Categorizing Inappropriate Texts

Aditi Gupta
aditi19292@iiitd.ac.in

Mansi Singhal
mansi19370@iiitd.ac.in

Nimisha Gupta
nimisha19315@iiitd.ac.in

Abstract

We are building a multi-label classification model that will detect inappropriate texts and help us to further categorize them. We are using a kaggle dataset which contains Wikipedia comments, and we employed NLP techniques to extract features from this dataset. We have used Naive Bayes and Logistic Regression as our baseline models and SVM, Random Forest and Neural Networks as our advance models. Out of these, tuned Neural Network with feature engineering, text cleaning and oversampling gave the best results for us with a score of 0.7 (Using 60% Recall + 40% Precision).

1. Introduction

Conversations that take place over online platforms can become inappropriate and discourteous, degrading the quality of online conversations and having serious consequences like trolling, cyberbullying, and accessing adult explicit content. To avoid such situations, we want to build a **multi-label classification model that will detect inappropriate texts** and help us to further categorize them.

2. Related Work

We are dealing with text classification and currently Linear Support Vector Machine and Deep Learning algorithms are among the famous best algorithms explored for these problems. We evaluated some of the best performing inappropriate text classification models on our dataset using our evaluation metric. The highest score among these models is coming around *0.71*. Most of these approaches have used neural networks like WordBatch and FM_FTRL [1, 4] to solve the classification problem. The best performing model [2] (state-of-the-art) has final score *0.8072* and used Support Vector Machine with Naive Bayes Features.

3. Dataset Evaluation

We are using a Kaggle dataset [3] containing Wikipedia comments labeled by human raters for inappropriate behaviour. The dataset [1] has 6 labels: 'toxic', 'severe toxic', 'obscene', 'threat', 'insult' and 'identity hate'. We have a total of 2,23,286 data points in our dataset and we have divided it into: 80% training, 10% validation and 10% testing set. Thus the training set has 1,78,626 samples and the validation and testing set have 22,329 samples each.

3.1. Feature Extraction

As we have a natural language processing problem, we have first performed text cleaning which involves the following:

- Converting to Lowercase
- Removing Special Characters
- Removing Numbers
- Removing Stop Words
- Replacing Contractions with their Full Forms
- Lemmitization

For factorizing the text data we have used 3 feature extraction techniques:

- *Bag of Word*: Bag of words describes occurrence of words within a document and creates a vocabulary of unique words and represents a sentence as a vector with length of vocabulary with true value showing presence of the word.
- *TF-IDF*: TF-IDF stands for Term Frequency — Inverse Document Frequency. Term Frequency: is a scoring of the frequency of the word in the current document. Inverse Document Frequency: is a scoring of

[1] Dataset link: https://drive.google.com/drive/u/1/folders/1yY_GmS89Rvm_YMB6TqNr3hZwrqVFmNvh

how rare the word is across documents. The multiplication of these two scores gives us the final score which reflects how important a word is in a collection.

- *Word2vec*: Word2vec learns meaningful relations between words and encodes them into vector. Its ojective is to have words with similar context occupy close spatial positions, so words like great and good will have vectors lying close to each other. After calculating vectors for each word, we average them and generate a vector for the whole text. **Word2Vec gave the best results with all the models**.

Results for Naive Bayes with all feature extraction techniques are shown in figure 1.

3.2. Feature Engineering

In addition to the words, some attributes of the comments can show contrast between toxic and clean comments like length of the comment and individual words, number of question marks, exclamation marks and capital letters in the comment. These were added as features as they might be useful in prediction whether a comment is toxic or not.

3.3. Evaluation Metric

For our dataset we have chosen the evalutaion metric to be sum of 60% recall value and 40% precision. We have chosen this evaluation metric since the cost of the model classifying a decent text as obscene (false positive) is far less than the cost of it missing an obscene text (false negative).

4. Methodology

To meet our end goal of building an inappropriate text classifier, we tried multiple varieties of models and tuned them to obtain best results. The two baseline models we used were - *Logistic Regression and Naive bayes*. The advanced models we used were- *SVM, Random Forest and Neural Network*. We have also made use of K-cold cross validation to get accurate scores.

