Project Report:
Toxic Comment Classifier

Shaunak Phatak

Problem Context

Advancements in technology has led to a deluge of data being generated daily. For example, 2.5 quintillion bytes of data was generated daily in 2018[1]. With technologies such as 5G, IoT (Internet of Things) devices, this number will keep on increasing. The internet has enabled users around the globe to stay connected through various social media platforms such as Facebook, Twitter and many others. As an example of the scale, Facebook has around 2.4 billion users on its platform[2]. With the ability to communicate easily on such platforms, it has also led to the unfortunate rise of online abuse and cyberbullying through people commenting on such platforms. It is thus imperative to curb such behavior. Manually detecting such toxic comments is cumbersome due to the enormity of the data. Natural language processing (NLP) is a useful tool that can be employed to automate such a task.
The goal of this project is to use various techniques available in NLP to develop a toxic comment classification model.

Problem Definition

The data used for this project was provided by Conversation AI as part of a Kaggle competition[3]. It contains around 160,000 Wikipedia comments which have been manually labeled. There are six different labels which include: 'toxic', 'severe toxic', 'obscene', 'threat', 'insult' and 'identity hate'. This is a multi-label classification problem as comments can be associated with multiple labels.

The next sections will discuss steps undertaken as part of a standard NLP pipeline such as text preprocessing, word representation and machine learning modeling.

Data Wrangling and Exploratory Data Analysis (EDA)

The distribution as shown in Table 1 indicates imbalance across all labels. This problem can be examined during modeling by applying over-sampling techniques such as SMOTE. Reiterating the information in Table 1, figure 1 indicates that safe comments are the largest in number accounting for 89% of all the dataset.

Label	Toxic	Severe Toxic	Obscene	Threat	Insult	Identity Hate
0	90.42	99	94.71	99.7	95.06	99.12
1	9.58	1	5.29	0.3	4.94	0.88

Table 1: Distribution of Comment Labels

Some comments can be associated with multiple labels. This distribution can be seen in Figure 2 where the horizontal axis indicates the number of labels associated with each comment. As seen most comments have a single label followed by 3 and 2 labels respectively.

