

CAPSTONE PROJECT

Machine Learning Engineer Nanodegree

Ajay Byanjankar

March 22, 2021

1 PROBLEM STATEMENT

Natural language processing has been a growing field of research with text data being available everywhere, such as social media, chats, emails, web pages, and survey responses. These text data, however, are unstructured and it can be difficult and time consuming to extract information from them. As such, with proper technical and analytical tools, text data can be a source of rich information. Natural language processing has been successfully applied in many fields to extract insights from text data to accomplish business problems. Text classification is one of the major problem being solved using natural language processing, where raw text data are assigned labels based on its content. Some of the most used cases of text classification are:

- Spam detection in emails
- Sentiment classification
 - Analyze tweets or comments in social media to determine if customers are talking positive or negative about a brand or a topic.”
 - Analyze customer feedbacks to from survey results and reviews to understand their like or dislike for the product.
 - Profanity and Abuse detecting in social media platform
 - Analyzing support tickets in assigning tags to common questions for quick customer service.

The problem proposed for the capstone project is the **‘Toxic Comment Classification’** posted on Kaggle. With the growth in use of online platforms, there has also been rise in threat of abuse and harassment online. This results in many people stopping themselves from expressing themselves and give up on seeking different opinions. In addition, online platforms struggle to effectively facilitate the conversations, and are limited or completely shut down user comments. Therefore, the main focus of the problem is to identify the level of toxicity in the comments, such as threats, obscenity, insults and identity-based hate. In addition, the challenge is to build multi-headed model, meaning a multi-label task, where a single comment can be classified to multiple labels. With the help if identifying the toxic labels in the comments, it can help in making online discussion more productive and respectful.

As a multi-label classification problem, it differs from multi-class classification problem in the sense that in multi-label an example or an instance can be associated with multiple class labels simultaneously, while in multiclass problem, an example can belong to only a single class. Here, in this problem, there are six labels of toxicity and a comment can contain either no any label (meaning it is clean comment) or one or more than one labels. Hence, the main task here is to create a text classification model, which should be able to assign all the possible toxic labels to a raw comment.

2 MODELING APPROACH


2.1 Problem Transformation

Solving a multi-label classification problem needs to be tackled differently than a multi-class problem. There are multiple approaches of tackling a multi-label classification problem, of which problem transformation approach is a commonly used approach. It transforms a multi-label classification task to a number of smaller classification tasks. Two of the most commonly used problem transformation approaches are Binary Relevance and Chain Classifier. With these approaches used for problem transformation, it is used with suitable machine learning algorithms to tackle the classification tasks. Here, Logistic Regression and Naïve Bayes will be used as the machine learning models for the classification tasks, which will set a benchmark performance. This benchmark performance will be compared with more advanced Deep learning models.

2.1.1 Binary Relevance

Binary Relevance is one of the simplest approach to solving a multi-label classification problem. Similar to one-vs-rest approach for multi-class problems, it transforms the multi-label task to a number of independent binary classification tasks (one for each labels), where predicting each label is considered to be a separate task. However, as it treats the classification tasks independently, it fails to consider the association between the labels and is its major drawback. But, it can be a helpful start to approaching multi-label classification problem and also set the benchmark performance. The transformation of multi-label classification problem to independent binary classification problem with Binary Relevance can be seen in figure below.

X	Y1	Y2	Y3	Y4
X_1	1	0	1	0
X_2	0	0	0	1
X_3	1	0	0	0
X_4	0	1	1	1



X	Y1
X_1	1
X_2	0
X_3	1
X_4	0

X	Y2
X_1	0
X_2	0
X_3	0
X_4	1

X	Y3
X_1	1
X_2	0
X_3	0
X_4	1

X	Y4
X_1	0
X_2	1
X_3	0
X_4	1

Figure 1 Binary Relevance Transformation

2.1.2 Classifier chain

Similar to Binary Relevance, classifier chain is a problem transformation method for multi-label classification. It tries to improve performance over Binary Relevance by taking advantages of labels associations, which is ignored in Binary Relevance. Classifier chain method generates a chain of binary classifiers. In doing so, it adds an additional input feature in the classifier as an output from the previous classifier in the chain. This approach allows the classifiers to take into account the label association present, helping in improving the classification performance. The multi-label problem transformation with classifier chain is shown below.