4.1. Exploratory Data Analysis

In our dataset, 22,468 out of 2,23,549 comments are classified as toxic i.e. 10.05%. These inappropriate comments contain 21,384 toxic comments, 1,962 severe_toxic comments, 12,140 obscene comments, 689 threat comments, 11,304 insult comments and 2,117 identity_hate comments as shown in figure 2. To understand the correlation and overlapping between these labels we plotted a correlation matrix and graph showing number of comments with multiple labels. The correlation matrix shown in figure 3 shows that there is a high overlap between 'toxic', 'obscene', and 'insult' comments. Since there are not a lot of data points for 'Severe_toxic', identity_hate' and 'threat', they don't really overlap much with any other labels.

The graph in figure 4 shows that most of the comments present in the dataset are clean and that comments with more than one toxicity labels decreases with the increase in the number of multiple labels. To understand what comments of various categories look like we used a tokenizer with standard stopwords to count the overall frequency of various words and plotted the ones with highest counts. figure 5 shows word frequencies of clean comments and figure 6 shows the word frequencies of comments under the threat category.

4.2. Preprocessing

After doing EDA, preprocessing using text cleaning, feature extraction and feature engineering techniques discussed in Section 3 were performed. Since we had a multi label classification problem in hand and most traditional learning algorithms are made for single label classification, we used the following multi-label classification techniques to transform our multi-label problem into multiple single-label problems so that the existing algorithms can be used:

- *Binary Relevance*: In Binary Relevance, each label is treated as a single class classification problem. Classifiers are run on each (X, y(i)) where i ranges for all labels. The drawback of binary relevance is that it does not consider correlations between each label and treat every label as separate.

- *Classifier Chains*: In this a chain of classifiers is formed where the subsequent classifier takes into account the labels generated by the previous model. The first classifier takes in the feature set (X) and predicts the result for y1 (First label). The next classifier adds y1 in the feature set and then predicts the result for y2. This chain continues on for all the labels and the total number of classifiers trained are equal to the number of labels. This technique takes into account label correlations.

- *Label Powerset*: In the Label power set, a unique class is assigned to every label combination possible. For example, labels 00, 01, 10, 11, in y1 and y2 respectively, it will generate four classes - 1, 2, 3, 4. And replace all the instances of label combination accordingly. Then a multi-class classifier is used to predict the results.

Most of our models gave best results with Classifier Chains, this may be because our labels are correlated with each other like for example severe_toxic is highly correlated to toxic label.

4.3. Various Classifiers

1. **Naive Bayes**: Naive Bayes classifier is useful when dealing with classification problems and when we have problems involving two or even multiple classes (in our case we have a multi-label dataset). It is based on the concept of the Bays theorem which a fundamental theorem in probability. This classifier assumes that every feature of the dataset is independent and gives predictions accordingly.

   Classifier Chain showed the best results for Naive bayes as shown in figure 7 . To find the best parameters of all models, we plotted graphs of the important parameters of all models and saw how different parameters impact our models. In case of Naive Bayes the best value for var smoothing is 0.1, As shown in figure 8. We got the same result through Grid Search too. The final score of Naive Bayes was 0.5462.

2. **Logistic Regression** Logistic Regression is a machine learning model which is used for classification problems, especially binary classifications. It uses the sigmoid function to categorize the features generated. It formulates a real number between 0 and 1 (since the range of a logistic regression is from 0 to 1).

   Classifier Chain showed the best results for Logistic Regression as shown in figure 9. For Logistic Regression, the best values are solver= newton-cg, C=0 and max_iter =45 as shown in figure 10. The final score of Logistic Regression was 0.5889.

3. **Random Forest** It is a machine learning algorithm which classifies data by building decision trees and giving predictions based on the trees generated. It is useful for regression as well as classification problems.

   Binary Relevance showed the best results with Random Forest as shown in figure 11. For Random Forest, the best parameters are criterion = 'entropy', n_estimators = 25, max_features = 'auto', max_depth = 12 as shown in figure 12. The performance of evaluation metric remains constant after n_estimators becomes 25. For max_depth, the test set stops improving performance after 12 so that is the best value. The evaluation metric of Tuned Random Forest did not increase much but the variance decreased drastically from 0.42 to 0.14. The final score was 0.6162.