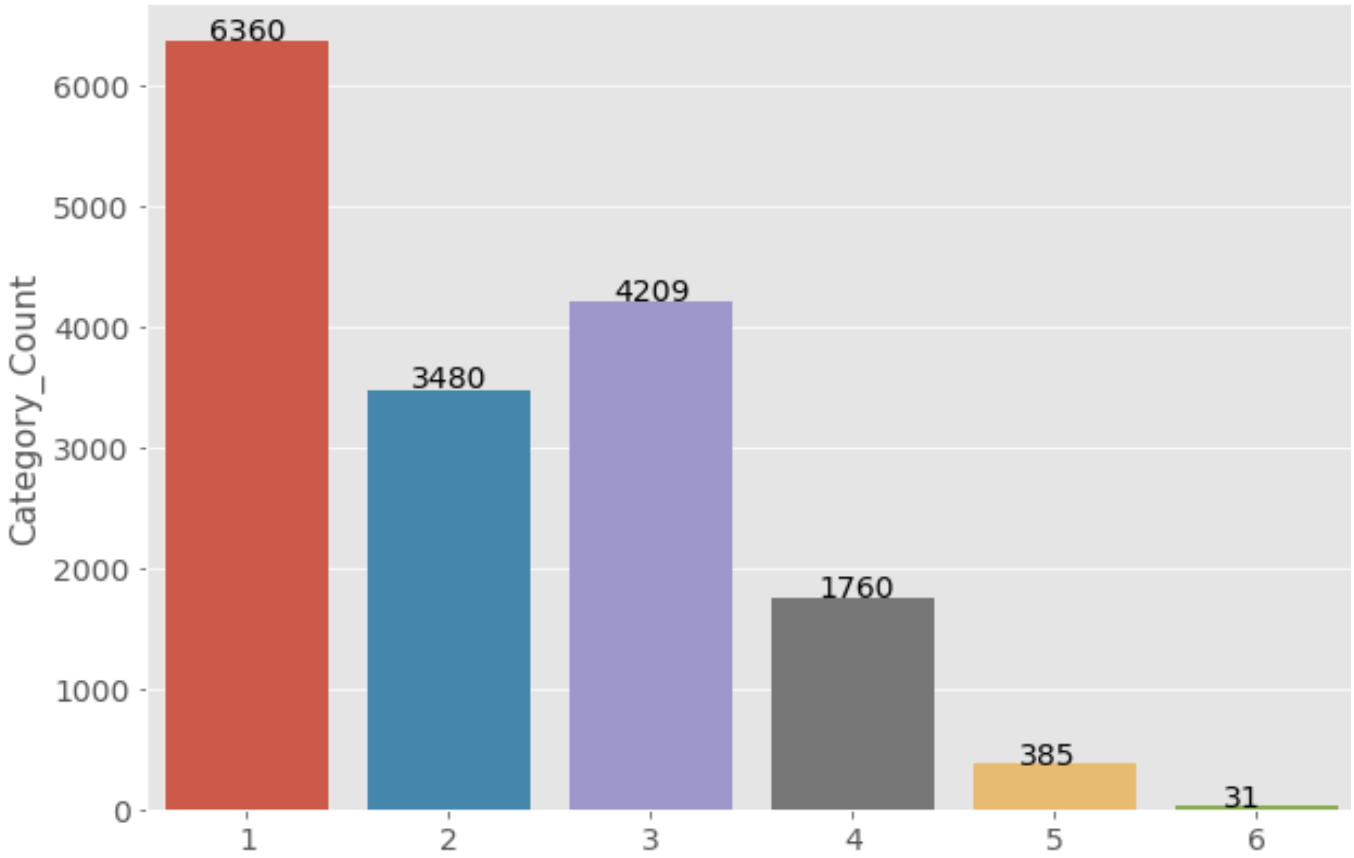

Figure 2: Multiple label counts

Apart from these categories, 143,346 are safe comments. Further, the top ten label combinations are indicated in Figure 3 below. This shows that the label 'toxic' appears in 7 of the top 10 combinations by count.

	toxic	severe_toxic	obscene	threat	insult	identity_hate	Count
0	0	0	0	0	0	0	143346
1	1	0	0	0	0	0	5666
2	1	0	1	0	1	0	3800
3	1	0	1	0	0	0	1758
4	1	0	0	0	1	0	1215
5	1	1	1	0	1	0	989
6	1	0	1	0	1	1	618
7	0	0	1	0	0	0	317
8	0	0	0	0	1	0	301
9	1	1	1	0	1	1	265

Figure 3: Comment counts for label combinations

Also, more than 90 % (100% or all comments for 'severe toxic') of comments with the other five labels have also labeled 'toxic.

Text preprocessing is an important part of an NLP pipeline to standardize text to be used for further analysis and machine learning. The steps followed to standardize comments in this project include the following:

- Convert all text to lower case
- Convert all accented characters to normal alphabets
- Remove any special characters such as emoji's, punctuations
- Remove contractions, Ex. 'Don't' → 'do not'
- Lemmatize text: The purpose of this step is to reduce inflected forms of words to their base forms, Ex. am, are, is → be
    - Stemming is a similar process to lemmatization although it may not always reduce a word to its formal base form

- Removal of stop words: These include words such as ‘a’, ‘the’, ‘an’. These words are removed as they do not help in providing any context or meaning about a comment.

