

# ML Project: Why so harsh?

Team name: Two

Vivek Tangudu (IMT2020110)

Harshadeep Donapati (IMT2020085)

## Introduction

Given a comment, classify it into 6 different classes of harsh. They are -

1. harsh
2. extremely harsh
3. vulgar
4. threatening
5. disrespect
6. targeted hate

For example, consider a comment, "You useless piece of trash! you are such an asshole that you deserve to rot in the gutter alongside sewage." That is certainly harsh! So, the label for this would be [1 1 0 0 1 1]. Translating to the comment being harsh, extremely harsh, disrespectful, and targeted hate.

## Exploratory Data Analysis

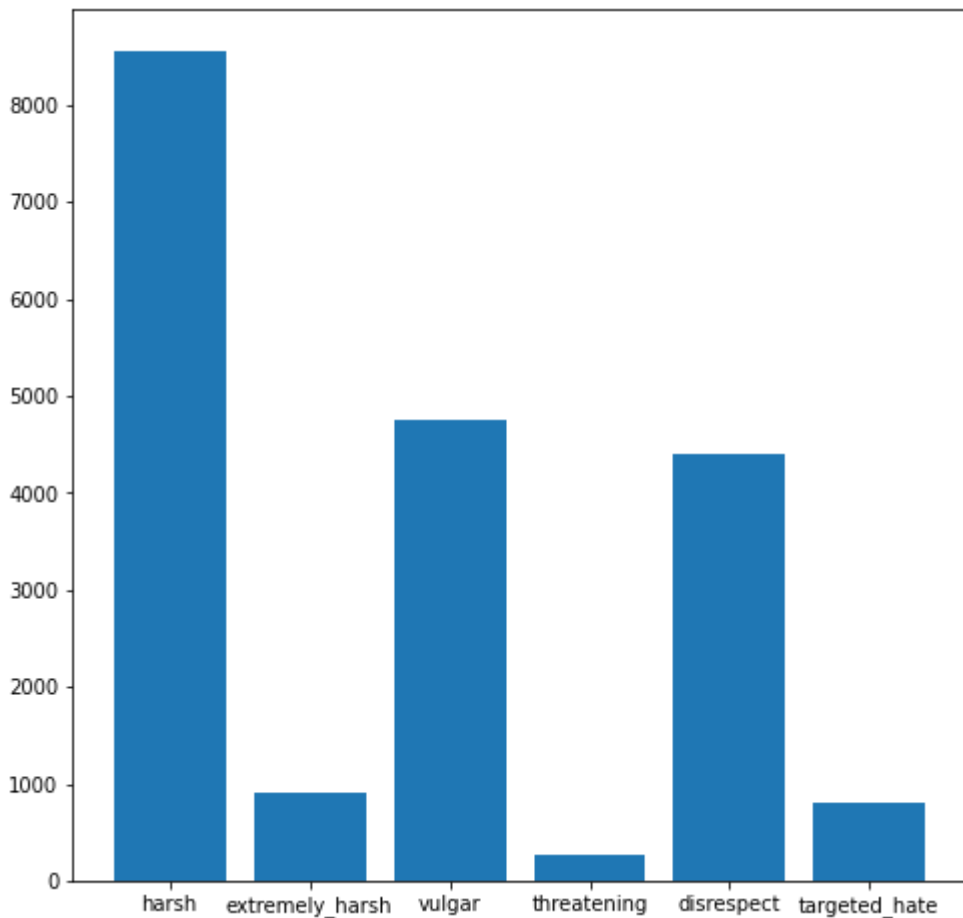
In the train dataset, the number of comments for each class are:

Class name	Number of comments in class
harsh	8559
extremely_harsh	917
vulgar	4742
threatening	268
disrespect	4392
targeted_hate	802

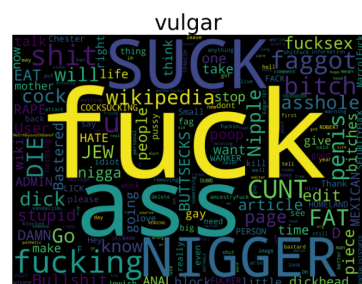
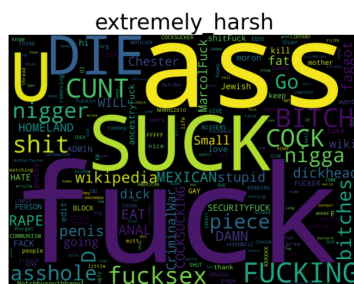
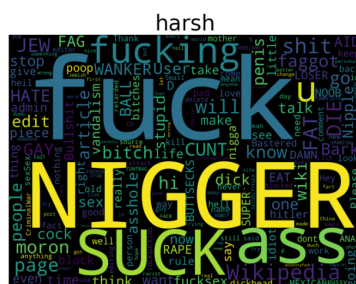
Length of train dataset = 89359

So, we can see that in training data most of the comments are not classified into any of these six classes.

