ML Mini Project

Project: Why So Harsh?

Venkat Suprabath Bitra (IMT2019091)

Vignesh Bondugula (IMT2019092)

Venkat (IMT2019091)
Vignesh (IMT2019092)

Table of Contents

Venkat (IMT2019091)
Vignesh (IMT2019092)

Introduction

The dataset given is about classifying the comments in a blog website (UpBlog) and judge them individually into 6 mutually independent classes of being harsh, namely:

1. Harsh
2. Extremely Harsh
3. Vulgar
4. Threatening
5. Disrespect
6. Targeted Hate

In the entire dataset the number of comments for each class was observed to be as follows:

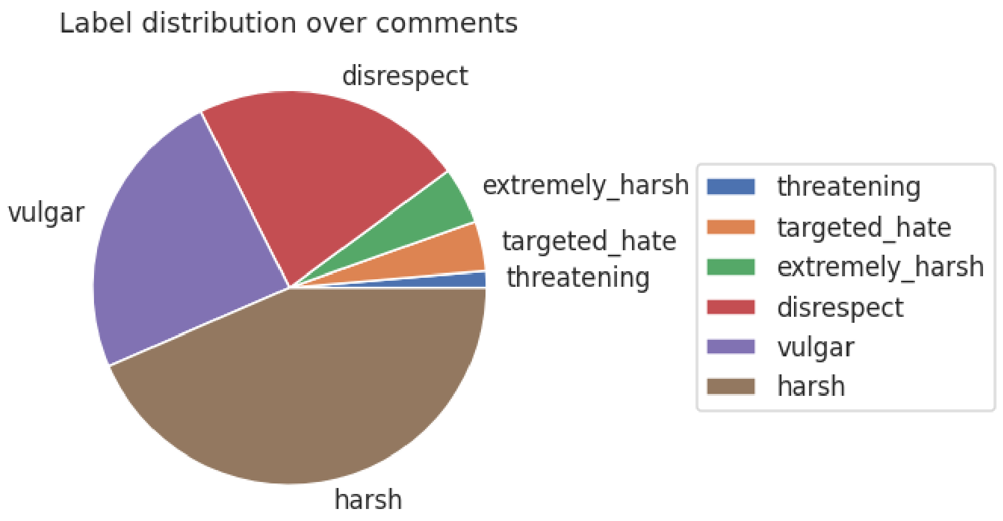
Column Name	**Number of Comments of the Class** (out of 127656 in train data)
Harsh	12227
Extremely Harsh	1310
Vulgar	6772
Threatening	383
Disrespect	6278
Targeted Hate	1147

We can observe from the above table that the given data is highly unbalanced. This will result in us overfitting and obtain a lot of false negatives. We can try to reduce the effects using class balancing features provided by scikit-learn. This also means we will have high accuracy scores as well as ROC-AUC scores.

Further Data Analysis has been shown in the EDA section.

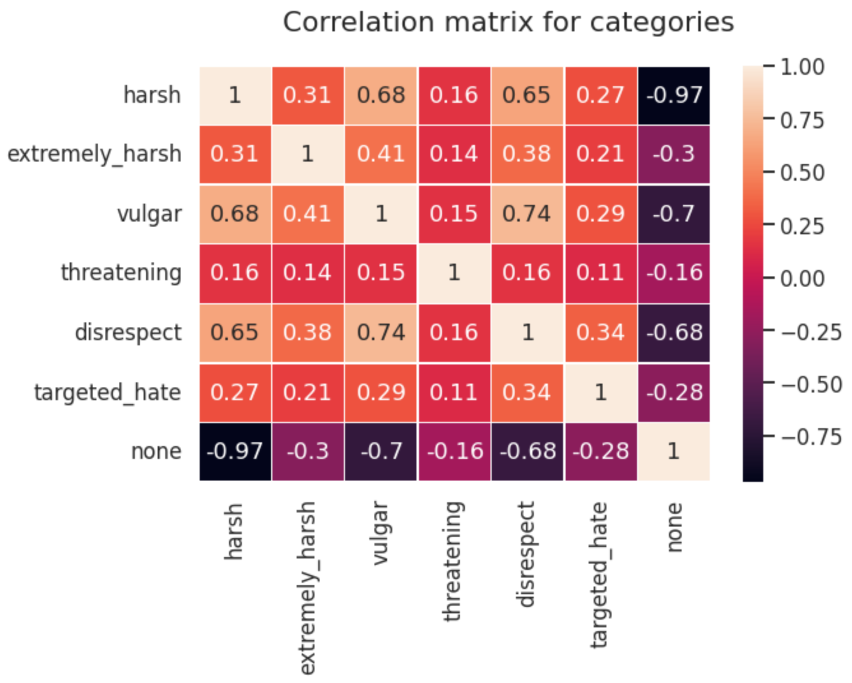
Exploratory Data Analysis

The distribution of occurrences of each label in the given dataset is as given below in the pie chart.