For example, the following figure shows the pre and post processed form of a comment:

```
df_test.loc[469,'comment_text']
```

```
'"\nThanks! And happy new year to you too!   \'\'\'\'\'\'  Let\'s talk about it! "'
```

```
df_test.loc[469,'clean_text']
```

```
'thanks happy new year let us talk'
```

Figure 4: Example: Cleaning comments

Post text standardization, the text needs to be transformed into numeric feature vectors to be used for machine learning. For this project, the following methods were tested:

1. Frequency based

Bag of Words: This method creates 1-hot encoded vectors for each comment with the vector length being equal to the number of unique words in the comment corpus

Term Frequency Inverse Document Frequency (tf-idf)
This method does not equally weight words as done in bag of words. The term tf or term frequency is a measure of the occurrence of a word in a document. It will have higher value if the word has a higher occurrence in the document. The term idf or inverse document frequency is a measure of whether a term is unique or common across all the documents in a corpus. Its value is higher when a word appears less frequently in the corpus.

2. Word Embeddings

The above methods only rely on count of word occurrences in a document and not on their semantic similarities. Word embeddings solve this drawback by generating vectors for words where words with similar context have similar vectors quantified by a high cosine similarity. For this project, two types of word embeddings, viz. Word2Vec and GloVe were used. Word2vec is a shallow neural network model that is used to predict words given its surrounding words or vice versa. The hidden layer in these neural networks are the required word embeddings used for a downstream task like text

classification. GloVe is model that uses a large co-occurrence matrix based on the training corpus and dimensional reduction to create word embeddings. The Word2vec model used in this project was trained on the Google News dataset[4] while the GloVe model was trained on the Wikipedia 2014 and Gigaword5 dataset[5].

For word embeddings, word vectors were averaged to generate comment vectors. For the cases where words from the corpus were not available in the pretrained model's vocabulary, an averaged vector of all the word vectors in the vocabulary was used to represent such words[6].

Modeling

There are multiple approaches such as one versus rest, classifier chains and label powerset that can be used for multi label classification. For this project, a one versus rest approach was used where six independent models were trained for each label.

The modeling work involved testing the following cases:

- Frequency based methods (Bag of Words/tf-idf) with and without resampling
- Word embeddings (Word2Vec and GloVe) with and without resampling

Three models viz. Multinomial Naïve Bayes, Logistic Regression and Light Gradient Boosting were used to test these word representations.

The metrics used for comparing models were AUC and F1 scores. Apart from these, other metrics such as Hamming Loss and Exact Match or Accuracy score were also tested.

Model	Word Vector Type	Mean Train AUC	Mean Test AUC	Mean Train F1	Mean Test F1
Multinomial Naïve Bayes (MNB)	Bag of Words	0.956	0.935	0.548	0.519
	Tf-idf	0.963	0.951	0.4	0.365
Logistic Regression	Bag of Words	0.987	0.94	0.625	0.487
	Tf-idf	0.988	0.978	0.522	0.509
Light Gradient Boost (LGBM)	Bag of Words	0.99	0.965	0.74	0.535
	Tf-idf	0.993	0.968	0.789	0.533

Table 2: Modeling Performance using bag of words and tf-idf without resampling

Table 2 shows results obtained using frequency methods without resampling. Some observations are discussed as follows:

- Logistic Regression and Light Gradient Boosting out-performed Multinomial Naïve Bayes
- Bag of words although with similar AUC scores to tf-idf, had slightly better F1 scores for MNB and LGBM.
- Although it had a slightly lower F1 score than LGBM, Logistic Regression with a tf-idf representation was the better model due to less overfitting indicated by the lower differences between train and test F1-scores compared to the LGBM model

For MNB, a multinomial distribution requires integer counts. However, tf-idf as seen in the results also produces similar results despite not satisfying this requirement[7].

Next, all above configurations were run with resampling. SMOTE (Synthetic Minority Oversampling Technique) was used to oversample the positive (minority) classes. The minority classes were oversampled to 60% of the majority class. As seen in Table 3, all models exhibited overfitting seen by the differences between their train and test AUC and F1 scores.
Varying the values of sampling fractions for SMOTE also resulted in similar behavior as seen in Table 3.