4. **SVM** Support Vector Machine is a strong ML algorithm and like random forest, it can also be used for regression as well as classification problems. It is mainly used for classification since the idea behind SVM is finding a hyperplane in an n-dimensional plane that can clearly categorize our data points.

   For SVM, both classifier chains and binary relevance showed similar results around 0.63 as shown in figure 13 .The final score was 0.6391. SVM hyperparameter tuning is could not be performed as it requires High RAM access.

5. **Neural Network** Neural Network is another very powerful Machine learning algorithm which consists of a series of algorithms which mimic the behaviour of an animal/human brain. It tries to identify the relationships between different aspects of our dataset using computing systems with interconnected nodes spread over a number of layers.

   We have implemented a basic Neural Network model with one hidden layer. It is a fully connected neural network which uses Sigmoid activation function. The score for Neural Network prior to hyperparameter tuning was 0.6514, then we used keras-tuner to find the best parameters for neural network as shown in figure 14 The score for **tuned Neural network** model was **0.7032**. This is the **best performing model** in our project.

4.4. Error Analysis

To check whether the training of our models is being done correctly or not and to make sure the model is not overfitting or underfitting, we have calculated scores for individual labels and generated learning curve and calculated variance for our best model (Neural Network).

Our model showed good scores for 'toxic', 'obscene' and 'insult' labels and comparatively lower scores for 'severe_toxic', 'threat' and 'identity_hate'. This was because the data points for the latter three are extremely less in number, thus our model is not trained properly for them. To solve this problem, we tried oversampling our data to increase the data points for these 3 labels. We chose oversampling over undersampling because the total inappropriate data points are less. We saw a significant improvement in the score of two labels- 'severe_toxic' and 'identity_hate' as shown in figure 15. The variance for our model came out to be 0.26.

The learning curve shown in figure 16 show that for a small dataset the model was overfitting but as the dataset size increased the model was fitting well to the data. This helps us conclude that the size of our dataset is big enough to provide stable training.

5. Result and Analysis

The model which has given the best result so far has been tuned Neural Network model with feature engineering, text cleaning, and oversampling, with a score of 0.7032. The Ablation analysis for the best model are shown in the table below. We can see from the scores given in the table that

text cleaning and tuning of the Neural Network model have majorly contributed in the improvement in performance.

Component	Final Score
Overall System	0.7032
Text Cleaning	0.6073
Feature engineering	0.7008
Oversampling	0.6792
Without tuning	0.6514

Table 1. Ablation Analysis

There is a significant gap between our results and the results of the state of the art. One possible reason for this can be that while we are using Neural Networks and the state of the art approach is using SVM with Naive Bayes features. Another difference is that while the state of the art approach has used TF-IDF method for feature extraction, we have used word2vec since that was giving us better results. There are also some differences in the text cleaning that we have employed and the state of art has used. A lot of our codes required high computational power which we didn't have and hence we couldn't generate those results. These include - Tuning SVM, Learning curves for SVM and Random Forest, trying BERT as a feature extraction method.

Some possible improvements that we could have made in our model are:

1. Taking into account the ordering of labels identified by the correlation matrix for the models using Classifier chain as it takes into consideration the label ordering, this could have given better results.

2. Identifying the least baseline score using random classifier could have given us a lower bound and thus helped in identifying underfitting of our models.

6. Contributions

Our individual contributions are as follows:

1. Aditi: Random Forest, Word2Vec, Bag of words, Tuning of Random Forest and baseline models, Text Cleaning, Learning curves, compared all models with all multi-label classification techniques.

2. Mansi: SVM, Tuning of Neural Network and SVM, TF-IDF, EDA, Text Cleaning, Tuning of baseline models, state-of-the-art score research and score calculation

3. Nimisha: Text cleaning, Error analysis, EDA, Logistic regression, Naive Bayes, tuning of SVM.

References

[1] Fork of blend it al + ensamble. https://www.kaggle.com/code/maksimovka/fork-of-blend-it-al-ensamble/script. 1

[2] Toxic comment classification. https://github.com/jayspeidell/ToxicCommentClassification-. 1

[3] Toxic comment classification challenge. https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data. 1

