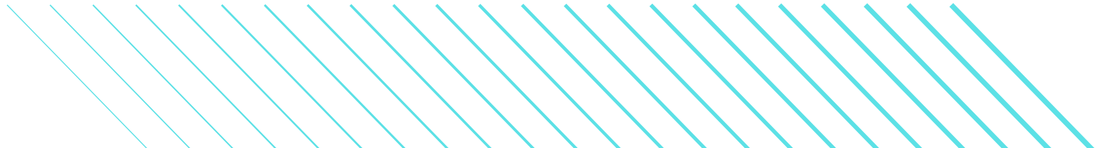
AI 511 - Project

WHY SO HARSH?

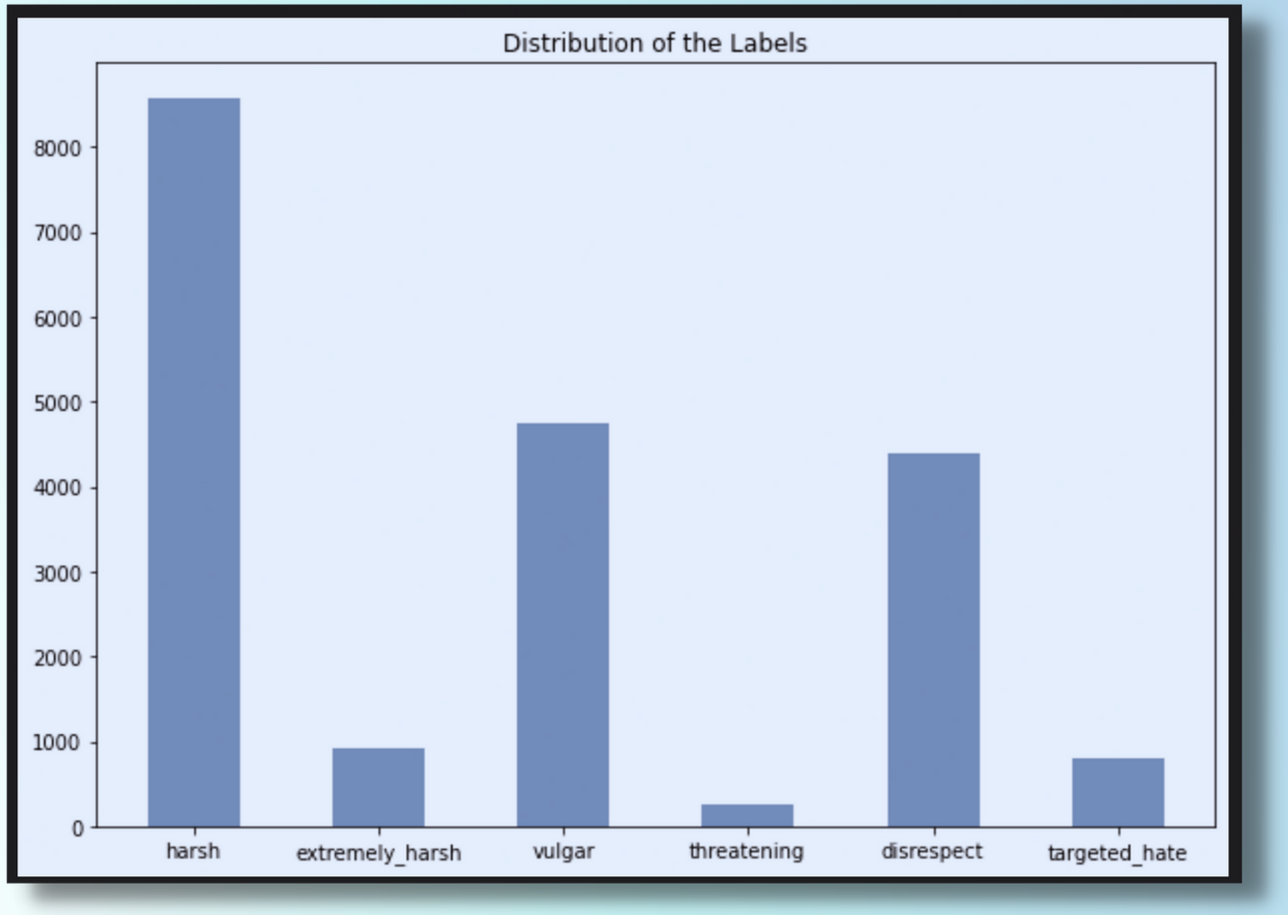


AGENDA

1. Exploratory Data Analysis
2. Text Pre-processing
3. Data Representation
4. Prediction Models
5. Analysis & Results

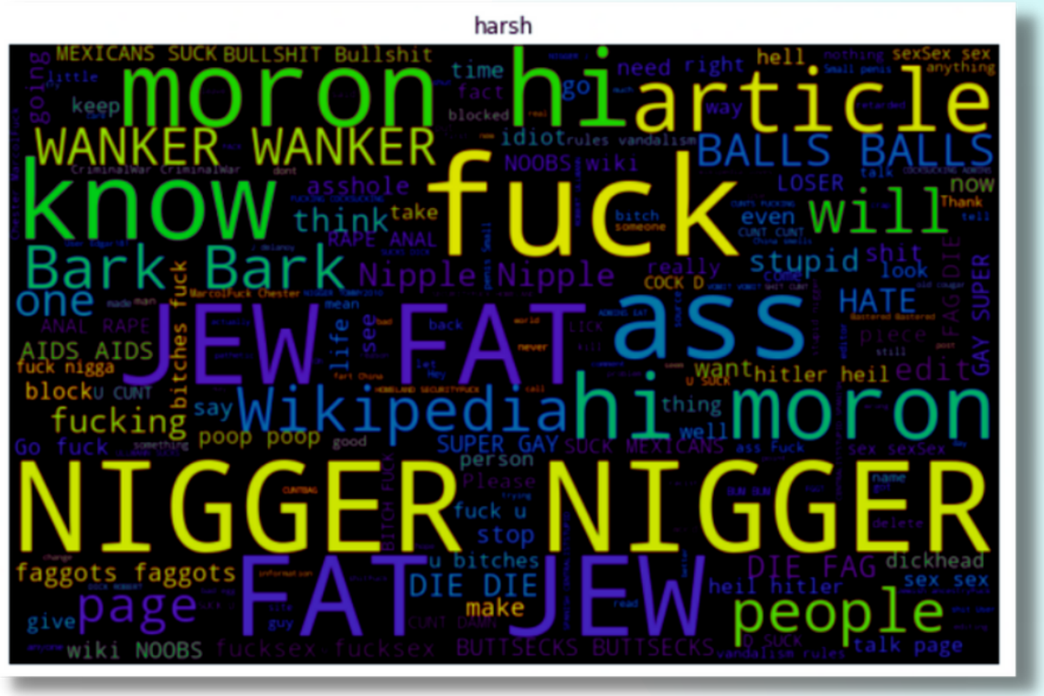
EXPLORATORY DATA ANALYSIS

- Null Values
- Duplicates
- Unique

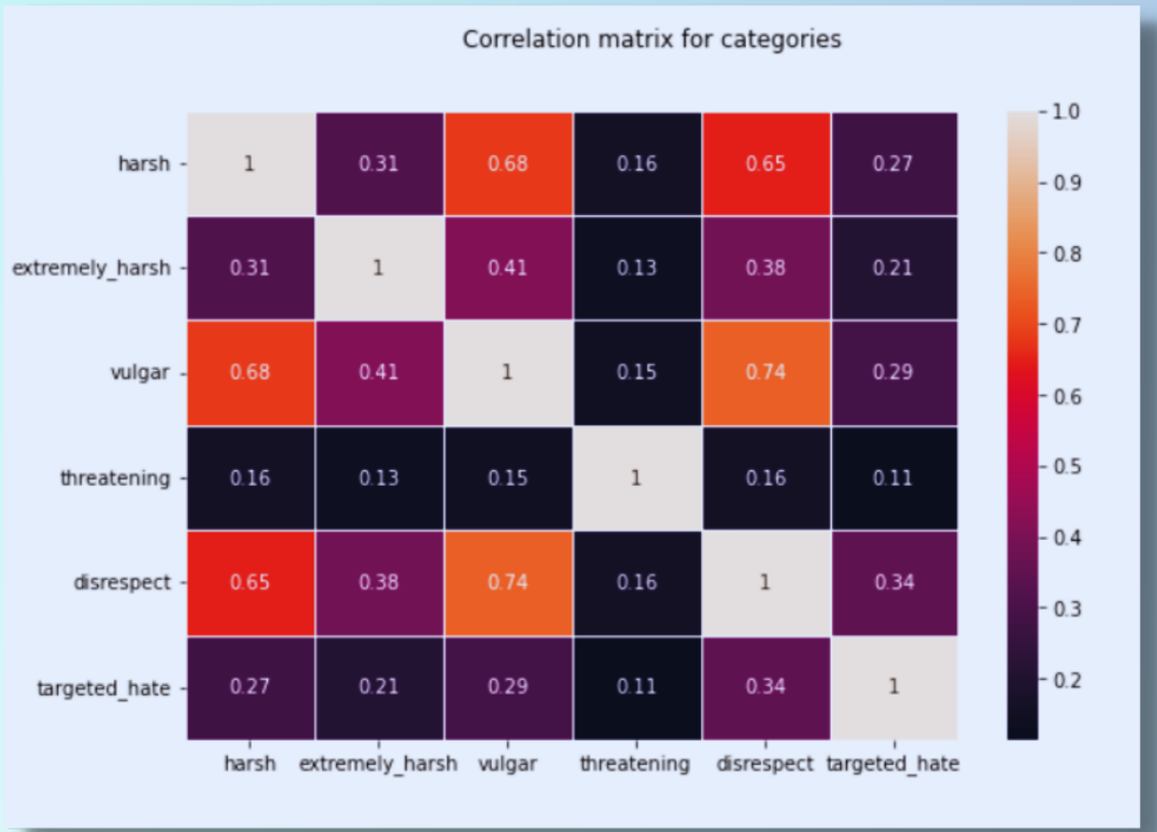


EXPLORATORY DATA ANALYSIS

- Word Cloud



- Correlation Matrix



PRE PROCESSING

- Expand Contractions
- Emoticon's handling and expanding
- Removing numbers, special char, extra spaces and lowering
- Stopwards Removal
- Lemmetization Using POS-Tagging

DATA REPRESENTATION

- N - grams
- Bag of Words
- One-hot encoding