Model	Word Vector Type	Mean Train AUC	Mean Test AUC	Mean Train F1	Mean Test F1
Multinomial Naïve Bayes (MNB)	Bag of Words	0.98	0.874	0.827	0.466
	Tf-idf	0.978	0.958	0.879	0.422
Logistic Regression	Bag of Words	0.989	0.881	0.95	0.35
	Tf-idf	0.989	0.963	0.954	0.48
Light Gradient Boost (LGBM)	Bag of Words	0.982	0.919	0.915	0.326
	Tf-idf	0.992	0.969	0.95	0.346

Table 3: Modeling Performance using bag of words and tf-idf with resampling

Thus, out of these methods, a Logistic Regression model with tf-idf representation and no resampling showed optimal performance. Next, hyper-parameter tuning was applied using grid search and varying parameters viz., the regularization term "C" and type of solver. The optimized models produced results similar to the values seen in Table 2.

For the word embedding based feature representations, table 4 shows the results for Logistic Regression and LGBM without resampling.

Model	Word Vector Type	Mean Train AUC	Mean Test AUC	Mean Train F1	Mean Test F1
Logistic Regression	Word2Vec	0.967	0.963	0.417	0.411
	GloVe	0.963	0.96	0.418	0.419
Light Gradient Boost	Word2Vec	0.989	0.941	0.858	0.458
	GloVe	0.993	0.953	0.868	0.452

Table 4: Modeling Performance using word embeddings without resampling

Both word embeddings showed similar results for all the tested models. Also, as seen earlier, LGBM exhibited overfitting indicated by differences in the values of AUC and F1 scores for train and test datasets. Logistic regression performed better than Light Gradient Boost without significant overfitting. Comparing with frequency-based models seen before, the word embedding models had slightly lower F1 scores but similar AUC scores. Using word embeddings did not appear to significantly boost performance.

Similar to before, using resampling with word embeddings also caused overfitting problems across both the models as seen from the results in Table 5. Also, hyper-parameters to tune the Logistic Regression model similar to the previous models did not yield significant improvements.

Model	Word Vector Type	Mean Train AUC	Mean Test AUC	Mean Train F1	Mean Test F1
Logistic Regression	Word2Vec	0.973	0.964	0.854	0.43
	GloVe	0.969	0.961	0.84	0.414
Light Gradient Boost	Word2Vec	0.995	0.967	0.955	0.54
	GloVe	0.994	0.964	0.95	0.534

Table 5: Modeling Performance using word embeddings with resampling (SMOTE)

Investigating the reasons for lack of performance improvements with word embeddings, it was observed that ~65% of the unique words in the comment corpus was not present in the pretrained Word2Vec model vocabulary while the number was ~47% for GloVe embeddings. This lack of information and substitution with an average vector may have been a possible reason for the lack of improvements over frequency-based methods.

Conclusions

This project aimed at building multi-label classification models to classify comments into six categories of toxicity.
Based on various configurations tested, it was observed that Logistic Regression with tf-idf representation performed the best across all models in terms of mean AUC and mean F1 scores.
Resampling to help with the imbalanced data did not help improving performance and caused overfitting across all models.
Word embeddings slightly underperformed frequency-based methods one of the possible reasons for this being that a large number of unique words in the comment corpus were not available in the pretrained models checked.

Future Work

Some of the methods that could be utilized to extend the project and improve model performance are discussed below:
The comment corpus contains words with incorrect spellings as well as informal words and phrases which may not have been captured by the pretrained word embeddings. One potential method to test would be to train the comments words on a sub-word vector model like FastText. Another method could to apply advanced techniques such as Deep Learning. Also, other modeling techniques such as label powerset or classifier chains can also be tested which consider correlations between the labels.

References

[1]https://seedscientific.com/how-much-data-is-created-every-day/
[2]https://ourworldindata.org/rise-of-social-media
[3]https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
[4]https://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/
[5]https://nlp.stanford.edu/projects/glove/
[6]https://stackoverflow.com/questions/49239941/what-is-unk-in-the-pretrained-glove-vector-files-e-g-glove-6b-50d-txt
[7]https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
[8] https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72
[9] https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff