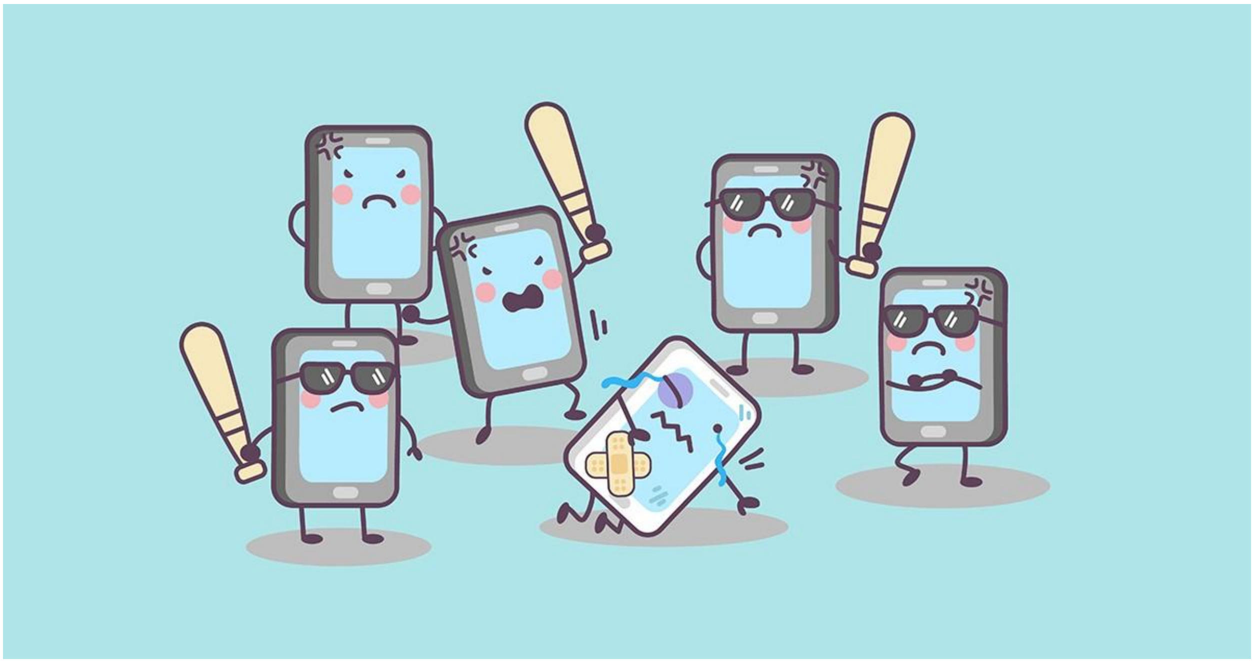
AI 511 ML **Mini Project**



Why So Harsh?

Team Members:

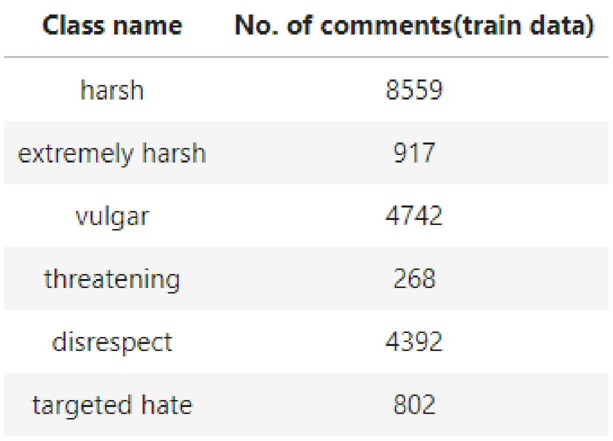
1. Niteesh - IMT2020007
2. Chakradhar - IMT2020021

Introduction:

In the given dataset for every sample given a comment, classify them into 6 different classes of being harsh as follows:

“Harsh”, “Extremely Harsh”, “Vulgar”, “Threatening”, “Disrespect”, “Targeted Hate”.