[4] Wordbatch fm$_ftrlusingmselb$0.9804.. 1

Figure 1. Results for Naive Bayes with all 3 Feature Extraction techniques

```
Model: Naive Bayes with Classifier Chain
Feature extraction method: Bag of Words
Recall: 0.8584529505582137
Precision 0.07746276174713967   (Too Low)

Model: Naive Bayes with Classifier Chain
Feature extraction method: TF-IDF
Recall: 0.8580542264752791
Precision 0.07754396079561833   (Too Low)

Model: Naive Bayes with Classifier Chain
Feature extraction method: Word2Vec
Recall: 0.7741228070175439
Precision 0.22799600728084082
```

Figure 2. Number of comments under each label

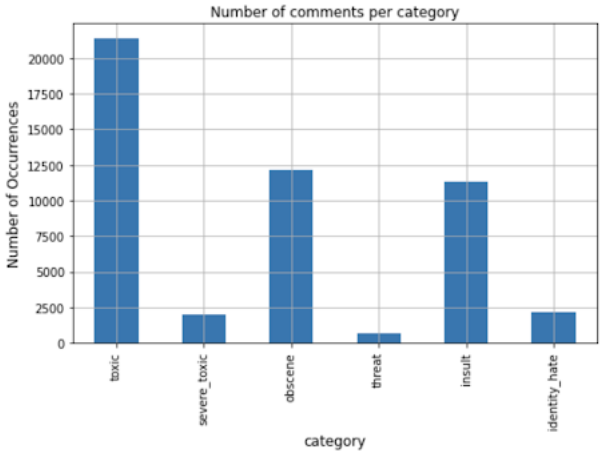


Figure 3. Correlation Matrix of all labels

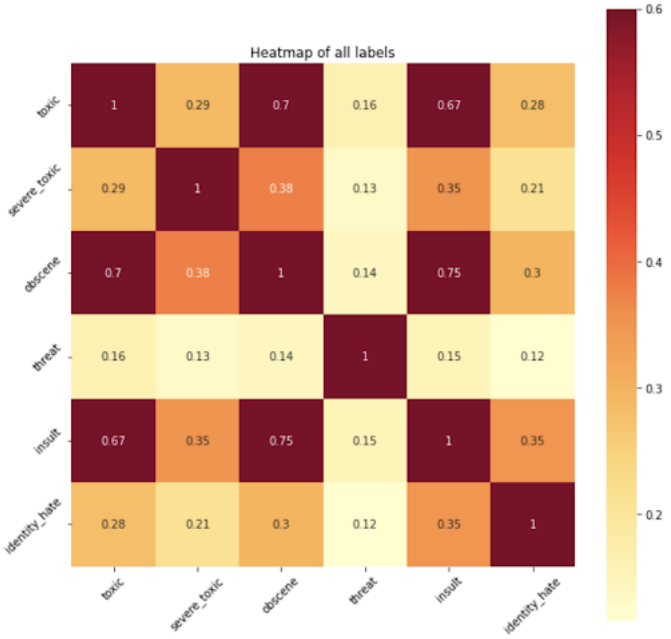


Figure 4. Number of comments with multiple labels

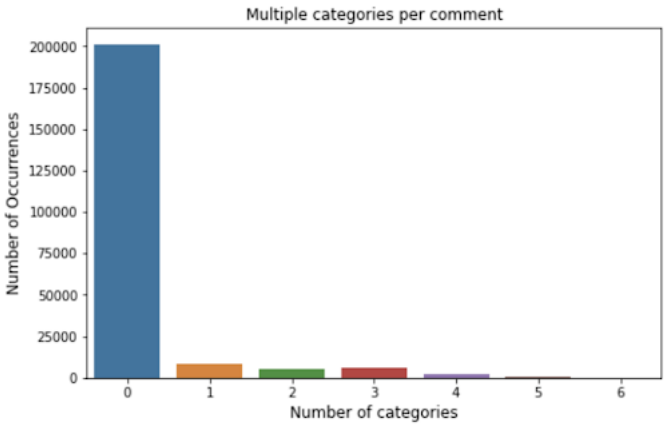


Figure 5. Word Frequency of Clean comments

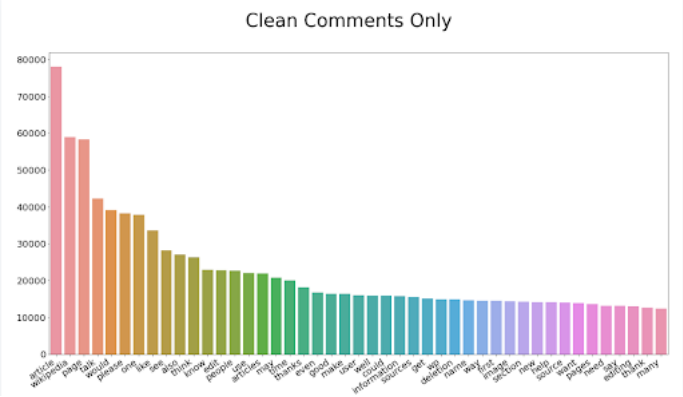


Figure 6. Word frequency of comments under the 'threat' category

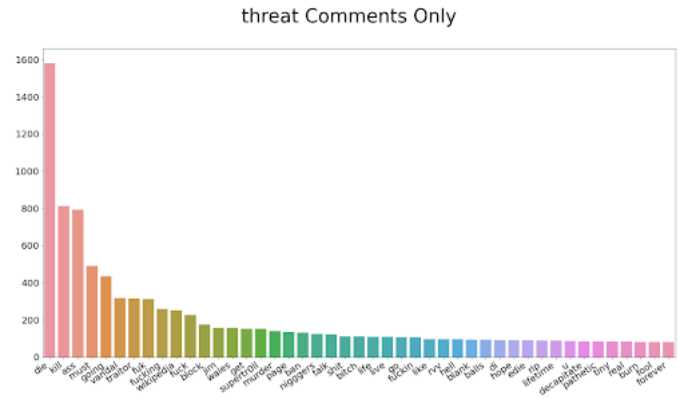


Figure 7. Performance of various Multi-label output techniques with Naive Bayes

```
Model: Naive Bayes with Classifier Chain
Feature extraction method: Word2Vec
Precision:  0.2109003381282595
Recall:  0.7760438633487979