Venkat (IMT2019091)
Vignesh (IMT2019092)

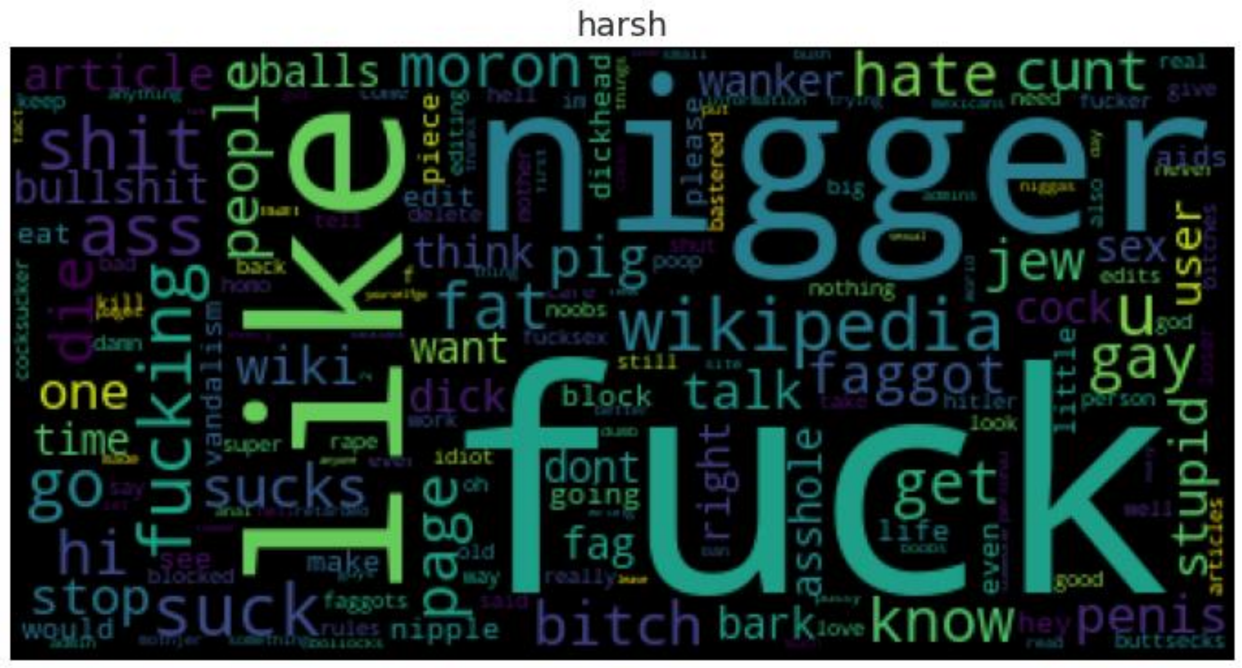
The correlation matrix of the labels in the given dataset is as follows:



Correlation matrix for categories

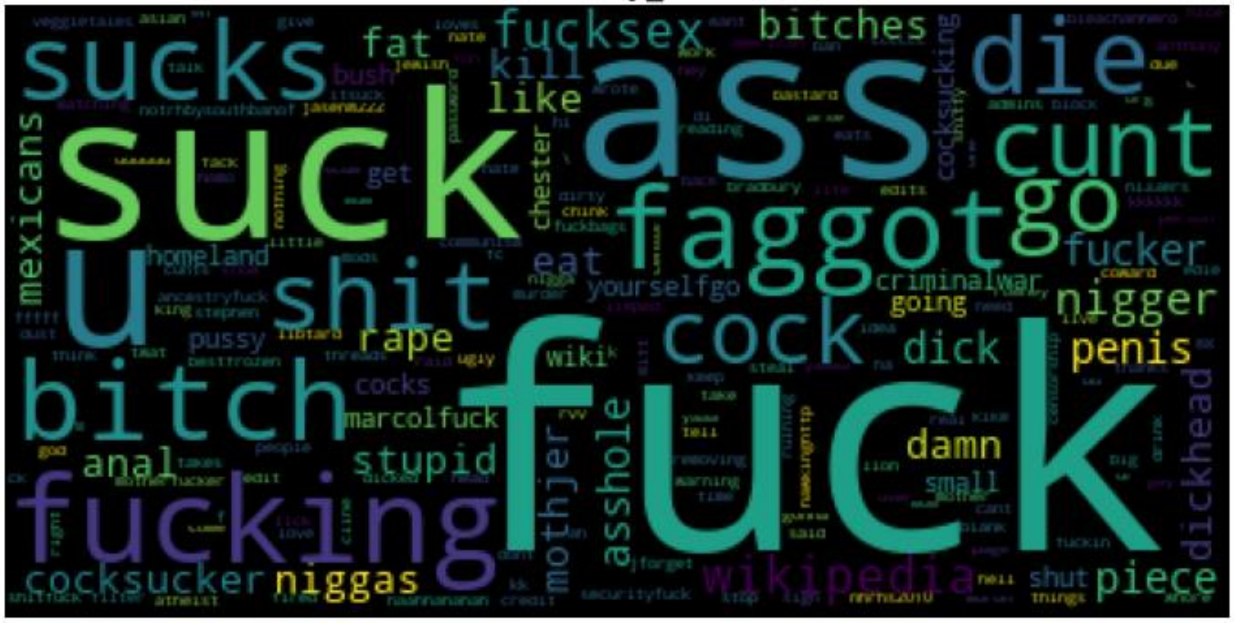
	harsh	extremely_harsh	vulgar	threatening	disrespect	targeted_hate	none
harsh	1	0.31	0.68	0.16	0.65	0.27	-0.97
extremely_harsh	0.31	1	0.41	0.14	0.38	0.21	-0.3
vulgar	0.68	0.41	1	0.15	0.74	0.29	-0.7
threatening	0.16	0.14	0.15	1	0.16	0.11	-0.16
disrespect	0.65	0.38	0.74	0.16	1	0.34	-0.68
targeted_hate	0.27	0.21	0.29	0.11	0.34	1	-0.28
none	-0.97	-0.3	-0.7	-0.16	-0.68	-0.28	1

The Word Cloud for different columns to assist in Bag of Words is shown below:

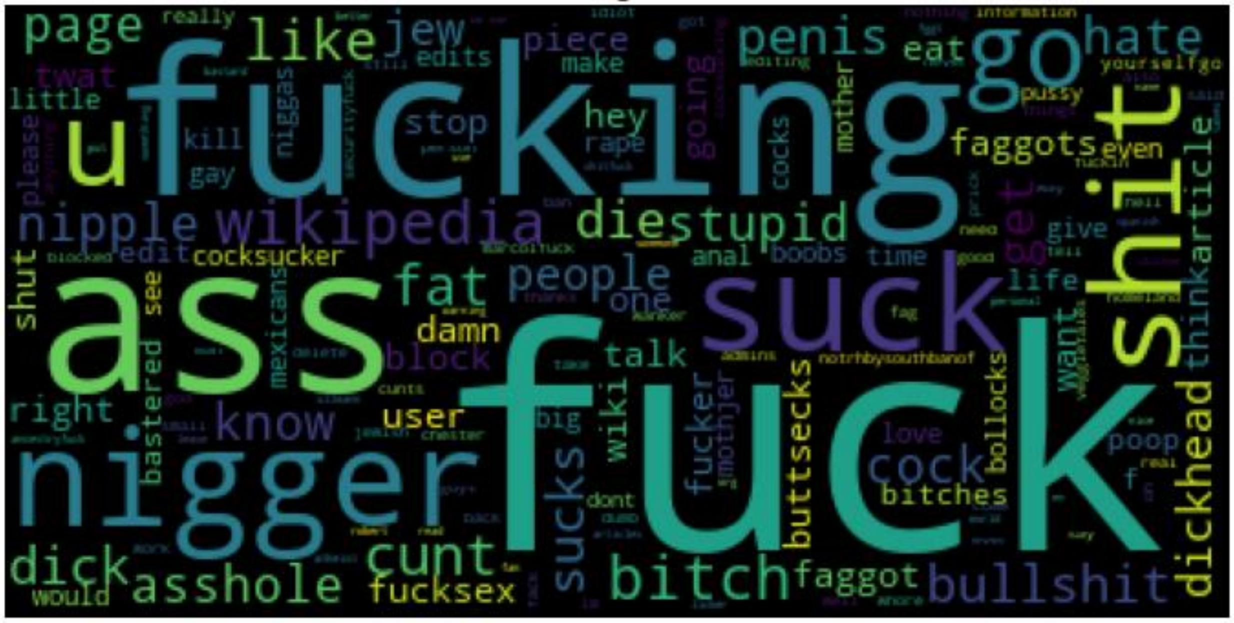


Venkat (IMT2019091)
Vignesh (IMT2019092)