In the train data set a number of texts with each class were observed as follows:

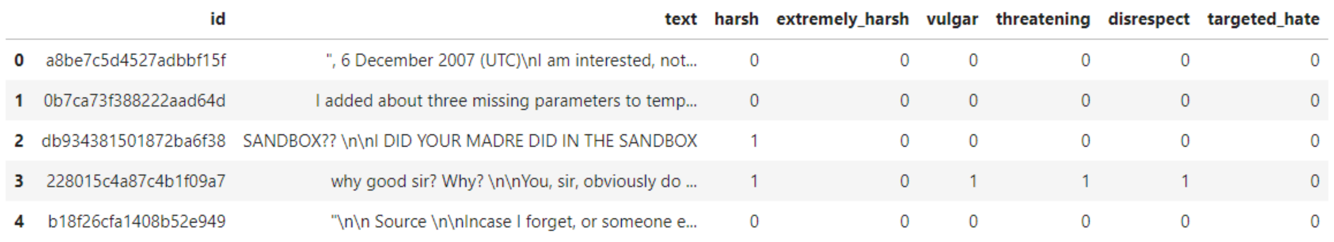


Class name	No. of comments(train data)
harsh	8559
extremely harsh	917
vulgar	4742
threatening	268
disrespect	4392
targeted hate	802

From the above table we can observe that there are a lot of negative values in the training data. Analysis of data is done in the EDA section.

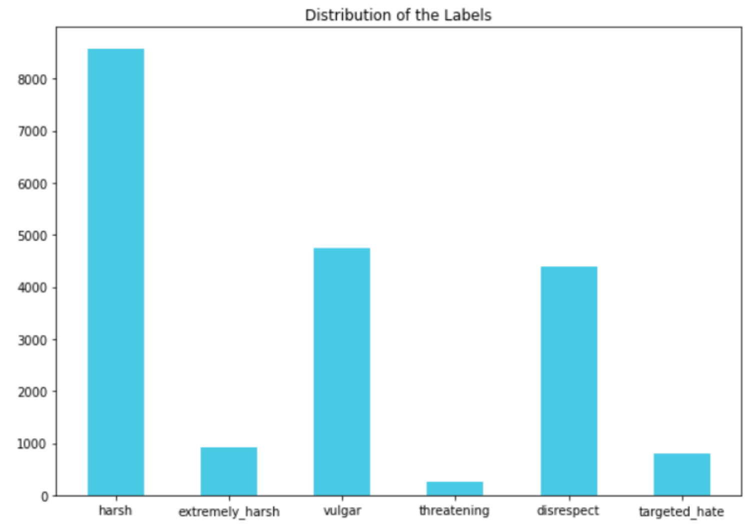
Exploratory Data Analysis:

1. Null Values: No null values in the Dataset
2. Duplicates: No duplicates
3. Unique:
    - The 6 classes have two unique values 0, 1
    - In the dataset every sample has unique id value

