SABUDH FOUNDATION

Toxic Comment Classification

Presented by Team-Quarantine

Introduction

Why do we need comment classification?

- Maintaining a decorum on an online forum is the need of the hour. Defined as “willful and repeated harm inflicted through the medium of electronic text”, cyber bullying puts targets under attack from a barrage of degrading, threatening messages conveyed using web sites, blogs, chat rooms, cell phones, web sites, e-mail.
- Toxic comment classification is about detection of different types of toxic comments by classifying them into one or more labels from ‘toxic’, ‘severe_toxic’, ‘obscene’, ‘threat’, ‘insult’ and ‘identity_hate’.

Objective

The objective is to develop pre-defined models and improve their accuracy as compared to the outputs provided by Perspective API.

EDA(Exploratory Data Analysis)

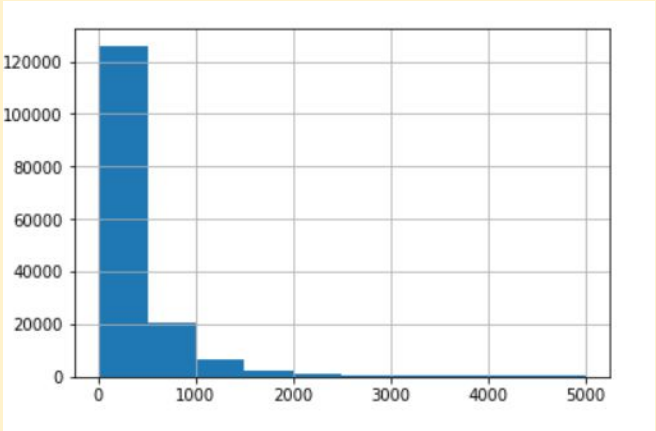
- It shows the number of clean comments with respect to total number of comments in the dataset.

```
Total comments =  159571
Total clean comments =  143346
Total tags = 35098
```

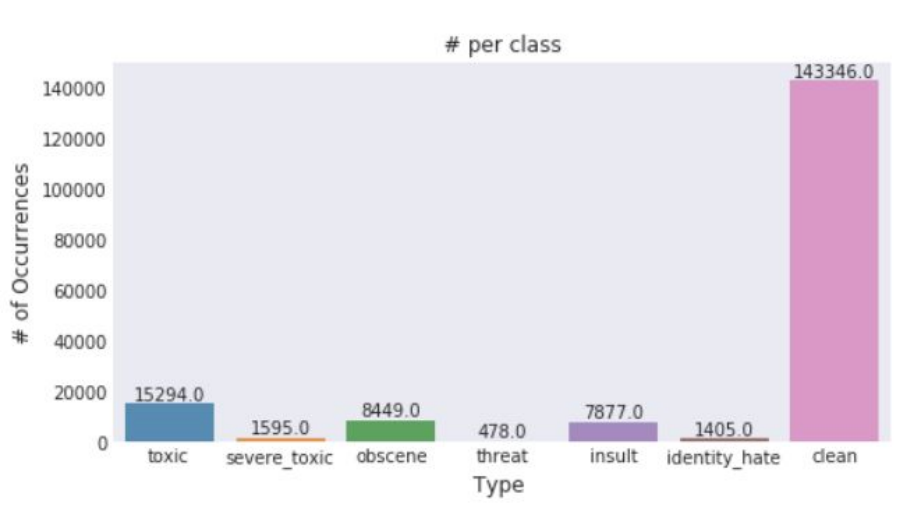
- The length of the comments varies a lot.