Also, the data is highly unbalanced with most of the comments being harsh, vulgar and disrespect.

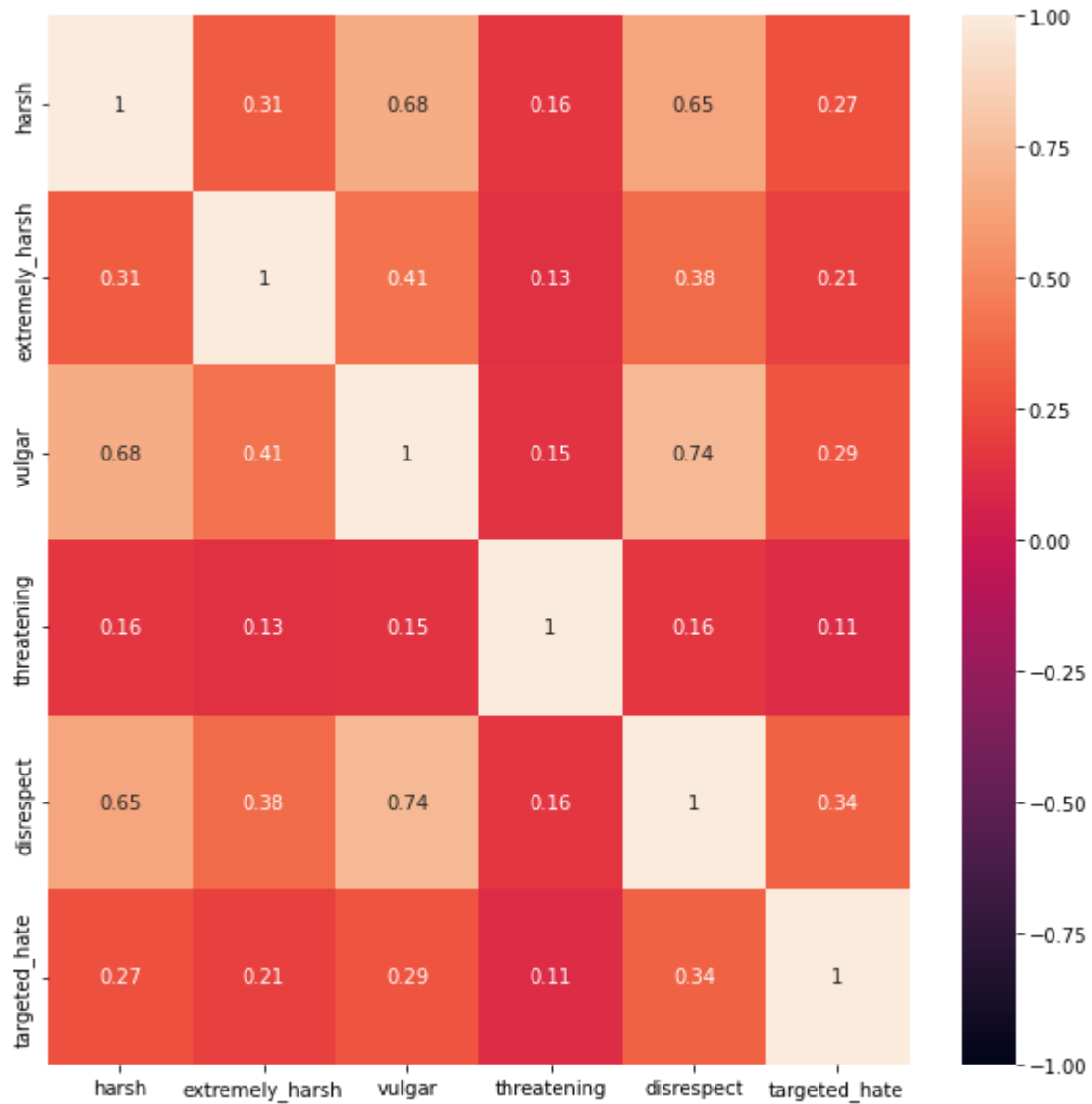


## Wordcloud



## Heatmap

The correlation matrix for the dataset is -



# Preprocessing using NLTK

## Resolving contractions

Resolve contractions using the contractions package.

Eg:

you're -> you are

ima -> I am going to

gotta -> got to

## Converting to lowercase letters

Convert everything into lower case.

## Replacing emoticons

Commonly used emoticons are replaced with their respectful meanings in english.

```
emoticons = {
    ':-)': 'happy',
    ':)': 'happy',
    ':-(': 'frown',
    ':( ': 'frown',
    'xD': 'laugh',
    ':/': 'sad',
    ':|': 'indecision',
    ':o': 'surprise',
    '<3': 'heart'
}
```

## Some cleanup of text

- Removing punctuation marks
- Removing all numbers

## Spell checking

- Out of various spell checkers(textblob, pypellchecker, JamSpell) we got best accuracy with [symspellpy](#). It is also the fastest among all others.

- After this any words containing digits are removed because the spellchecker may correct the spellings of such words. (For Eg, hello123 -> hello)

## **Removing stopwords**

Stopwords are removed using NLTK. Some other words like http, https, www etc. which are left after tokenization are also added into the NLTK's stopwords set.

## **Lemmatization**

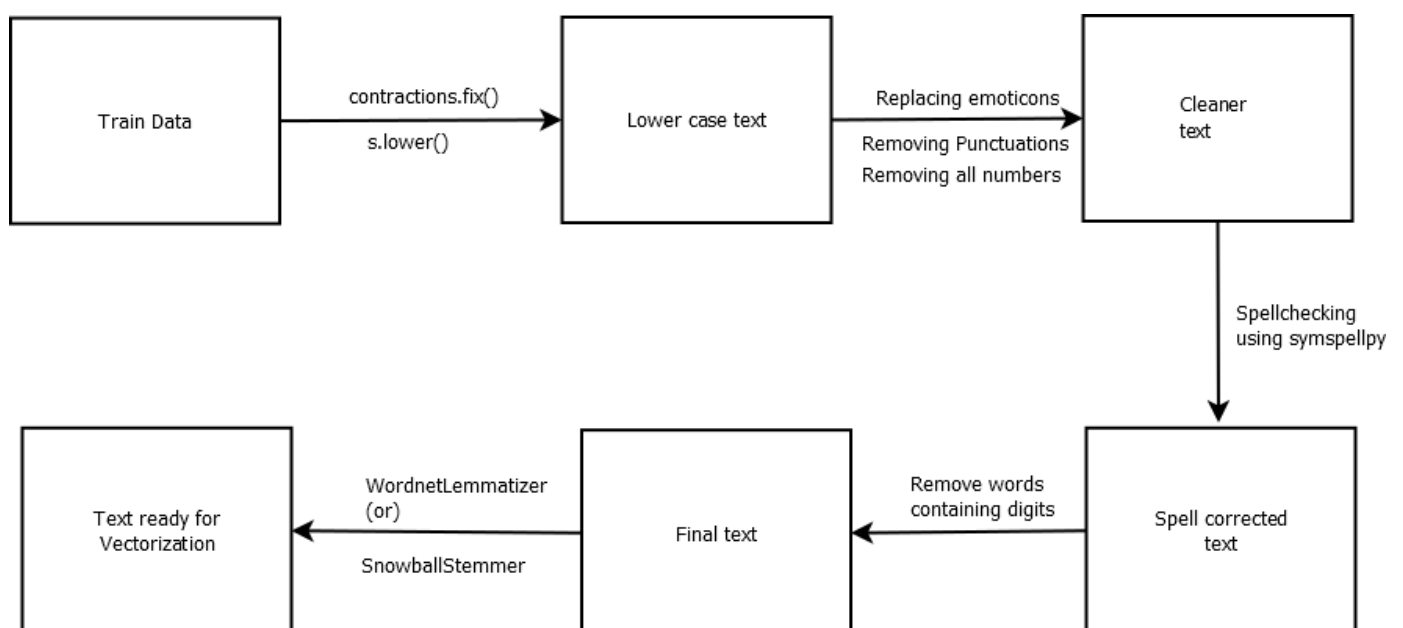
WordNetLemmatizer from NLTK library is used for lemmatizing comments.

## **Stemming**

SnowballStemmer from NLTK library is used for stemming comments. PorterStemmer was also tried, but it gave less accuracy.

Out of both , Stemming gave good results.