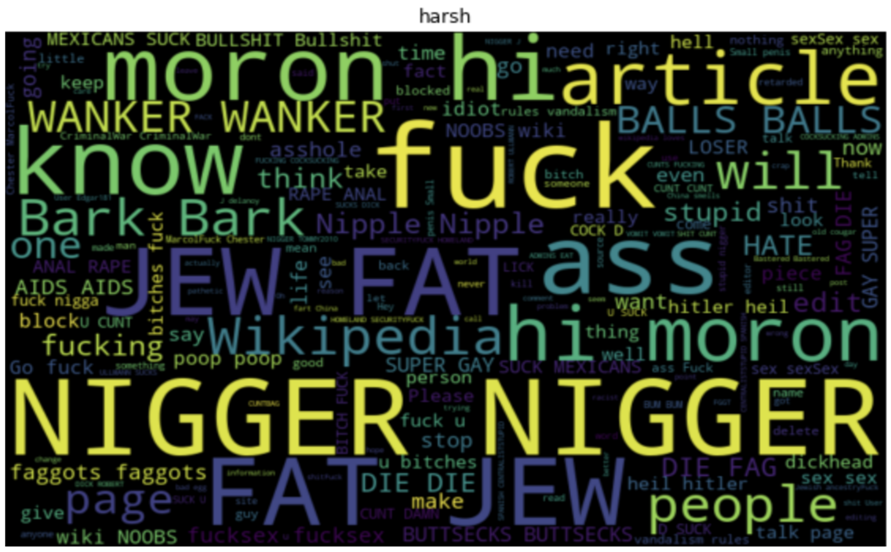


	id	text	harsh	extremely_harsh	vulgar	threatening	disrespect	targeted_hate
0	a8be7c5d4527adbbf15f	", 6 December 2007 (UTC)\nI am interested, not...	0	0	0	0	0	0
1	0b7ca73f388222aad64d	I added about three missing parameters to temp...	0	0	0	0	0	0
2	db934381501872ba6f38	SANDBOX?? \n\nI DID YOUR MADRE DID IN THE SANDBOX	1	0	0	0	0	0
3	228015c4a87c4b1f09a7	why good sir? Why? \n\nYou, sir, obviously do ...	1	0	1	1	1	0
4	b18f26cfa1408b52e949	"\n\n Source \n\nIncase I forget, or someone e...	0	0	0	0	0	0

Distribution of labels:



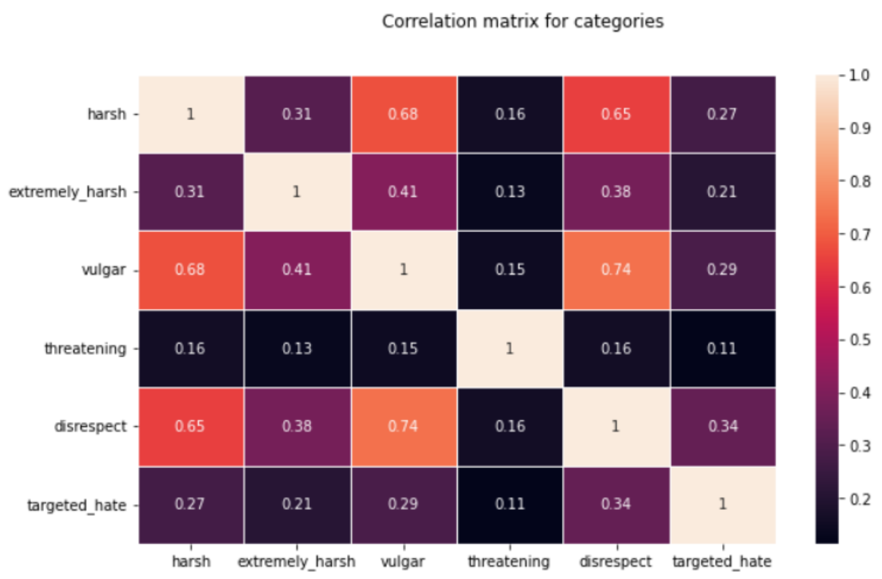
Word Cloud gives the image composed of words that are used in a specific column, where the size of each word denotes its frequency or significance and given is the word cloud representation of the text that has the labels as ‘harsh’.



The correlation matrix:

Helps to check the relation between classes.

As we can see in the table where no two labels are not correlated to each other, so we can say that classes are independent of each other.



Text pre-processing:

1. Expand Contractions:

- In order to convert them into a presentable form that is analyzable. Expanding contractions can further reduce dimensionality, or even help filter out stopwords.
- A contraction is an abbreviation for a sequence of words. Expanding those words into the context.

Before applying expand contractions on data

```
'", 6 December 2007 (UTC)\nI am interested, not in arguing, but in the policies which resolve our ongoing content dispute. Also, se
e Wikipedia: WikiProject United States presidential elections for what I\'ll be working on. Also, the moneybomb closer just self-re
verted on two different requests, which echoed what I would have requested.  I will rephrase #3, which I didn\'t see an answer to,
building on our agreement that ""moneybomb"" should not be a redlink: Given the deletion reversion, what should be the outline of t
he article called ""moneybomb"" or should it be submitted for AFD again in due time? (If the latter, see the previous version of #
3.) However, this version will require a detailed answer because any ambiguity will only necessitate clarifying questions.   22:3
2"'
```