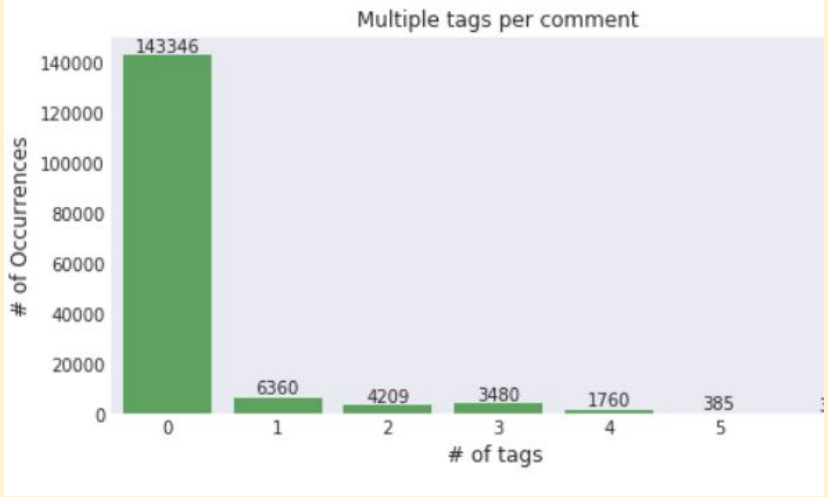
```
lens = train.comment_text.str.len()
lens.mean(), lens.std(), lens.max()
```

```
(394.0732213246768, 590.7202819048923, 5000)
```



- The plots below show number of occurrences of different types of comments and comments having multiple tags respectively, which clearly depicts imbalanced classes.





The toxicity is not evenly spread out across classes. Hence we face class imbalance problems.

SMOTE(Synthetic Minority Oversampling Technique)

Since there was a huge difference in number of non-toxic(clean) and other class(toxic,obscene etc) comments, binary classification was adopted.

```
0    143647
1     15924
Name: toxicity, dtype: int64
```

‘0’ represents clean comments

‘1’ represents toxic comments

After binary classification, SMOTE was used to solve the imbalance problem.

SMOTE aims to balance class distribution by randomly increasing minority class examples by replicating them.

Before applying SMOTE

```
Before OverSampling, counts of label '0'i.e .Clean: 143647
Before OverSampling, counts of label '1' i.e. Toxic: 15924
```

After applying SMOTE

```
After OverSampling, counts of label '0' i.e Clean: 143647
After OverSampling, counts of label '1' i.e Toxic: 143647
```

After the oversampling process, the data is reconstructed and several classification models were applied on the processed data.

Embeddings

Word embeddings are distributed representations of text in an n-dimensional space. They turn text data to vectors.

For this purpose, we considered these four word Embedding algorithms:

- Tf-idf
- GloVe
- Word2vec
- fastText

TF-IDF

Tf-idf (“Term-frequency -Inverse Data Frequency”).

Term-frequency (tf) gives us the frequency of the word in each document in the corpus. It is the ratio of number of times the word appears in a document compared to the total number of words in that document.

TF-IDF Features for first 2 documents:

	tfidf
dolls	0.267670
metallica	0.259920
vandalisms	0.257178
closure	0.231638
hardcore	0.226447
...	...
factit	0.000000
factitious	0.000000
factitude	0.000000
factiva	0.000000
єı	0.000000

189775 rows × 1 columns

	tfidf
aww	0.385230
2016	0.346801
colour	0.310714
seemingly	0.307215
matches	0.282300
...	...
factional	0.000000
factionalism	0.000000
factionists	0.000000
factions	0.000000
єı	0.000000

189775 rows × 1 columns

GloVe

GloVe or Global Vectors is a count-based model. GloVe captures both local statistics (local context information of words) and global statistics (word co-occurrence) of a corpus, in order to come up with word vectors.

The embeddings are optimized directly, so that the dot product of two word vectors equals the log of the number of times the two words will occur near each other.

The resulting embeddings show interesting linear substructures of the word in vector space.

Word2vec

word2vec trains words against other words that neighbor them in the input corpus.

It does so in one of two ways, either using context to predict a target word (a method known as continuous bag of words, or CBOW), or using a word to predict a target context, which is called skip-gram.

Word2vec and GloVe both fail to provide any vector representation for words that are not in the model dictionary. This is where fastText comes in.

fastText

fastText is another word embedding method that is an extension of the word2vec model. Instead of learning vectors for words directly, fastText represents each word as an n-gram of characters.

