Project Report

Why So Harsh?

Created By - Team NHK?
IMT2019081 - Shivansh Sethi
IMT2019085 - Sukhamjot Singh

Task

Given a comment, classify it into “harsh”, “extremely harsh”, “vulgar”, “threatening”, “disrespect”, “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.

Data

The data set was provided on Kaggle. The data set
included two files:

- train.csv - the training set
- test.csv - the test set

The training data set contained 127656 data points. The test data set contains
31915 data points.

Exploratory Data Analysis

Frequency of Words

Created Worcloud for Each Category.

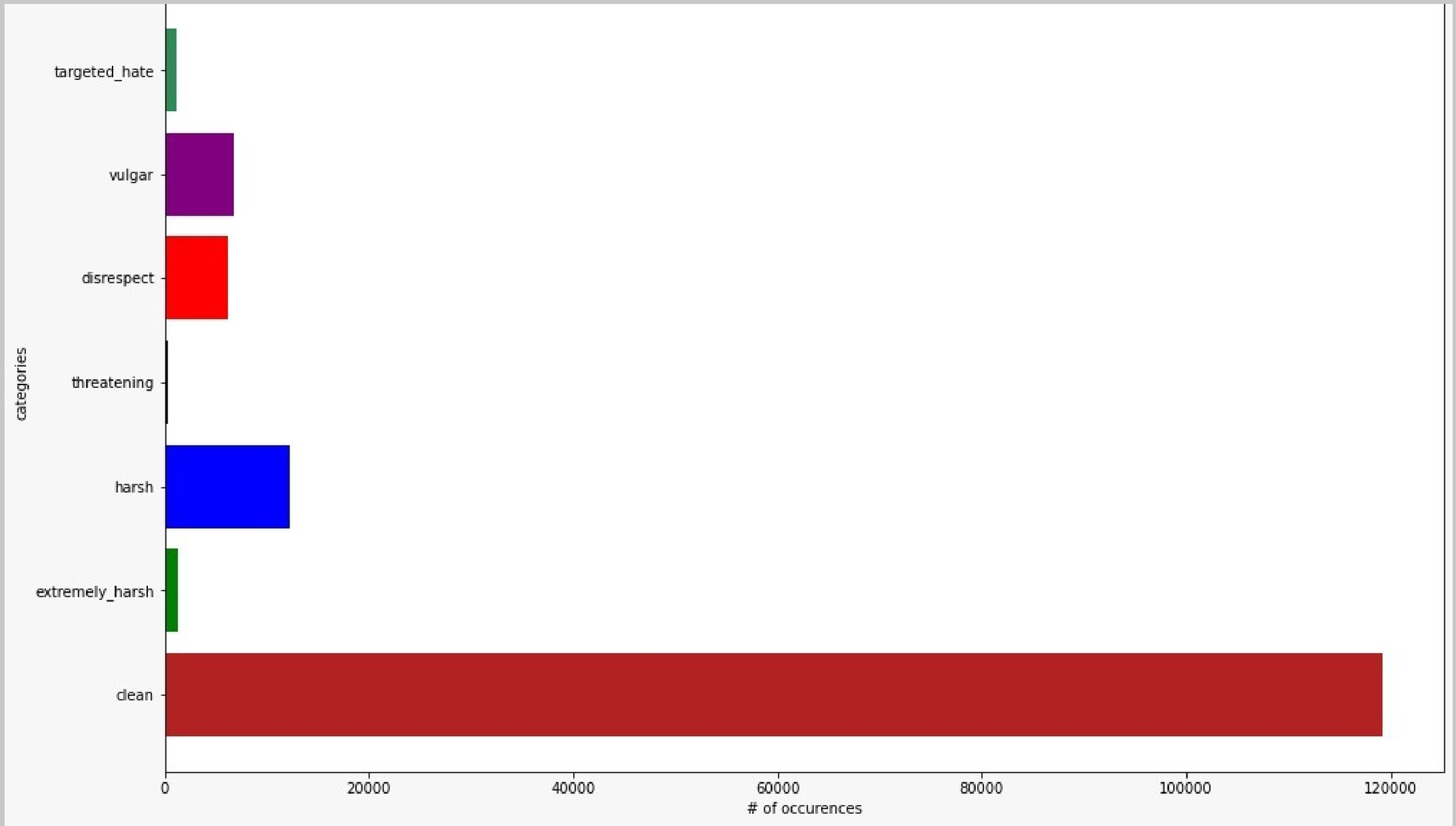
Correlation Map

Created heatmap for the numerical features we added.

Barplot

Displayed the number of Label counts.