After applying expand contractions on data

```
'", 6 December 2007 (UTC) I am interested, not in arguing, but in the policies which resolve our ongoing content dispute. Also, see
Wikipedia: WikiProject United States presidential elections for what I will be working on. Also, the moneybomb closer just self-rev
erted on two different requests, which echoed what I would have requested. I will rephrase #3, which I did not see an answer to, bu
ilding on our agreement that ""moneybomb"" should not be a redlink: Given the deletion reversion, what should be the outline of the
article called ""moneybomb"" or should it be submitted for AFD again in due time? (If the latter, see the previous version of #3.)
However, this version will require a detailed answer because any ambiguity will only necessitate clarifying questions. 22:32" '
```

2. Emoticons Handling and Expanding:

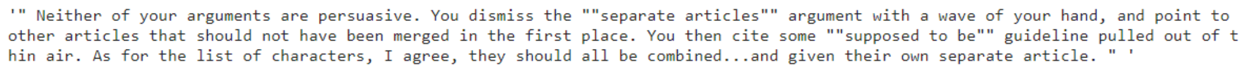
- **An emoticon** is a short form of “Emotion & Icon”. Replacing the emoticon with appropriate meaning of the text.

Examples: :-) —> Slightly happy

3. <u>CleanUp & Normalizing Text:</u>

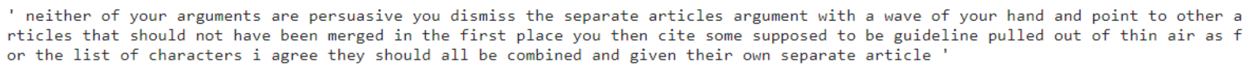
- Converting upper-case characters to lower-case
- Removing all the Punctuation characters from text execpt '?', '!'
- Removing all extra spaces between words

Text before:



```
'" Neither of your arguments are persuasive. You dismiss the ""separate articles"" argument with a wave of your hand, and point to
other articles that should not have been merged in the first place. You then cite some ""supposed to be"" guideline pulled out of t
hin air. As for the list of characters, I agree, they should all be combined...and given their own separate article. " '
```

Text after:

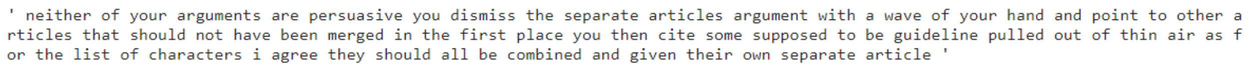


```
' neither of your arguments are persuasive you dismiss the separate articles argument with a wave of your hand and point to other a
rticles that should not have been merged in the first place you then cite some supposed to be guideline pulled out of thin air as f
or the list of characters i agree they should all be combined and given their own separate article '
```

4. <u>Stopwords Removal:</u>

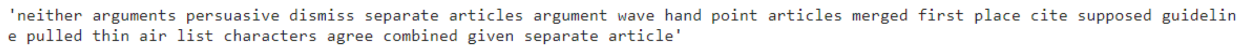
- Removing some words that don’t directly apply to interpretation.
- Removing stop word links.

Text before



```
' neither of your arguments are persuasive you dismiss the separate articles argument with a wave of your hand and point to other a
rticles that should not have been merged in the first place you then cite some supposed to be guideline pulled out of thin air as f
or the list of characters i agree they should all be combined and given their own separate article '
```

Text after



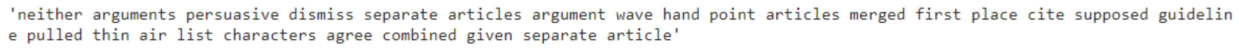
```
'neither arguments persuasive dismiss separate articles argument wave hand point articles merged first place cite supposed guidelin
e pulled thin air list characters agree combined given separate article'
```

5. <u>Lemmatization using POS tagging</u>

- The process of grouping together the inflected forms of a word so they can be analyzed as a single item, identified by the word's lemma, or dictionary form.
- And there is difference between the lemmatization using POS tagging and without using POS tagging.