So, for example, take the word, “*artificial*” with n=3, the fastText representation of this word is *<ar, art, rti, tif, ifi, fic, ici, ial, al>*

Once the word has been represented using character n-grams, a skip-gram model is trained to learn the embeddings. This model is considered to be a bag of words model with a sliding window over a word because no internal structure of the word is taken into account.

fastText works well with rare words. So even if a word wasn’t seen during training, it can be broken down into n-grams to get its embeddings.

Models

Logistic Regression

Logistic regression is used widely to examine and describe the relationship between a binary response variable (e.g., ‘success’ or ‘failure’) and a set of predictor variables.

The primary objective of logistic regression is to model the mean of the response variable, given a set of predictor variables.

Synonyms for “Predictor Variables”:
Independent variables, X-variables, Input variables

Applying LR using TF- IDF

Accuracy-0.955

```
[15] from sklearn.metrics import accuracy_score
     print ("Accuracy : ", accuracy_score(y_test, y_pred))

     Accuracy :  0.9556008146639511
```

RMSE-0.044

```
[16] from sklearn.metrics import mean_squared_error
     mean_squared_error(y_test, y_pred)

     0.04439918533604888
```

Applying LR using TF- IDF without SMOTE (binary classification)

Confusion Matrix

```
[14] from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, y_pred)

     print ("Confusion Matrix : \n", cm)
```

```
Confusion Matrix :
 [[28561   141]
 [ 1276  1937]]
```

Applying LR using TF- IDF & SMOTE (binary classification)

Accuracy -0.92

```
[45] from sklearn.metrics import accuracy_score
     print ("Accuracy : ", accuracy_score(y_test_old, y_pred))

     Accuracy :  0.9276202412658624
```

RMSE -0.072

```
[48] from sklearn.metrics import mean_squared_error
     mean_squared_error(y_test_old, y_pred)

     0.07237975873413756
```

Applying LR using TF- IDF & SMOTE (binary classification)

Confusion Matrix

```
[44] from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test_old, y_pred)

     print ("Confusion Matrix : \n", cm)
```

```
Confusion Matrix :
 [[26707  1995]
 [  315  2898]]
```

Applying LR using GloVe & SMOTE

Accuracy

```
[19]: from sklearn.metrics import accuracy_score
      accuracy=accuracy_score(y_test,predicted_res)
      print(accuracy)
```

```
0.8146541999849598
```

Confusion Matrix

```
[20]: from sklearn.metrics import confusion_matrix
      confusion_matrix(y_test,predicted_res)
```

```
Out[20] array([[29369,  6574],
               [  820,  3130]])
```

SGDClassifier

This applies regular Stochastic Gradient Descent to train a linear SVM classifier. It does not converge as fast as the LinearSVC class, but it can be useful to handle huge datasets.

SGDClassifier(alpha=1/(m*C))

The C hyperparameter: a smaller C value leads to a wider street but more margin violations

Applying SGDClassifier using GloVe & SMOTE

[22]:
```
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(y_test,predicted_res)
print(accuracy)
```

```
0.7489534504800341
```

[23]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,predicted_res)
```

Out[23]
```
array([[26435,  9508],
       [  507,  3443]])
```

RNN

Model Summary-

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture[1] used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 100)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 100, 128)          2560000
_________________________________________________________________
lstm_layer (LSTM)            (None, 100, 60)           45360
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 60)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 60)                0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                3050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 2,608,461
Trainable params: 2,608,461
Non-trainable params: 0
_________________________________________________________________
```

Applying RNN without SMOTE (binary classification)

Accuracy-0.98 Val Accuracy- 0.97

```
Train on 111699 samples, validate on 47872 samples
Epoch 1/2
111699/111699 [==============================] - 285s 3ms/step - loss: 0.0748 - accuracy: 0.9724 - val_loss: 0.0730
- val_accuracy: 0.9734
Epoch 2/2
111699/111699 [==============================] - 278s 2ms/step - loss: 0.0475 - accuracy: 0.9828 - val_loss: 0.0854
- val_accuracy: 0.9693
```