Percentage of Clean Comments = 89%



Words frequented in Harsh Comments

Words frequented in targeted_hate Comments

Words frequented in Clean Comments

Words frequented in threatening Comments

Words frequented in extremely_harshComments

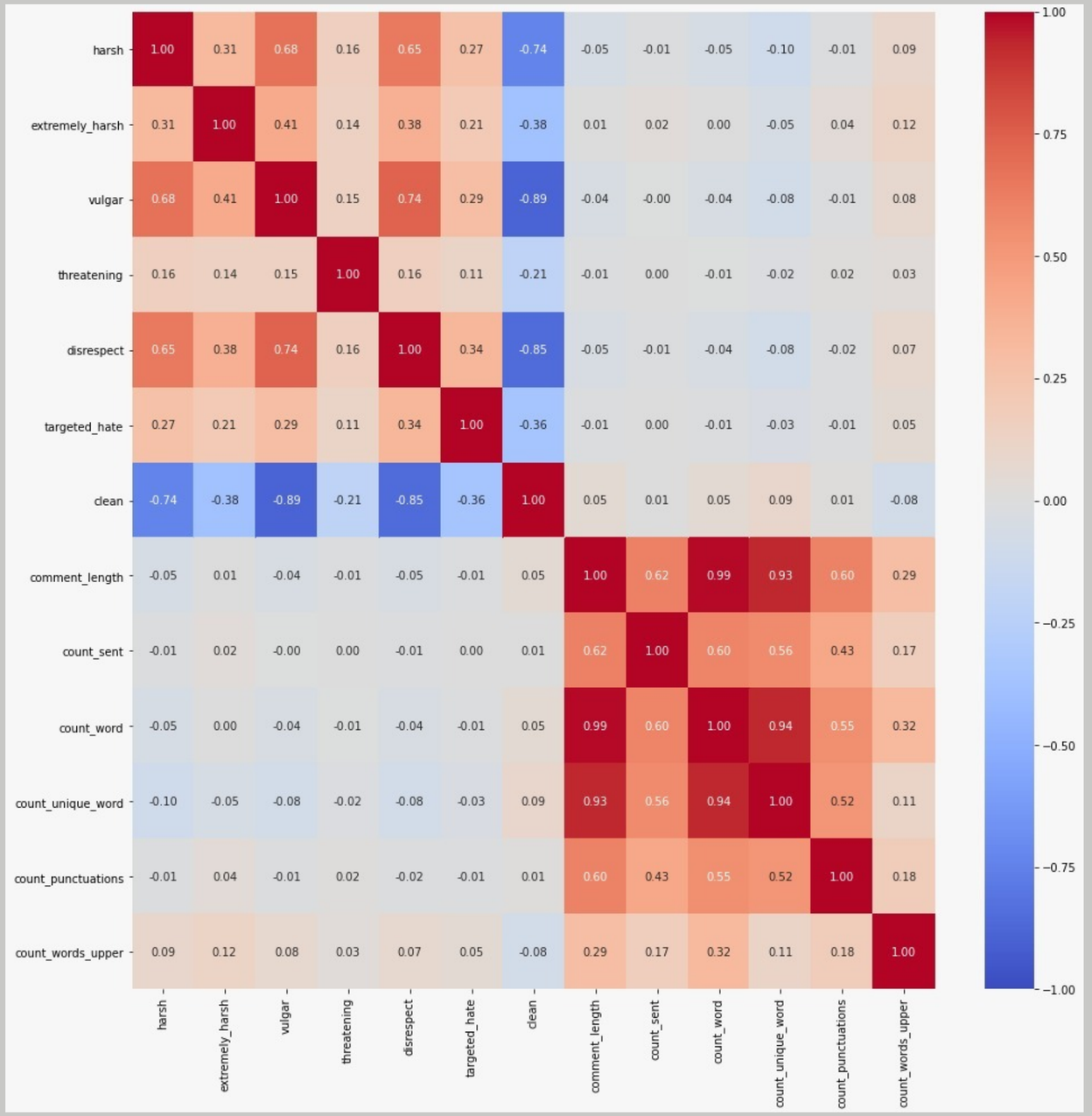
Words frequented in disrespect Comments

Feature Engineering

Numerical Features added

- Length of whole Comment
- Sentence count in each comment:
- Word count in each comment:
- Unique word count
- Punctuation count
- Upper Case words count
- Number of Clean words

Correlation between numerical features added.



Preprocessing

- Removed numbers, punctuation and usernames.
- Removed tabs, extra spaces.
- Converted text to lowercase.
- Modifications to text grammar (haven't -> have not).
- Removed stopwords("the", "is").
- Lemmatized the text features.