- Word2Vect
- TF-IDF Vectorization

BOW Representation

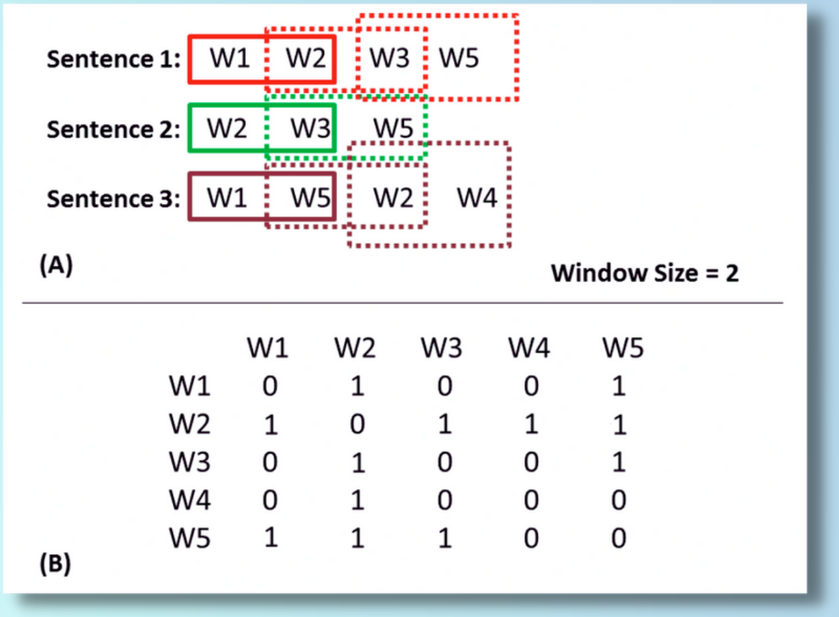
Representing the sentence, "it is the best of the best "

It is the best of a an

[1 ,1 ,1 ,1 ,1 ,0 ,0]
(only the words present in the document are activated)

(or)

[1 ,1 ,2 ,2 ,1 ,0 ,0]
(the word count is taken into consideration instead of activation)



