we did found some patterns which can be used,but those patterns did not had many significant number of observations.

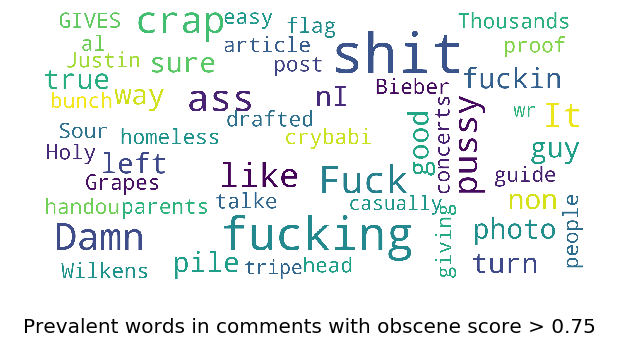
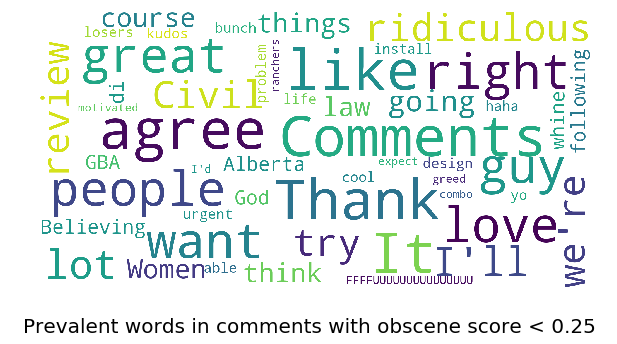
The features which we could have used would be any based on combining bunch of features making one category of comments i.e all comments mentioning religion are non-toxic or toxic. As after combing all similar features would had made significant number of observations to create its own category.

After looking at all possible ways to make available feature useful we decided to drop all features and redirect our attention to observe comments column and try to create feature out of them, thus in the following section we will first observe words based on toxicity features and their relation and later we try to create numerical feature out of comments column.

### 

### **Word Cloud Plots**

It is understood that different types of comments will have different type of words thus we decided to firstly make sure it holds true,we also need to find are there any patterns that we can identify with respect to words.

To understand these patterns we have plotted most frequently occurring words in comments based on different types of toxicity with target higher than 0.75 and less than 0.25. Below is an example for obscene type of comments

After looking at all the plots it can be clearly observed that we do have some light pattern like **comments having abusive words are almost all toxic** and comments with **words like Love, civil are almost always non-toxic**.It was also observed that there are certain words like **People, trump can be toxic as well as non-toxic**. Based on observation using bag of words will help us create these features thus we will use bag of words in our model.

It was also important to note that we did not found any patterns with respect to type of words for example words with Contractions or words with capital letters being toxic or non-toxic. But to clearly observe that we will create features from comments column.

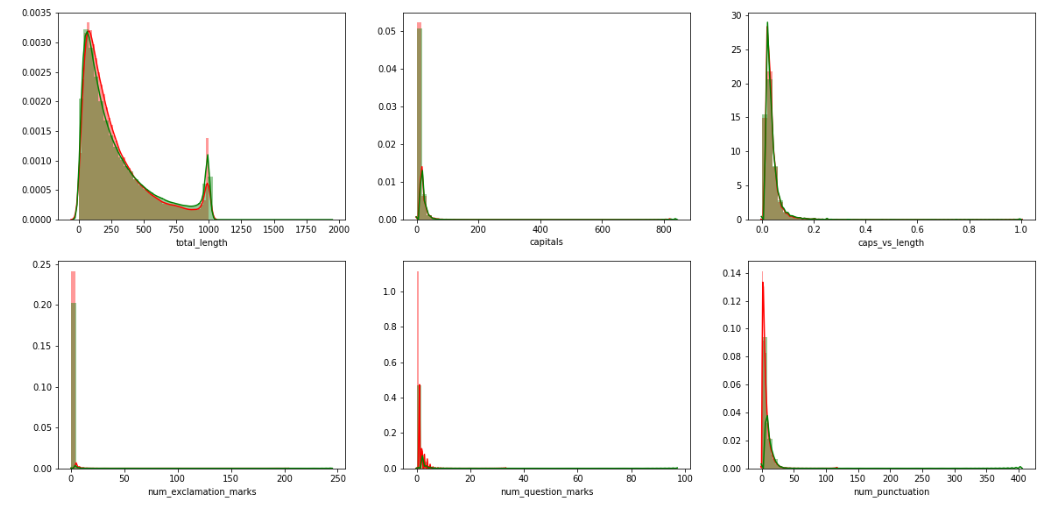
### **Feature Engineering**

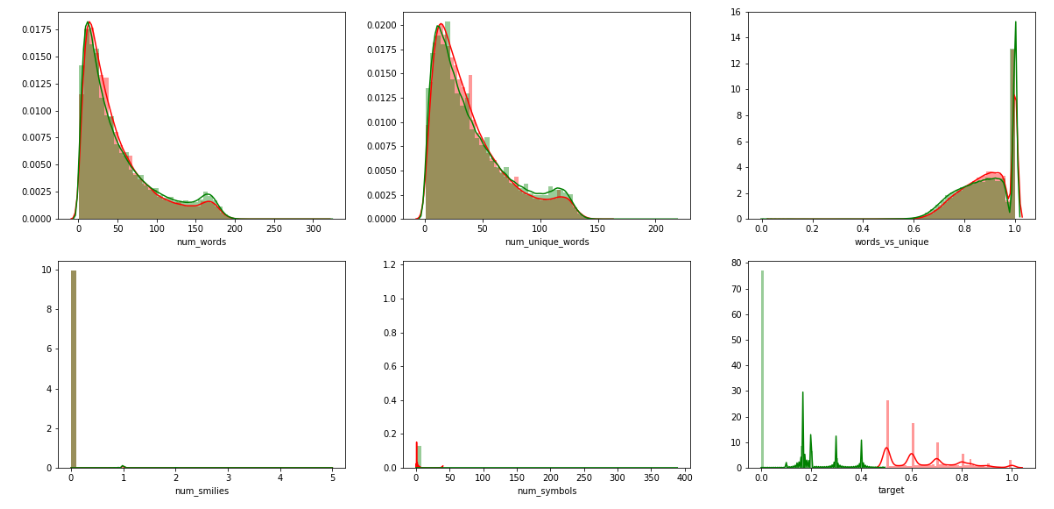
### After looking at word cloud of comments we were not able to find any clear pattern thus we tried to create new features based on comments. In this section we tried to create features like Comment\_Length, number of capital words , capital vs length of comments, number of exclamation marks, number of question marks, number of punctuation , number of symbols , number of unique words and number of smiles (standard smiles)

##### **Distribution of comment data from features created**

It is always good to plot new feature to understand their distribution and their relation,

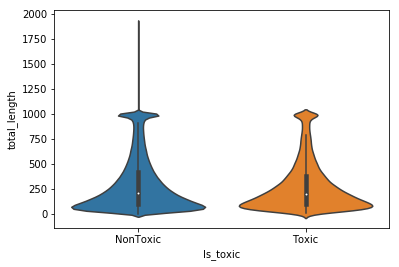
It can also help us to find any abnormality in the comment column. Thus we have plotted decided to observe them.





#### After looking at the plot we have found some redundant features like number of unique words which can be clearly observed as number of words and the number of unique words have very similar distribution.

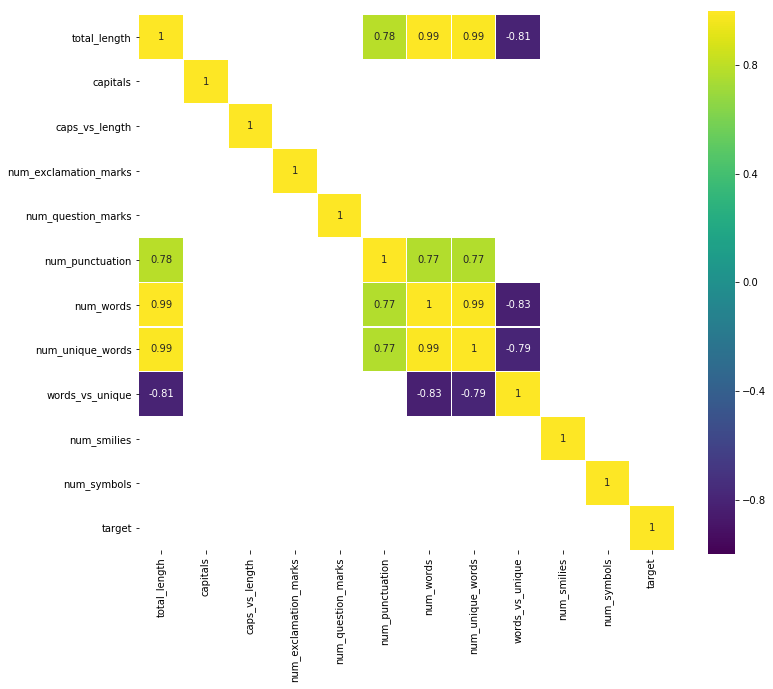
#### 



When looking at total length distribution plot we can see a binomial distribution as we can see one peak on around 1000 and and another near 100. It is also observed that data is skewed on lower length. Based on the above observation we can only say that most comments are sort but if they are longer than a certain threshold they are likely to be very long.

### **CoRelation**

It is always a good idea to see relation between features and target

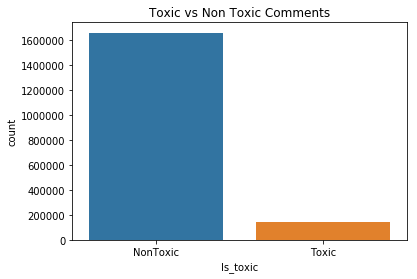


#### Above plot highlight correlation more than 60 percent which can be positively or negatively correlated.Based on plot no new feature have high correlation with the target variable.

#### Also our above observation of having similar distribution for number of words and the number of unique words is justified here as we can clearly see correlation being 99%, meaning we will have to drop one of these features.It is also noteworthy that Total length as we know is basically the same as the number of words thus we have to one of these columns as well drop the column. Now looking at other column we can see number of punctuation and number of words are correlated indicating longer comments have more punctuation.

#### Overall when looking at prediction point of view and looking at distribution and available value it was pretty clear that no information can be obtained from new features. As most features have either same value most of the time irrespective or comments being toxic or nontoxic and also most comments have value 0.

**Number of comments**



There is clearly an imbalance in data with most of the comments are not toxic. Based on that we had decided to remove some non-toxic comments to balance data, we did this by selecting around top half a million rows.which also helped to create word and TF IDF vectors of a size our RAM could have handled.

## **OBSERVATION AND ASSUMPTION**

* Based on the Above analysis all newly created column and existing supporting column does not have correlation with target
* There is very uneven distribution of toxic and non-toxic comments
* creating word or sentence vector on above dataset will be very time consuming on available processing unit machine
* non-toxic comments will be filtered out to create 30-70 ratio of toxic and non-toxic comments, concerning available resources and prioritising the importance of understanding and implementation of NLP Concept

## **Text Data Cleaning**

Text cleaning will be performed mainly in 5 steps, which will be

* Lower casing
* Expanding Contractions
* Removing Special Characters
* Removing Stopwords

#### **Lower Case**

Here all alphabet will be converted to lowercase as all available mapping are in lower case alphabet as well as it will remove inconsistent typing errors and make standard text words and sentences.

#### **Removing Special Characters**

Special characters and symbols are usually non-alphanumeric characters or even occasionally numeric characters (depending on the problem), which add to the extra noise in unstructured text. Usually, simple regular expressions (regexes) can be used to remove them.

#### **Removing Stopwords**

Words which have little or no significance, especially when constructing meaningful features from text, are known as stopwords or stop words. These are usually words that end up having the maximum frequency if you do a simple term or word frequency in a corpus. Typically, these can be articles, conjunctions, prepositions and so on. Some examples of stopwords are a, an, the, and the like.

#### **Lemmatization**

Lemmatization is very similar to stemming, where we remove word affixes to get to the base form of a word. However, the base form in this case is known as the root word, but not the root stem. The difference being that the root word is always a lexicographically correct word (present in the dictionary), but the root stem may not be so. Thus, root word, also known as the lemma, will always be present in the dictionary. Both nltk and spacy have excellent lemmatizers. We will be using spacy here.

#### **Expanding Contractions**

Contractions are shortened version of words or syllables. They often exist in either written or spoken forms in the English language. These shortened versions or contractions of words are created by removing specific letters and sounds. In case of English contractions, they are often created by removing one of the vowels from the word. Examples would be, do not to don’t and I would to I’d. Converting each contraction to its expanded, original form helps with text standardization

## **Train test split**

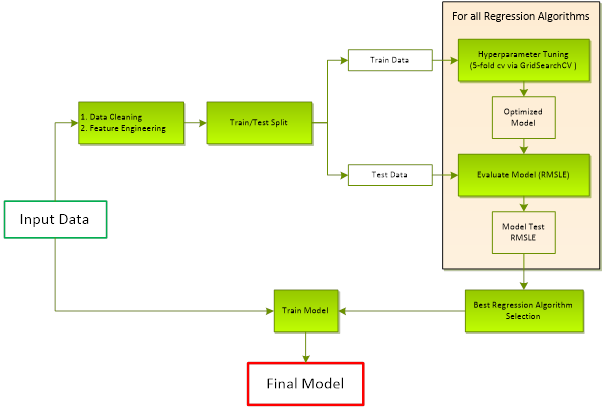


Figure depicts the overall procedure followed to obtain the Final Model. The provided data is first cleaned and transformed using Feature Engineering. We then split the data into Train set (for Hyperparameter tuning) and Test set (for Model Evaluation). Using Accuracy our evaluation metric, we compare various models and select the regression algorithm based on best accuracy score on the Test data. The final model used for submission is then obtained by again training the selected Regression Algorithm on the entire Input Data set.

## **Evaluation Metric**

The data consists of highly imbalanced data where most of comment we have are non-Toxic comments.

**Accuracy** is the most intuitive performance measure and it is simply a ratio of correctly predicted observations.

Accuracy=(Number of comment correctly identified as toxic )/(Total number of comments)

**Precision** is the ratio of correctly predicted positive observations to the total predicted positive observations. Precision for a genre is given by

Precision(comment=Toxic) = (Number of comments‘ correctly’ identified as Toxic )/(Total number of comments that have been identified as toxic)

**Recall** is the ratio of correctly predicted positive observations to the all observations in actual class. Recall for a genre is given by

Recall(Comment=Toxic) = (Number of Comments correctly identified as Toxic)/(Total number of Toxic comment in the data set)

**F1 Score** is the Harmonic mean of Precision and Recall.

F1 Score = 2\*(Recall \* Precision) / (Recall + Precision)

Below is the method we use to come up with a single F1 score for our model performance. For each genre, compute Precision, Recall and F1 Score

Compute a weighted average of the F1 score – weighted by the support. This is used as our final metric

## **Algorithm Details**

Text based classification can be broadly summarized into the below 2 blocks

* Text Encoder – encodes text data into numeric vectors
* Classification Models – ML models used to make predictions for a single class

### **Text Encoder**

We cannot work with text directly when using machine learning algorithms. Instead, we need to convert the text to numbers. Text encoding is a process wherein the text data is converted into unique numeric vectors which can be consumed by ML models. Below are few text **encoders** that were considered for this project

**Count Vectorizer**

Word Count with **Count Vectorizer** (Bag of Words Model): The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document.

**TF-IDF Vectorizer**

Word Frequencies with **TF-IDF Vectorizer** (Bag of Words Model): In the standard CountVectorizer model above, we used just the term frequency in a document of words in our vocabulary. In TF-IDF, we weight this term frequency by the inverse of its popularity in all documents. For example, if the word "movie" showed up in all the documents, it would not have much predictive value. It could actually be considered a stop word. TF-IDF is obtained by down weighing its counts by 1 divided by its overall frequency.

### **word embedding**

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Conceptually it involves a mathematical embedding from a space with many dimensions per word to a continuous vector space with a much lower dimension.

Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix,probabilistic models, explainable knowledge base method, and explicit representation in terms of the context in which words appear

Software for training and using word embeddings includes Tomas Mikolov's Word2vec, Stanford University's GloVe, AllenNLP's Elmo, fastText, Gensim, Indra and Deeplearning4j.

We have used Glove and FastText here

#### **Glove**

GloVe, coined from Global Vectors, is a model for distributed word representation. The model is an [unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning) algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity. Training is performed on aggregated global word-word [co-occurrence](https://en.wikipedia.org/wiki/Co-occurrence) [statistics](https://en.wikipedia.org/wiki/Statistics) from a corpus, and the resulting representations showcase interesting linear substructures of the [word vector space](https://en.wikipedia.org/wiki/Word_vector_space). It is developed as an [open-source](https://en.wikipedia.org/wiki/Open-source_software) project at Stanford. As log-bilinear regression model for unsupervised learning of word representations, it combines the features of two model families, namely the global matrix factorization and local context window methods.

#### **FastText**

fastText is a library for learning of [word embeddings](https://en.wikipedia.org/wiki/Word_embedding) and text classification created by [Facebook](https://en.wikipedia.org/wiki/Facebook)'s AI Research (FAIR) lab. The model allows to create an [unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning) or supervised learning algorithm for obtaining vector representations for words. Facebook makes available pretrained models for 294 languages.fastText uses a [neural network](https://en.wikipedia.org/wiki/Artificial_neural_network) for word embedding.

## **Models used**

Following machine learning algorithm were used

* LogisticRegression
* DummyClassifier
* Bernoulli Naive Bayes
* Multinomial Naive Bayes
* SVM Classifier with SVC
* RandomForestClassifier
* Keras Sequential model with GloVe Embedding,Flatten and Dense layers
* Keras Sequential model with Fast Text Embedding,Flatten and Dense layers

### **Approach**

After performing standard text data cleaning operation machine learning algorithm were applied with mainly two approaches , firstly word vector and TF-IDF Vectors were created and these word vectors were used to perform hyperparameter tuning is done with RandomsearchCV. With repetitive process by looking at best parameter and score on each hyperparameter combination. Once parameters are finalized final model is build on which Accuracy and F1 SCore were calculated for test and train combination.

For word embedding process firstly top 90,000 words were selected and data were transferred in integer sequence and later were embedded with word vectors like Glove and Fast Text which were fed to our neural network.which are evaluated based in binary\_crossentropy loss finally loss and accuracy are plotted

### **LogisticRegression with count Vectors**

For Logistic Regression, data were first converted to word vectors then it was tuned with hyperparameter using the Train Set using RandomizedSearchCV with 5-fold cross validation. Below table summarizes the optimal hyperparameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Count Vectorizer** | Ngram = (1, 2) | min\_df =0.01 | analyzer="word" |
| **Logistic Regression** | C=1.4 | max\_iter= 10 |  |

The **Accuracy And** **F1 score** and the model train and test score summary for the model is given in the table below

|  |  |  |
| --- | --- | --- |
| **Score** | **Test** | **Train** |
| **Accuracy** | 0.80 | 0.80 |
| **F1 score** | 0.345 |  |

**Observations:**

Unlike tree based on regression technique using logistic with Bag of word model does not increase accuracy while iterating multiple times, C=1.4 is optimum value for logistic regression, Accuracy or F1 Score are not very significant to use in our model but it is a good starting point

### 

### 

### 

### 

### 

### **Dummy Classifier with count Vectors**

We can see that the accuracy scores are not that bad. However, the precision and recall scores are just unacceptable. This is because of the imbalanced targets, as most of the targets have label 0 and few have label 1. So even a dumb classifier which always predicts the most common class would give a respectable accuracy score. So we need to compare our classifier's performance with once such most-common-class-classifier

For Dummy Classifier, data were first converted to word vectors then it was tuned with hyperparameter using the Train Set using RandomizedSearchCV with 5-fold cross validation. Below table summarizes the optimal hyperparameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Count Vectorizer** | Ngram = (1, 2) | min\_df =0.01 | analyzer="word" |
| **Dummy Classifier** | strategy="most\_frequent" |  |  |

The **Accuracy And** **F1 score** and the model train and test score summary for the model is given in the table below

|  |  |  |
| --- | --- | --- |
| **Score** | **Test** | **Train** |
| **Accuracy** | 0.77 | 0.892 |
| **F1 score** | 0.0 |  |

**Observations:**

After performing hyper parameter tuning best strategy was "most\_frequent” for Dummy Classifier and we have obtained Accuracy not significantly good and F! Score is zero means every prediction is non-Toxic which is most frequent class Therefore, we need better models. Hence we will explore other models better suited for text classification purposes viz Naive Bayes Classifier and Support Vector Machines

### **Bernoulli Naive Bayes using Count Vectors as features**

For Bernoulli Naive Bayes Classifier, data were first converted to word vectors then it was tuned with hyperparameter using the Train Set using RandomizedSearchCV with 5-fold cross validation. Below table summarizes the optimal hyperparameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Count Vectorizer** | Ngram = (1, 2) | min\_df =0.01 | analyzer="word" |
| **Bernoulli Naive Bayes** | alpha=100 |  |  |

The Accuracy And F1 score and the model train and test score summary for the model is given in the table below

|  |  |  |
| --- | --- | --- |
| **Score** | **Test** | **Train** |
| **Accuracy** | 0.80 | 0.80 |
| **F1 score** | 0.43 |  |

**Observations:**

After performing hyper parameter tuning best parameters observed is alpha=100 is which successfully obtained F1 Score much higher than logistic regression while keeping almost the same Accuracy which indicate a good balance of prediction of toxic comments than logistic regression

### 

### 

### 

### **Multinomial Naive Bayes using TF-IDF Vectors as features**

For Multinomial Naive Bayes Classifier, data were first converted to word vectors then it was tuned with hyperparameter using the Train Set using RandomizedSearchCV with 5-fold cross validation. Below table summarizes the optimal hyperparameters

|  |  |
| --- | --- |
| **TF IDFVectorizer** | Ngram = (1, 1) |
| **Bernoulli Naive Bayes** | alpha= 0.03 |

The **Accuracy And** **F1 score** and the model train and test score summary for the model is given in the table below

|  |  |  |
| --- | --- | --- |
| **Score** | **Test** | **Train** |
| **Accuracy** | 0.851 | 0.892 |
| **F1 score** | 0.574 |  |

**Observations:**

After performing hyper parameter tuning best parameters observed is alpha=0.03, which help us obtain significantly better Accuracy to 85% and F1 score went really up to 0.57 making prediction accuracy more significant

### 

### 

### 

### 

### **SVM Classifier using TF-IDF**

For SVM Classifier, data were first converted to word vectors then it was tuned with hyperparameter using the Train Set using RandomizedSearchCV with 5-fold cross validation. Below table summarizes the optimal hyperparameters

|  |  |  |  |
| --- | --- | --- | --- |
| **TFIDF** | Ngram = (1, 1) | min\_df =1 | analyzer="word" |
| **SVM Classifier** | max\_iter': 100 | kernel': 'rbf’ | C= 1.1 |

The Accuracy And F1 score and the model train and test score summary for the model is given in the table below

|  |  |  |
| --- | --- | --- |
| **Score** | Test | Train |
| **Accuracy** | 0.7829 | 0.784 |
| **F1 score** | 0.3142 |  |

**Observations:**

Based on tuning parameters best possible parameter with good balance of accuracy and balance between classifier prediction alpha=100 is optimum value obtained for SVM Classifier even though accuracy remain same F1 score went little up to logistic regression but still multinomial naive bias is best looking predicting algorithm.

### **Random Forest using Count Vectors**

For Random Forest Classifier, data were first converted to word vectors then it was tuned with hyperparameter using the Train Set using RandomizedSearchCV with 5-fold cross validation. Below table summarizes the optimal hyperparameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Countvectorizer** | Ngram = (1, 2) | min\_df =0.01 | analyzer="word" |
| **Random Forest** | max\_depth=3 | min\_samples\_split=10 | min\_samples\_leaf=7 |

The Accuracy And F1 score and the model train and test score summary for the model is given in the table below

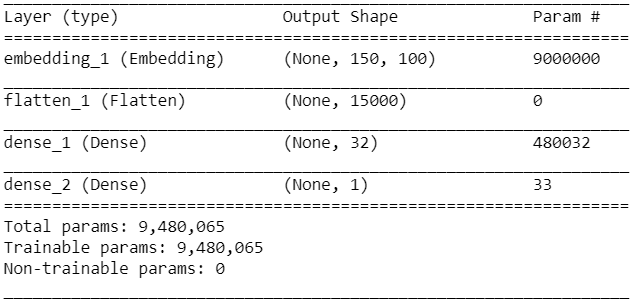
|  |  |  |
| --- | --- | --- |
| **Score** | Test | Train |
| **Accuracy** | 0.7685 | 0.769 |
| **F1 score** | 0 |  |

**Observations:**

After trying back bag of word model with random forest it we obtained a 0.76 accuracy and 0 F1 score meaning it's not a good model with predicting everything as non-toxic increasing no of trees and and no of iteration can increase accuracy but based on analysis time and resource consumed, multinomial naive bias looks way better option.

### **GloVe Word Embedding with Neural network**

Here first words were cleaned and parameters were initialized and then following architecture was created



Here first layer is basically embedding layer where each sentence with (100 words with 150 vectors each) are fedded then was flattened which has 15000 nodes(100(word)\*150(vec)) in next phase for hidden dense layer which has 32 nodes and 480032 (15000 \* 32 + 32) and with activation function as relu which later was fed to our final prediction node with 33 parameters(32\*1+1) with sigmoid function to convert to binary output

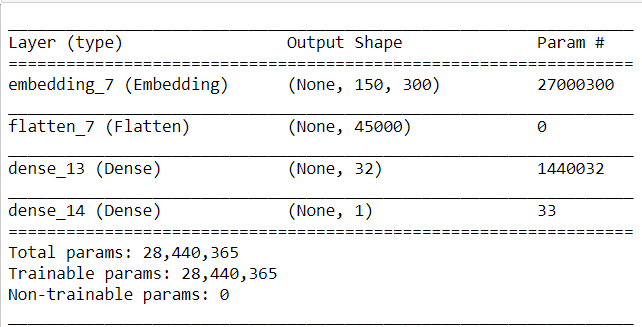
After training above model with 20 epch with train and test data following observation were obtained

|  |  |  |  |
| --- | --- | --- | --- |
|  | Train | Test | Validate |
| Accuracy | 0.939 | 0.897 | 0.898 |
| F1-Score | 0.863 | 0.762 | 0.765 |