extremely_harsh

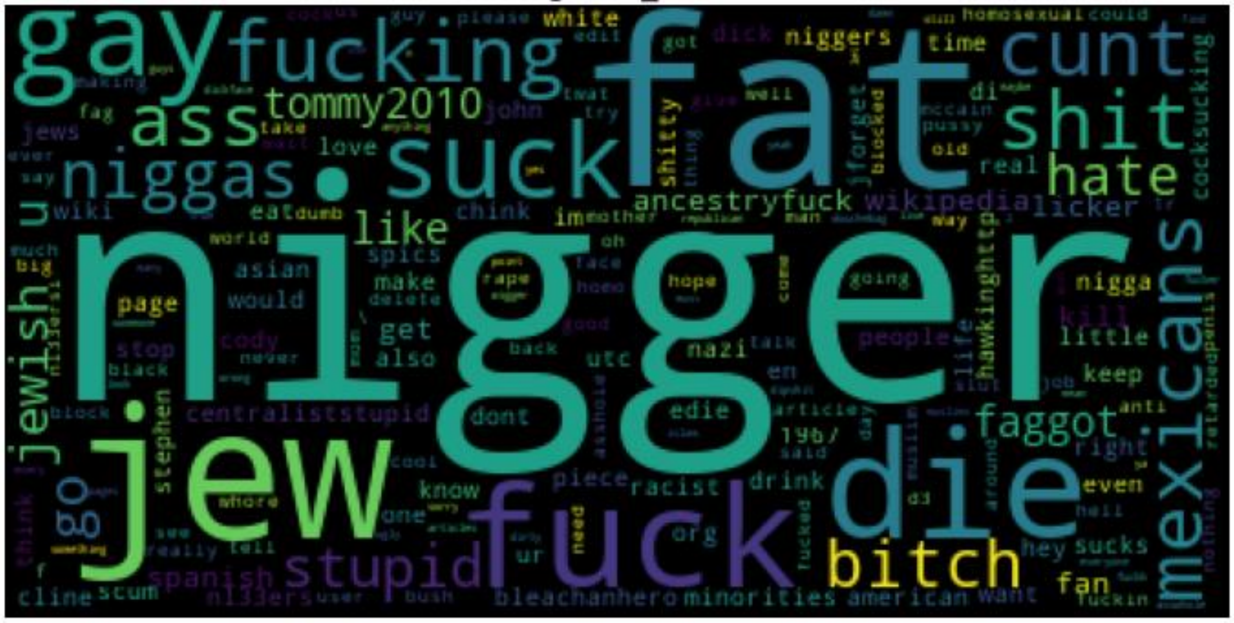


vulgar

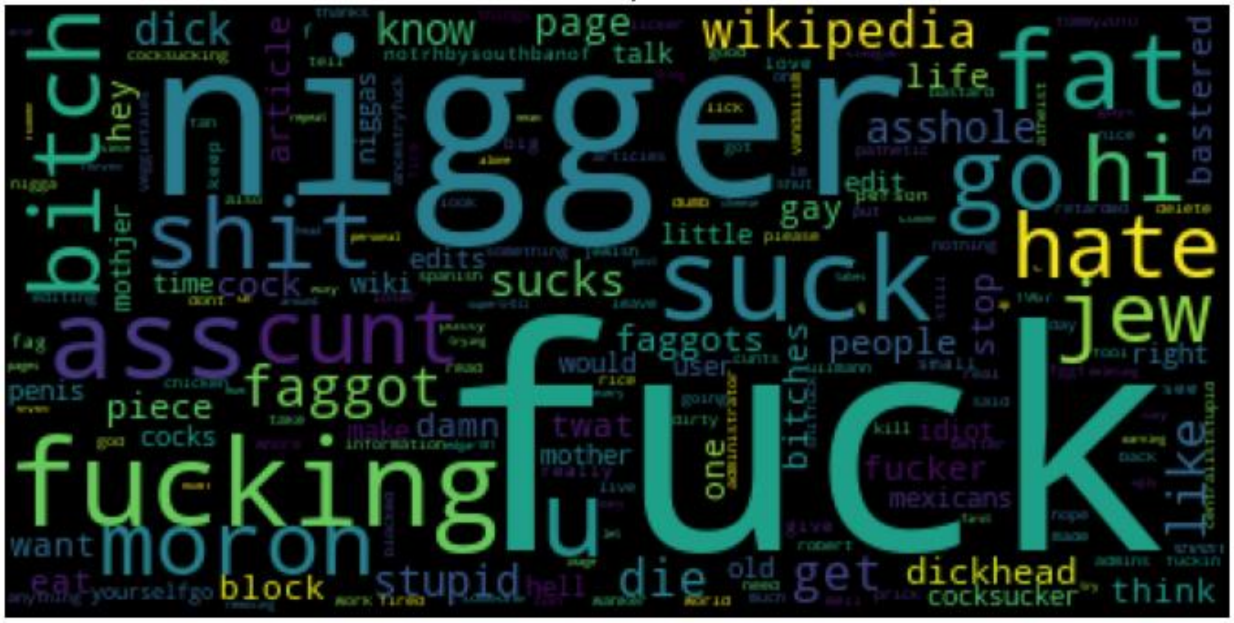


Venkat (IMT2019091)
Vignesh (IMT2019092)