With SMOTE (binary classification)

Accuracy- 0.95 (+/- 0.03) Val Accuracy-0.94 (+/- 0.03)

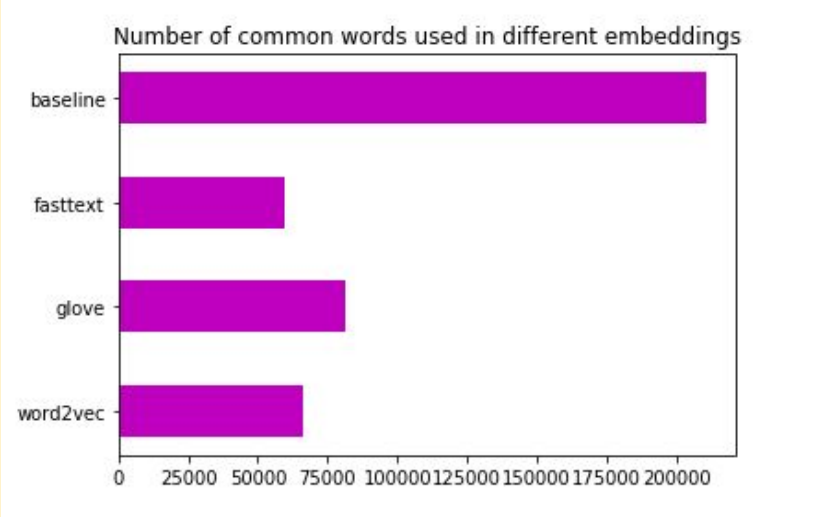
```
Train on 201105 samples, validate on 86189 samples
Epoch 1/2
201105/201105 [==============================] - 498s 2ms/step - loss: 0.2378 - accuracy: 0.8997 - val_loss: 0.1421
- val_accuracy: 0.9370
Epoch 2/2
201105/201105 [==============================] - 499s 2ms/step - loss: 0.1232 - accuracy: 0.9532 - val_loss: 0.1376
- val_accuracy: 0.9399
```

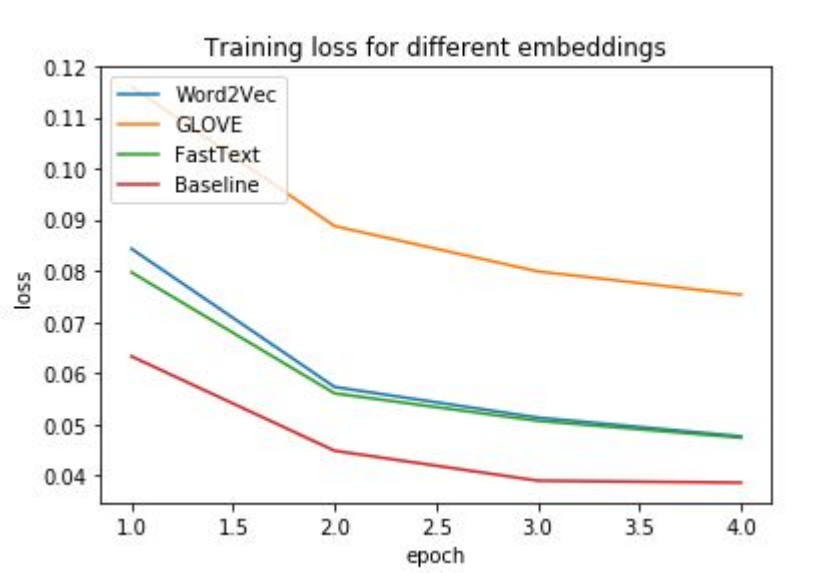
CONCLUSION

The model which seemed to perform the best-

The baseline model **RNN-LSTM** which has minimum Loss and is trained on

Word sequences.





THANK YOU

Team Members-

- Akash
- Ashita
- Jaspreet
- Jigar
- Navjot
- Nikhil
- Pulkit