	W1	W2	W3	W4	W5
W1	0	1	0	0	1
W2	1	0	1	1	1
W3	0	1	0	0	1
W4	0	1	0	0	0
W5	1	1	1	0	0

TF-IDF VECTORIZATION

- Term Frequency - Inverse Document Frequency
- It considers the importance of the word in a document into consideration.
- Extract features from the sentences.
- Word Vectorizer - words
- Character Vectorizer - char
- hstack

$$TF - IDF(wt) = TF_{i,j} \times IDF(wt)$$

WORD VECTORIZATION

```
from sklearn.feature_extraction.text import TfidfVectorizer
import re

# TF-IDF Vectorization of words
word_vect = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode',
                            stop_words='english', analyzer='word',
                            token_pattern = '(?u)\\b\\w\\w+\\b\\w{,1}',
                            min_df = 2, max_df = 0.5, ngram_range=(1, 2),
                            max_features = 40000)

word_vect.fit(train_text_list)
word_train_features = word_vect.transform(train_text_list)
word_test_features = word_vect.transform(test_text_list)
```

CHARACTER VECTORIZATION

```
from sklearn.feature_extraction.text import TfidfVectorizer

# TF-IDF Vectorization of char
char_vect = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode',
                            analyzer='char', stop_words = 'english',
                            ngram_range=(2, 6), min_df = 2,
                            max_df = 0.5, max_features = 40000)

char_vect.fit(train_text_list)
char_train_features = char_vect.transform(train_text_list)
char_test_features = char_vect.transform(test_text_list)
```

HSTACK

```
from scipy.sparse import hstack
train_features = hstack([word_train_features, char_train_features])
test_features = hstack([word_test_features, char_test_features])
print(train_features.shape)
print(test_features.shape)
```

PREDICTION MODELS

- **Probabilistic Models**
  Logistic Regression
  Naive Bayes
- **Tree Based Models**
  Random Forest
- **Using Cross Validation**
  LogisticRegressionCV
- **Ensemble Models**
  XG Boosting
- **Linear Models**
  Ridge Classifier
  SGD Classifier

RESULTS

Model	Validation score	Kaggle score
Logistic Regression	**0.98587**	**0.98489**
Ridge	**0.98558**	**0.98509**
XG Boosting	**0.97319**	**0.96883**
Random Forest	**0.96701**	**0.95645**
Naive Bayes	**0.93095**	**0.92714**
SGD Classifier	**0.88276**	**0.88871**

LOGISTIC REGRESSION

- class_weight = 'balanced'

  automatically adjust weights inversely proportional to class frequencies in the input data

- solvers and features
- GridSearchCV
- over sampling
- under-sampling

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import pickle as pk

pred_df = {
    'id' : test_data['id']
}

for i in Columns:
    model = LogisticRegression(class_weight = 'balanced')
    x_train = train_features
    y_train = train_data[i].to_numpy()
    model.fit(x_train, y_train)
    pk.dump(model, open(i+"_Logistic.pkl", "wb"))
    y_pred = model.predict_proba(test_features)[:, 1]
    pred_df[i] = y_pred

df_submit = pd.DataFrame(pred_df)
df_submit.to_csv('logistic_regression_sub.csv', index=None)
```

RIDGE CLASSIFICATION

- alpha = 32
  controlling regularization strength
- fit_intercept = True
  fit the intercept for this model
- solver = 'sag'
  iterative procedure, uses the dedicated regularized least-squares routine
- tol = 0.0006
  precision of the solution
- max_iter = 500
  Maximum number of iterations for gradient solver

RIDGE CLASSIFICATION

```
from sklearn.linear_model import Ridge
import pickle as pk
import numpy as np

pred_df = {
    'id' : test_data['id']
}

for i in Columns:
    model = Ridge(alpha=32, copy_X=True, fit_intercept=True, solver='lsqr',
                                    max_iter=500, random_state=0, tol=0.0006)
    x_train = train_features
    y_train = train_data[i].to_numpy()
    model.fit(x_train, y_train)
    pk.dump(model, open(i+"_Ridge.pkl", "wb"))
    y_pred = model.predict(test_features)
    pred_df[i] = y_pred

df_submit = pd.DataFrame(pred_df)
df_submit.to_csv('ridge_sub.csv', index=None)
```

Team Members:

1. Niteesh - IMT2020007
2. Chakradhar - IMT20200

THANK YOU :)