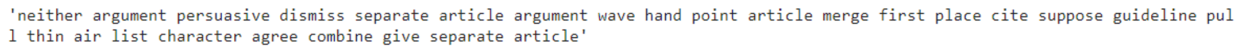
Ex: Without POS - tagging
    He leaves office early - He leaf office early
Ex: With POS - tagging
    He leaves office early - He leave office early

Text before



```
'neither arguments persuasive dismiss separate articles argument wave hand point articles merged first place cite supposed guidelin
e pulled thin air list characters agree combined given separate article'
```

Text after



```
'neither argument persuasive dismiss separate article argument wave hand point article merge first place cite suppose guideline pul
l thin air list character agree combine give separate article'
```

6. <u>Vectorization of Data:</u>

- The Process of converting words into numbers is Vectorization. This is done using TF-IDF. TF-IDF or Term Frequency–Inverse Document Frequency, is a numerical statistic that’s intended to reflect how important a word is to a document. The TF-IDF method captures the semantics of the text and does not equal importance to all words in the text. With the TF-IDF's valuation of each word in a document, it can identify words with the highest values and deem them to be keywords.
- Formula to calculate complete TF-IDF value is –

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

Word Vectorizer: Extract all the word features in the given data by the word count of the required range.

```python
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 Vectorizer: Extract all the character features in the given data of required range using ngram feature.

```python
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: Using this we get combined features of both word and character vectorization in 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)
```

The hyper tuning parameters that we used as follows:
sublinear_tf = true - Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).
strip_accents='unicode' - Remove accents and perform other character normalization during the preprocessing step. 'unicode' is a slightly slower method that works on any characters.
analyzer - Whether the feature should be made of word or character n-grams.
ngram_range - *The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used.*
min_df - When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature.
max_df - When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold.

Pickles:

A pickle file is used to store a trained model. The file format for a pickle file is '.pckl'. Without having to retrain, the learned model may be imported from this file and used to produce predictions. A pickle file can be used to cut training time. Using a pickle file makes our code more effective and simpler to debug.

Prediction Models:

1. Probabilistic models:

   These models are a statistical technique used to take into account the impact of random events or actions in predicting the potential occurrence of future outcomes.

   - **Logistic Regression:**

     For the current dataset accuracy we got in regression is "0.98489", which was better than Naive Bayes.

```python
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score,accuracy_score
from sklearn import metrics
import pickle as pk

pred_df = {'id' : test_data['id']}
Columns = ['harsh', 'extremely_harsh', 'vulgar', 'threatening', 'disrespect', 'targeted_hate']

trainX, testX, trainY, testY = train_test_split(train_features, train_data[Columns], train_size=0.8)
testP = pd.DataFrame()

model = LogisticRegression(class_weight = 'balanced')
# model = LogisticRegressionCV(class_weight = 'balanced', solver='liblinear', n_jobs = -1)
for i in Columns:
    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"))
    pred_df[i] = model.predict_proba(test_features)[:, 1]

    yTrain = testY[i]
    testP[i] = model.predict_proba(testX)[:, 1]

print("F1 accuracy:", f1_score(testY.values, testP, average=None))
print("Accuracy: ", accuracy_score(testY.values, testP))
df_submit = pd.DataFrame(pred_df)
df_submit.to_csv('logistic_regression_sub.csv', index=None)
```

- **Logistic regression - Grid Search CV**
  By using cross validation method of GridSearchCV - we can be able to find the "C" - hyper tuning parameter for Logistic regression

```python
model = LogisticRegression(class_weight = 'balanaced')
Columns = ['harsh', 'extremely_harsh', 'vulgar', 'threatening', 'disrespect', 'targeted_hate']
param_gridLR = {
    'C': [0.1, 0.2, 0.3, 0.4, 1],
    'penalty': ['l2'],
    'solver': ['lbfgs', 'liblinear']
}

for i in Columns:
    searchLR = GridSearchCV(model, cv = 5, param_grid=param_gridLR, verbose = 3, refit = True)
    x_train = train_features
    y_train = train_data[i].to_numpy()
    searchLR.fit(x_train , y_train)
    print('Mean Accuracy: %.3f' % searchLR.best_score_)
    print('Config: %s' % LR_searchLR.best_params_)
```

- **Naive Bayes:**
  The output accuracy was found to be "0.92714". Here the Kaggle score has changed a lot from the validation score. In this, gaussian is not a good fit. There are various types of Naive bayes models: GaussianNB, BernoulliNB, CategoricalNB, ComplementNB, MultinomialNB. From with ComplementNB gave the best accuracy results.