Final score of the model:  0.5499864532605825
0.5499864532605825
```

```
Model: Naive Bayes with Classifier Chain
Feature extraction method: Word2Vec
Precision:  0.2109003381282595
Recall:  0.7760438633487979
Final score of the model:  0.5499864532605825
0.5499864532605825
```

```
Model: Naive Bayes with Label Powerset
Feature extraction method: Word2Vec
Precision:  0.15967219960329887
Recall:  0.6450864614086883
Final score of the model:  0.45092075668653253
0.45092075668653253
```

Figure 8. Hyperparameter tuning for Naive Bayes

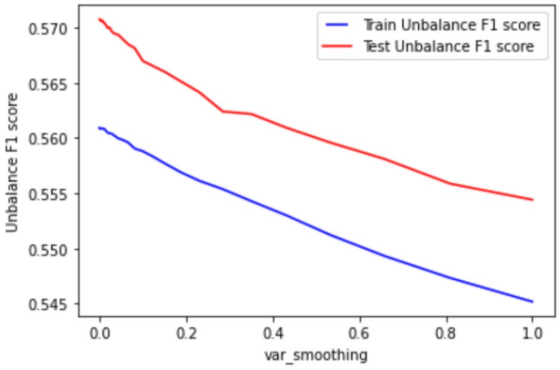


Figure 9. Performance of various Multi-label output techniques with Logistic Regression

```
Model: Logistic Regression with Binary Relevance
Feature extraction method: Word2Vec
Precision:  0.7482344632768362
Recall:  0.4468578658793758
Final score of the model:  0.5674085048383599
```

```
Model: Logistic Regression with Label Powerset
Feature extraction method: Word2Vec
Precision:  0.755637707948244
Recall:  0.43104175453395194
Final score of the model:  0.5608801358996688
```

```
Model: Logistic Regression with Classifier Chain
Feature extraction method: Word2Vec
Precision:  0.7055053888727061
Recall:  0.5107549557148883
Final score of the model:  0.5886551289780154
```