Based on trying different architecture conceding accuracy and execution time significantly high accuracy and F1 score just by adding one hidden layer with 32 nodes obtaining 89% accuracy and 0.76 F1-Score by using GloVe Pre-Trained word vector

### **Fast Text Word Embedding with Neural network**

Here first words were cleaned and parameters were initialized and then following architecture was created



Here first layer is basically embedding layer where each sentence with (100 words with 300 vectors each) are fedded then was flattened which has 15000 nodes(100(word)\*150(vec)) in next phase for hidden dense layer which has 32 nodes and 480032 (15000 \* 32 + 32) and with activation function as relu which later was fed to our final prediction node with 33 parameter(32\*1+1) with sigmoid function to convert to binary output

After training above model with 20 epch with train and test data following observation were obtained

|  |  |  |  |
| --- | --- | --- | --- |
|  | Train | Test | Validate |
| Accuracy | 0.976 | 0.894 | 0.893 |
| F1-Score | 0.949 | 0.769 | 0.765 |

Based on trying different architecture conceding accuracy and execution time significantly high accuracy and F1 score just by adding one hidden layer with 32 nodes obtaining 87% accuracy and 0.769F1-Score by using GloVe Pre-Trained word vector

**Observation**

After performing many classification using count word vectors, TF-IDF vectors and using predefined word embeddings with Sklearn and neural network library certain conclusions and observation can be made

## 

* Neural network does perform much better than count or TF-IDF vectors
* TF-IDF vectors are better than Count word vectors
* Most algorithm using count vector try to predict same result as non-toxic comment
* GloveWord and Fast Text has much better prediction accuracy with F1-score indicating most toxic comments are identified correctly also keeping non-toxic separate

## **Summary**

### **Data Exploration Conclusions**

Data exploration was performed in two steps one was features other than comments column and second section was based on features created from comments column. Based on overall EDA some important points observed were

* We do have features based on Sexual orientation,Religion and Disability which can help predict comment being toxic or not but we need more number of observations.
* We do have words which can be either toxic or non toxic thus using bag or words and TF-IDF would be a good idea
* There is no information in features created from Comments like number of capital letters or number of punctuation and many more.

### **Modeling Conclusions**

Based on the available data and using different vectorising approach following are significant conclusion can be made

* The best predicting model uses Glove Embedded vectors and FastText Embedded vectors with simple neural network to make classification and both achieved an overall F1 score of 0.77. It is noteworthy observation that execution time of Glove was better simply because we used Glove Vector with 100D where are we used 300D for fast vector.
* Models created from TF-IDF were much better than model from count vector as almost all models with count vectores were basically classifying all results as non-toxic comments irrespective of number of iteration or other hyper parameter
* Even though TF-IDF better result than count vector they fell far behind with respect to accuracy and F1 Score obtained by neural network.

### **Limitations and Scope for Model Improvements and future work**

Following points are indication of limitation as well as many ways to improve current model and it use cases

* We had very imbalance dataset gathering more toxic comments would help improve our model.
* Due to limiting processing prower we had removed certain nontoxic comments
* TF-IDF vectors were very large in dimension by performing dimensionality reduction on those vectors will improve our execution process and time of execution.
* Current model with word embedding are built using simple one hidden layer of neural network , this can definitely be improved by adding more layers like Convolutional and pooling which can help to extract more feature
* Web app can be created where users can enter text comments and predicted toxicity can be displayed
* API can be created for user so that toxic comment can be classified and flagged as soon as they are entered which can help websites or application block or restrict unwanted content.