targeted_hate

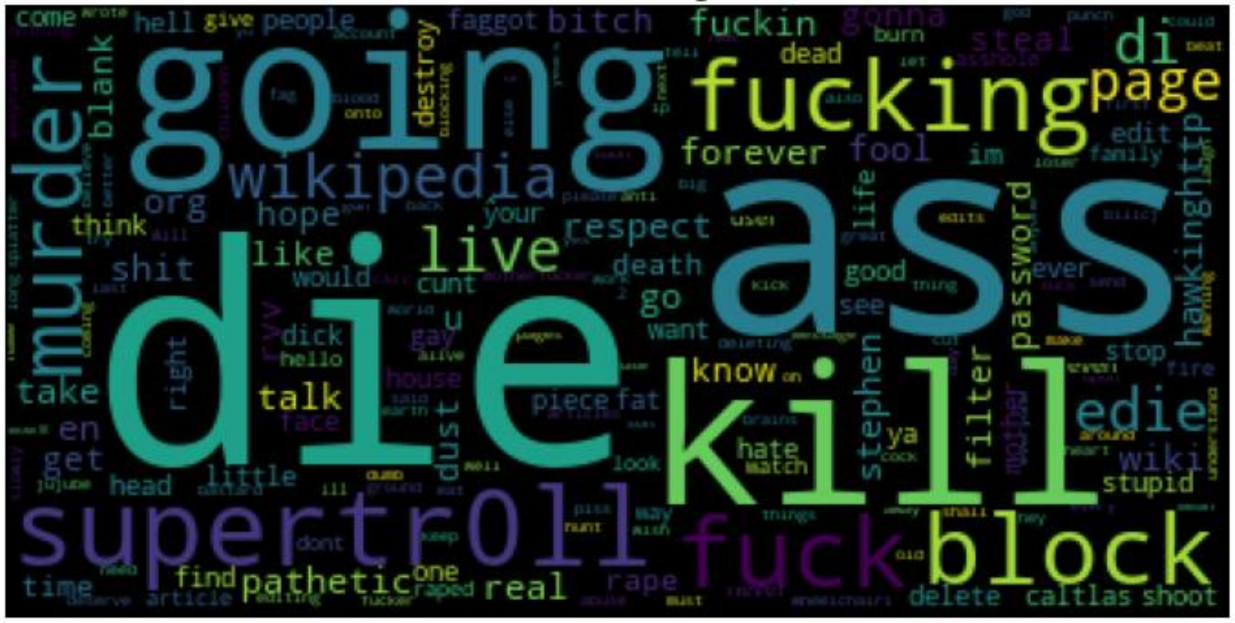


disrespect

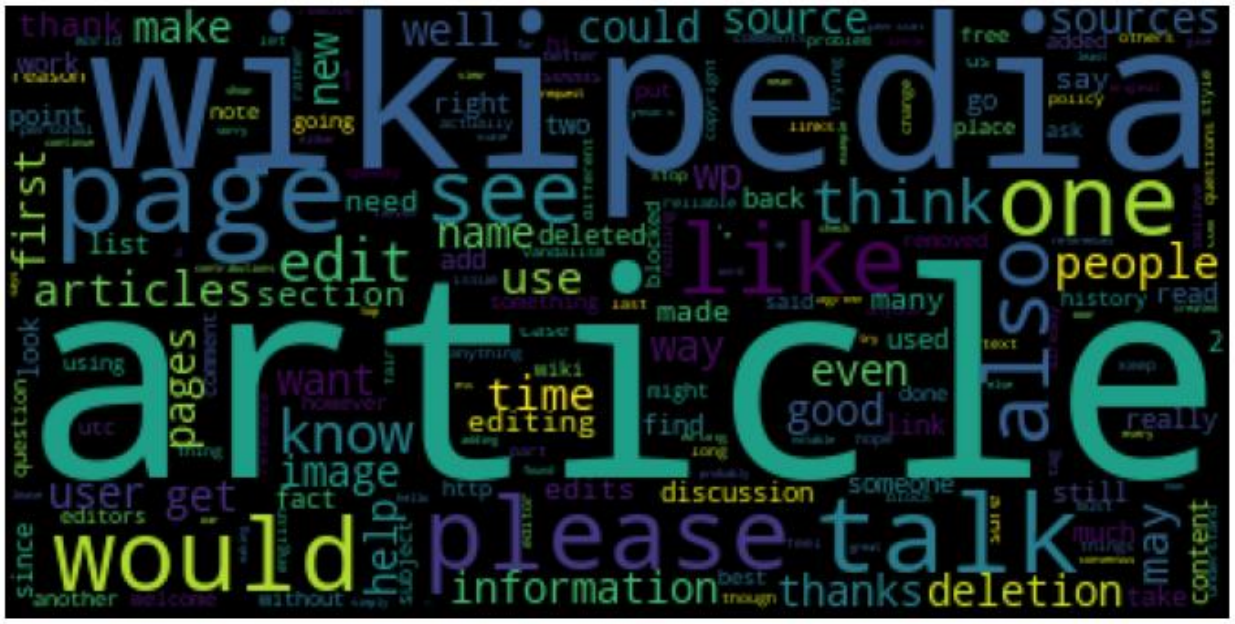


Venkat (IMT2019091)
Vignesh (IMT2019092)