Figure 10. Hyperparameter tuning for Logistic Regression

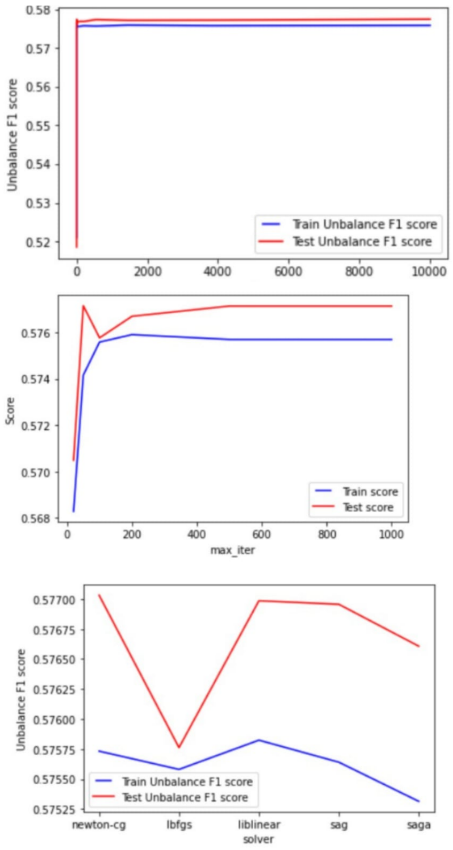


Figure 11. Performance of various Multi-label output techniques with Random Forest

```
Model: Random Forest with Classifier Chains
Feature extraction method: Word2Vec
Precision:  0.8054620276842499
Recall:  0.4312036851592229
Final score of the model:  0.5809070221692337

Model: Random Forest with Binary Relevance
Feature extraction method: Word2Vec
Precision:  0.8517975055025678
Recall:  0.45925632911392406
Final score of the model:  0.6162727996693815

Model: Random Forest with Label Powerset
Feature extraction method: Word2Vec
Precision:  0.8438864628820961
Recall:  0.31409995936611135
Final score of the model:  0.5260145607725053
```

Figure 12. Hyperparameter tuning for Random Forest

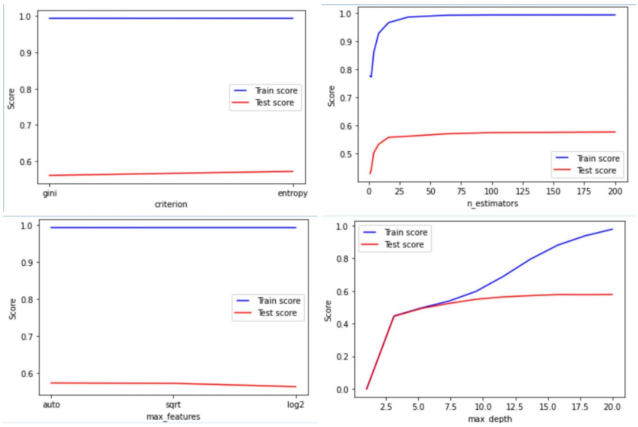


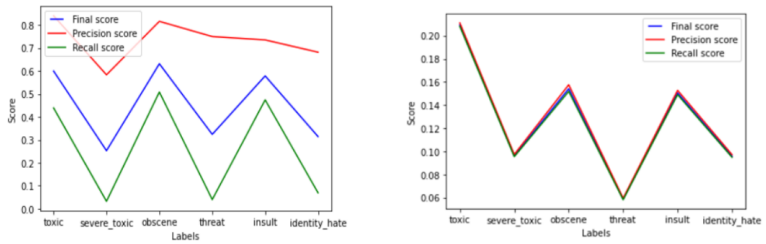
Figure 13. Performance of various Multi-label output techniques with SVM

```
Model: Support Vector Machine with Classifier Chains
Feature extraction method: Word2Vec
Precision:  0.8015776699029126
Recall:  0.5308418726140245
Final score of the model:  0.6391361915295797

Model: SVM with Binary Relevance
Feature extraction method: Word2Vec
Precision:  0.8397565922920892
Recall:  0.5046728971962616
Final score of the model:  0.6387063752345927

Model: SVM with label powerset
Feature extraction method: Word2Vec
Precision:  0.8517975055025678
Recall:  0.45925632911392406
Final score of the model:  0.6162727996693815
```

Figure 14. Hyperparameter tuning of Neural Network

Value	Best Value So Far	Hyperparameter
32	160	units
sigmoid	sigmoid	activation
False	False	dropout
0.001871	0.00058807	lr

Figure 15. Scores before and after oversampling



Before oversampling

After oversampling

Figure 16. Learning curve for Neural Network