## **A complete flow chart of preprocessing**



## **TF-IDF Vectorization**

- $tf - idf(t, d) = tf(t, d) * idf(t)$
- TF-IDF (Term Frequency - Inverse Document Frequency) is a metric to calculate how relevant a word is in a document.
- Both char and word vectorization are done separately.
- Both vectorizers are fitted using whole text (by concatenating test comments and train comments.)
- Then both word and char features are merged using the hstack function.

## **Parameters taken**

- strip\_accents = 'unicode' (Remove accents and perform other character normalisation during the preprocessing step)
- stop\_words = 'english'
- max\_features = None (Consider all possible features for max accuracy)
- min\_df = 2 (ignore terms that appear in less than 2 documents)
- sublinear\_tf = True (Apply sublinear tf scaling, i.e. replace tf with  $1 + \log(tf)$ )
- ngram\_range

## **Pickle**

- After completing text preprocessing and vectorization we start with pickle files.
- Two sets of train and test pickle files are generated. One with lemmatized data and the other with stemmed data.
- The size of the pickle files in both cases is ~500 MB for test data and ~1.2 GB for train data.
- Major models are also stored in pickle files.

## Models

Many models were trained. Hyperparameter tuning was done using GridSearchCV and RandomizedSearchCV computations. In other cases default parameters were used.

Model	roc_auc_score
DecisionTreeClassifier(random_state=0) + max_features = 5000(TF-IDF)	0.7243301030228403
GradientBoostingClassifier + max_features = 5000 (TF-IDF)	0.9377581341818485
svm.SVC	0.9408509947507766
RandomForestClassifier(criterion='gini',max_depth =100, max_features=500, max_leaf_nodes=None, min_samples_split=10, n_estimators=120) + max_features = 5000 (TF-IDF)	0.9539631536900225
ClassifierChain(base_lr, order='random', random_state=0)	0.9776171359862732
SGDClassifier(loss="log", penalty="l2", max_iter=100000, class_weight='balanced')	0.9824327470814311
LogisticRegression(class_weight='balanced', C=1)	0.98410









Model	Kaggle score
LogisticRegression(class_weight='balanced', C=1)	0.98401
SGDClassifier(loss="log", penalty="l2", max_iter=100000, class_weight='balanced')	0.98199

Best model is Logistic Regression.

After the kaggle competition was over, Ridge model was tried and it performed better than Logistic Regression.

Model	Kaggle score
SGDClassifier(loss="log", penalty="l2", max_iter=100000, class_weight='balanced') + WordnetLemmatizer	0.98254
LogisticRegression(class_weight='balanced', C=1) + WordnetLemmatizer	0.9849
Ridge(copy_X=True, solver='sag', random_state=33, alpha=45) + WordnetLemmatizer	0.98542
SGDClassifier(loss="log", penalty="l2", max_iter=100000, class_weight='balanced') + SnowballStemmer	0.98259
LogisticRegression(class_weight='balanced', C=1) + SnowballStemmer	0.98499
Ridge(copy_X=True, solver='sag', random_state=33, alpha=45) + SnowballStemmer	0.98537

	<b>submission_sgdc_stemm.csv</b> Complete (after deadline) · Harsha · 7h ago · SnowballStemmer	<b>0.98112</b>	<b>0.98259</b>	<input type="checkbox"/>
	<b>submission_rid_stemm.csv</b> Complete (after deadline) · Harsha · 8h ago · SnowballStemmer	<b>0.98396</b>	<b>0.98537</b>	<input type="checkbox"/>
	<b>submission_lr_stemm.csv</b> Complete (after deadline) · Harsha · 8h ago · SnowballStemmer	<b>0.98374</b>	<b>0.98499</b>	<input type="checkbox"/>
	<b>submission_rid.csv</b> Complete (after deadline) · Harsha · 9h ago	<b>0.98361</b>	<b>0.98542</b>	<input type="checkbox"/>
	<b>submission_sgdc.csv</b> Complete (after deadline) · Harsha · 9h ago	<b>0.98039</b>	<b>0.98254</b>	<input type="checkbox"/>
	<b>submission_lr.csv</b> Complete (after deadline) · Harsha · 9h ago	<b>0.98286</b>	<b>0.9849</b>	<input type="checkbox"/>

## Miscellaneous

- scikit-learn-intelex package is used to accelerate scikit-learn algorithms by 10 -100 times.
- Time taken for notebook execution is approximately 15 - 20 minutes.

## Future Scope

- Maybe try neural networks and voting classifiers to improve accuracy.