threatening



none

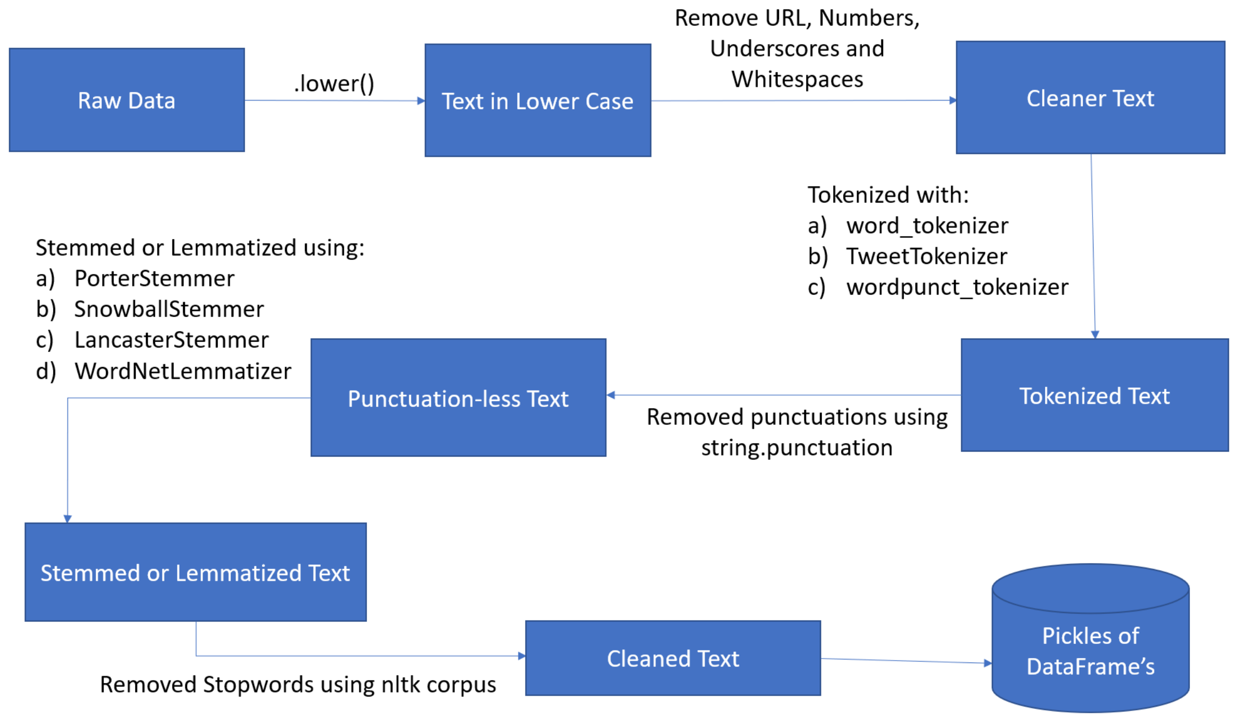


Text Preprocessing

1. NLTK

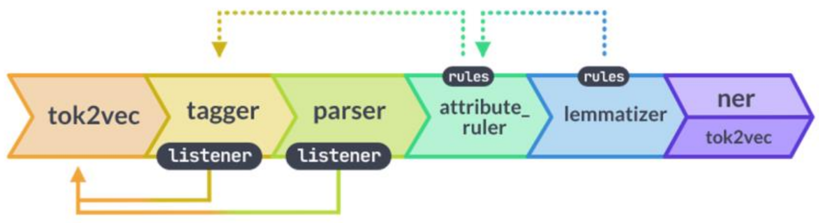
The NLTK Library was used for preprocessing of the textual data to remove and fix erroneous words and clean the text for better processing.

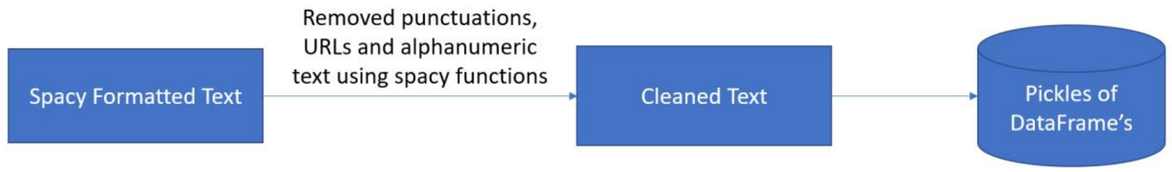
The pipeline flow of preprocessing has been shown in the figure below:



2. Spacy

The Spacy library provides a pre-built pipeline of preprocessing which applies everything shown in first part of the below picture. This tokenizes the words and then does Parts of Speech (POS) tagging, parses the data, applies custom rules to catch specific types of phrases, lemmatizes them and does a named entity recognition (ner). Spacy provides the ability to check if phrases are URL, alphanumeric and dates. This enabled easier removal of these kinds of phrases from the data and obtain a much cleaner dataset of text.




Spacy Formatted Text
Removed punctuations, URLs and alphanumeric text using spacy functions
Cleaned Text
Pickles of DataFrame's

Venkat (IMT2019091)
Vignesh (IMT2019092)

Text Featurization

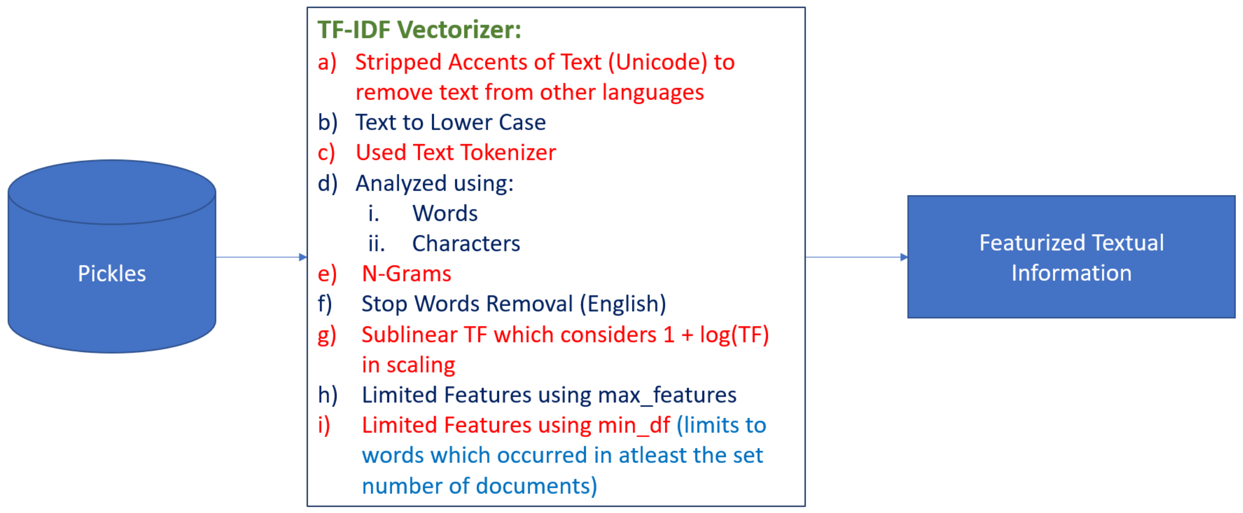
1. Bag of Words (BOW)

In Bag of Words, the text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. The features and the feature vectors obtained are then used for classification. This is done using CountVectorizer class in scikit-learn.