Vectorization

```
word_vectorizer = TfidfVectorizer(
        sublinear_tf=True,
        strip_accents='unicode',
        tokenizer=lambda x: re.findall(r'[^\p{P}\W]+', x),
        analyzer='word',
        token_pattern='(?u)\\b\\w\\w+\\b\\w{,1}',
        min_df=5,
        norm='l2',
        ngram_range=(1, 1),
        max_features=20000
```

```
char_vectorizer = TfidfVectorizer(
 sublinear_tf=True,
 strip_accents='unicode',
 analyzer='char',
 token_pattern=None,
 min_df=5,
 ngram_range=(2, 4),
 max_features=20000)
```

- We used TF-IDF vectorizer to vectorize the words and chars separately and then used make_union() function to fit the text data on the vectorizers.
- In char_vectorizer n-gram_range = (2,4) which vectorizes list of characters combined in length 2 - 4.
- Max features were adjusted to get the best possible score.

Top 3 approaches

# 1	Used a combination of Ridge Classifier and Logistic Regression with hyperparameter tuning.
# 2	Used Ridge Classifier with hyperparameter tuning
# 3	Used Logistic Regression with hyperparameter tuning

Approach #1

```
if x in ['targeted_hate', 'threatening']:
  model = LogisticRegression(n_jobs=-1, random_state=0, C=3)
  model.fit(X_encoded,y[x])
  score = roc_auc_score(y_valid[x],model.predict_proba(X_valid)[:,1])
  predict_dic[x] = model.predict_proba(test_encoded)[:,1].tolist()
else:
  model = RidgeClassifier(alpha=27,fit_intercept=True, solver='sag', max_iter=200,random_state=0)
  model.fit(X_encoded,y[x])
  d = model.decision_function(X_valid)
  probs = np.exp(d) / np.sum(np.exp(d))
  score = roc_auc_score(y_valid[x],probs)
  d_test = model.decision_function(test_encoded)
  probs_test = np.exp(d_test) / np.sum(np.exp(d_test))
  predict_dic[x] = probs_test.tolist()
```

- This approach provided the best kaggle score - 98.328
- 'Targeted_hate' and 'Threatening' classes were performing better on Logistic Regression while the other classes were performing better with RIdge Classifier with hyperparameters in the image above.

Approach #2

```
model = GridSearchCV(
  RidgeClassifier(n_jobs=-1, random_state=0),
  param_grid={'solver':['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga', 'lbfgs']}
  )
model.fit(X_encoded,y[x])
d = model.decision_function(X_valid)
probs = np.exp(d) / np.sum(np.exp(d))
score = roc_auc_score(y_valid[x],probs)
d_test = model.decision_function(test_encoded)
probs_test = np.exp(d_test) / np.sum(np.exp(d_test))
predict_dic[x] = probs_test.tolist()
```

- This approach provided the second best kaggle score - 98.05
- Here all the classes were trained with Ridge Classifier with hyperparameters obtained using GridSearchCV.

Approach #3

```
model = GridSearchCV(LogisticRegression(n_jobs=-1, random_state=0), param_grid={'C':[1,3,5,10,15]}
model.fit(X_encoded,y[x])
score = roc_auc_score(y_valid[x],model.predict_proba(X_valid)[:,1])
predict_dic[x] = model.predict_proba(test_encoded)[:,1].tolist()
```

- This approach provided the third best kaggle score - 97.76
- Here all the classes were trained with Logistic Regression with hyperparameters obtained using GridSearchCV.

Other Methods and approaches.

- We tried also stemming instead of lemmatization.
- We tried using BagOfWords(Count Vectorizer) instead of TfIDF vectorizer.
- TruncatedSVD was also tried but max_features parameter of Tf-IDF vectorizer was giving better results.
- We tried UnderSampling and OverSampling methods but the models were overfitting on the data.

Other Models Tested on validation data.

- RandomForestClassifier(Default parameters - score - approx 85
- NB-SVMClassifier - score - approx 96
- MultinomialNaiveBayes - score approx 94

Required Libraries/versions.

```
print('pandas',pd.__version__)
print('numpy', np.__version__)
print('sklearn', sklearn.__version__)
print('regex',re.__version__)
print('scipy',scipy.__version__)
print('ntlk',nltk.__version__)
print('imblearn', imblearn.__version__)
print('wordcloud', wordcloud.__version__)
print('PIL',PIL.__version__)
print('seaborn',sns.__version__)
print('matplotlib', matplotlib.__version__)
```

✓ 0.3s

```
pandas 1.1.5
numpy 1.21.2
sklearn 0.24.2
regex 2.5.109
scipy 1.5.4
ntlk 3.6.5
imblearn 0.8.0
wordcloud 1.8.1
PIL 8.0.1
seaborn 0.11.0
matplotlib 3.3.3
```