```python
import pickle as pk
from sklearn.naive_bayes import ComplementNB as naive_bayes

pred_df = {'id' : test_data['id']}
Columns = ['harsh', 'extremely_harsh', 'vulgar', 'threatening', 'disrespect', 'targeted_hate']

model = naive_bayes()
for i in Columns:
    x_train = train_features
    y_train = train_data[i].to_numpy()
    model.fit(x_train, y_train)
    pk.dump(model, open(i+"_NaiveBayes.pkl", "wb"))
    pred_df[i] = model.predict_proba(test_features)[:, 1]

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

2. <u>Tree-Based Models:</u>

- **Random Forest:**

  The training time of Random Forest is very high here. Output accuracy was found to be “0.95645” which is not good. By changing depth also results are not good.

```python
import pickle as pk
from sklearn.ensemble import RandomForestClassifier as rfc

model = rfc(max_features = 999, max_depth = 100, min_samples_split = 10,
            criterion = 'gini', n_estimators = 120, max_leaf_nodes = None,
            min_weight_fraction_leaf = 0.0)

pred_df = {'id' : test_data['id']}
Columns = ['harsh', 'extremely_harsh', 'vulgar', 'threatening', 'disrespect', 'targeted_hate']

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

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

3. <u>Ensemble Models:</u>

   This model is based on combining several models. This approach allows production of better predictive performance compared to a single model.

   - **XG Boosting:**

     With XG boosting the prediction accuracy score was found to be “0.96883”.

```python
import xgboost as xgb
import pickle as pk

pred_df = {'id' : test_data['id']}
Columns = ['harsh', 'extremely_harsh', 'vulgar', 'threatening', 'disrespect', 'targeted_hate']

model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
for i in Columns:
    x_train = train_features
    y_train = train_data[i].to_numpy()
    model.fit(x_train, y_train)
    pk.dump(model, open(i+"_XGBoosting.pkl", "wb"))
    pred_df[i] = model.predict_proba(test_features)[:, 1]

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

4. Linear Models:

- **Ridge:**

  Using the ridge classifier the output accuracy score obtained was "0.98509".

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

pred_df = {'id' : test_data['id']}
Columns = ['harsh', 'extremely_harsh', 'vulgar', 'threatening', 'disrespect', 'targeted_hate']

model = Ridge(alpha=32, copy_X=True, fit_intercept=True, solver='lsqr',
              max_iter=500, random_state=0, tol=0.0006)
for i in Columns:
    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"))
    pred_df[i] = model.predict(test_features)

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

- **SGD Classifier:**

  With this classifier, we got less accuracy test scores. Here the accuracy was "0.88871".

```python
import pickle as pk
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.calibration import CalibratedClassifierCV

pred_df = {'id' : test_data['id']}
Columns = ['harsh', 'extremely_harsh', 'vulgar', 'threatening', 'disrespect', 'targeted_hate']

model = make_pipeline(StandardScaler(with_mean=False), SGDClassifier(loss='hinge',max_iter=1000, tol=1e-3))
for i in Columns:
    x_train = train_features
    y_train = train_data[i].to_numpy()
    model.fit(x_train, y_train)
    calibrator = CalibratedClassifierCV(model, cv='prefit')
    model1 = calibrator.fit(x_train, y_train)
    pk.dump(model1, open(i+"_SGDC.pkl", 'wb'))
    pred_df[i] = model1.predict_proba(test_features)[:, 1]

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

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

Best Kaggle Score:

The model is Ridge classification with kaggle score: 0.98509

References:

NLP Lemmatization PreProcessing

Logistic Regression Naive Bayes

Ridge Classification

Random Forest

XG Boosting

SGD Classifier