2. TF-IDF with word and character analyzer

TFIDF (term frequency-inverse document frequency), is a numerical statistic that is used to show how importance a word to a document in a collection of documents (corpus). The TFIDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

The pipeline for TFIDF is as shown below:


Pickles
TF-IDF Vectorizer:
a) Stripped Accents of Text (Unicode) to remove text from other languages
b) Text to Lower Case
c) Used Text Tokenizer
d) Analyzed using:
i. Words
ii. Characters
e) N-Grams
f) Stop Words Removal (English)
g) Sublinear TF which considers 1 + log(TF) in scaling
h) Limited Features using max_features
i) Limited Features using min_df (limits to words which occurred in atleast the set number of documents)
Featurized Textual Information

Training Models

Several models were used to train the data and the results were published in Kaggle. The most important submissions and submission from each model is summarized in the table below. The parameters used for the models were hyperparameter tuned using GridSearchCV (GCV) or RandomizedSearchCV (RCV) in cases other than default.

Preprocessing	**Model**	**Validation Score**	**Kaggle Score**
NLTK + BOW	DecisionTreeClassifier (default)	0.859	0.84027
NLTK + TFIDF	LogisticRegressionCV (class_weight='balanced')	0.966	0.96329
NLTK + TFIDF (5 columns) and Spacy + TFIDF (1 column)	LogisticRegressionCV (class_weight='balanced')	0.978	0.95813
NLTK + TFIDF	*LogisticRegression*	*0.988*	*0.97615*

	(class_weight='balanced')		
Spacy + TFIDF	LogisticRegression + GCV (class_weight='balanced') (C= a) Harsh: default b) E. Harsh: 0.1 c) Vulgar: 0.6 d) Threatening: 0.3 e) Disrespect: 0.6 f) T. Hate: 0.4)	0.989	0.96174
NLTK + TFIDF	LogisticRegression + GCV (class_weight='balanced') (C= a) Harsh: 1 b) E. Harsh: 0.125 c) Vulgar: 0.5 d) Threatening: 0.25 e) Disrespect: 0.5 f) T. Hate: 0.5)	0.969	0.93948
NLTK + TFIDF	XGBRFClassifier (default)	0.977	0.92498
NTLK + TFIDF	*SGD* *(loss: 'modified_huber')* *(class_weight: 'balanced')*	*0.978*	*0.97413*
NTLK + TFIDF	SGD + GCV (loss= a) Harsh: 'mh' b) E. Harsh: 'mh' c) Vulgar: 'mh' d) Threatening: 'mh' e) Disrespect: 'mh' f) T. Hate: 'log') ['mh': 'modifed_huber'] (class_weight: 'balanced')	0.975	0.97247
NTLK + TFIDF	Stacking [LR, SDGC, RidgeClassifier]	0.998	0.91683
NTLK + TFIDF	BaggingClassifier (SGDC) SGDC (loss: 'modified_huber')	0.991	0.94339
NTLK + TFIDF	AdaBoosting (SGDC) SGDC (loss: 'modified_huber')	0.975	0.95709
NTLK + TFIDF	RandomForestClassifier + RCV	0.977	0.96658
NTLK + TFIDF	XGBClassifier (default)	0.995	0.96688
NTLK + TFIDF	GradientBoostingClassifier (default)	0.988	0.97536

TFIDF	*Ridge* *(alpha: 30)* *(solver: 'sag')* *(max_iter: 150)*	*0.991*	*0.98313*

From the above table, we can see that Ridge model performed the best among all the models tried so far. The second best we saw was LogisticRegression with default parameters. These were submitted as final models to Kaggle. We also observed that using only processing done with TFIDF character analyzer combined with word analyzer gave better than data obtained by the NLTK or Spacy preprocessed data.

Best Models

The best models are italicized in the above table and are presented below:

Preprocessing	**Model**	**Validation Score**	**Kaggle Score**
NLTK + TFIDF	*LogisticRegression* *(class_weight='balanced')*	*0.988*	*0.97615*
NTLK + TFIDF	*SGD* *(loss: 'modified_huber')* *(class_weight: 'balanced')*	*0.978*	*0.97413*
TFIDF	*Ridge* *(alpha: 30)* *(solver: 'sag')* *(max_iter: 150)*	*0.991*	*0.98313*

Conclusion

In conclusion, the best model we have obtained after trying out several models and hyperparameter tuning is Ridge with basic preprocessing done by TFIDF alone. More concrete analysis by more rigorous hyperparameter tuning could be performed to improve the performance by the three above models.