X	Y1
X_1	1
X_2	0
X_3	1
X_4	0

Classifier 1

X	Y1	Y2
X_1	1	0
X_2	0	0
X_3	1	0
X_4	0	1

Classifier 2

X	Y1	Y2	Y3
X_1	1	0	1
X_2	0	0	0
X_3	1	0	0
X_4	0	1	1

Classifier 3

X	Y1	Y2	Y3	Y4
X_1	1	0	1	0
X_2	0	0	0	1
X_3	1	0	0	0
X_4	0	1	1	1

Classifier 4

Figure 2 Classifier chain transformation

2.2 Deep Learning Methods

2.2.1 LSTM

LSTM (long short-term memory) is a type of recurrent neural networks (RNN). It is mostly used in problems related to sequence predictions as it can handle the order dependence in the sequence. LSTM is being widely used for text classification problem, as it helps in maintaining the order of words in the text that can preserve the semantic meaning of the text.

2.2.2 Transfer Learning with BERT

Transfer learning is a popular approach in deep learning, where a pre-trained model is used as the starting point for training a new model in similar task. It is getting popular in the field of computer vision and natural language processing as it saves huge computational and time resources required for modeling on such tasks. In addition, with knowledge gained from pre-trained models, transfer learning helps in generalization and performance improvement. With advancements in researches in Natural language processing, there has been many sophisticated techniques to address the problem. BERT (Bidirectional Encoder Representations from Transformers) is one of them, which has been producing state of the art results in problems related to natural language processing.

3 Evaluation Metrics

There are many evaluation metrics specially designed for multi-label classification tasks, such as hamming score, log loss and subset accuracy. Since, the classification problem has been taken from Kaggle competition, there is already a specific evaluation metric assigned to evaluate the performance. Hence, this problem will use the same evaluation metric, so that it becomes easy to compare our results against the Kaggle score. As per the Kaggle Competition, the evaluation metric that will be used for the evaluation is mean column-wise ROC AUC. It represents the average of the individual AUC scores of each predicted column (a label is represented as a column). So, it will be used as the main evaluation metric. Using the same approach, accuracy and log loss will also be calculated for additional evaluation metrics.

ROC AUC: AUC stands for area under the ROC curve, where ROC is the Receiver Operator Characteristics that is the graphical representation of performance of a binary classification model. It is created by plotting False Positive Rate against the True Positive Rate at different threshold values. AUC is simply a numerical score that summarizes the ROC curve, to evaluate the performance of the classifier. Its value ranges between 0 and 1, where 1 is the perfect score and score of 0.5 indicates the model being no good than a random guessing. As stated above, we will calculate AUC score individually for each labels and average them.

Accuracy: Accuracy simply represents the number of correct classifications made out of the total samples. It can be calculated as number of correct predictions divided by the total data samples. But, for multi-label classification, accuracy can be calculated as subset accuracy, also referred as Exact Match Ratio. This is considered to be a harsh metric, where the predicted set of labels for an example should exactly match the actual labels. However, for this problem, the Kaggle approach to calculating mean ROC AUC will be applied, where individual accuracy scores for each of the labels will be calculated and take the average of them as the final score.

Log loss: Log loss, also known as cross entropy, is the classification evaluation metric based on probabilities. It evaluates the classification performance by comparing the actual labels with its predicted probabilities, where it penalizes the predictions that is far from the actual labels. Its value increases when the predicted probabilities are far from actual labels, and lower value indicates good performance of the classifier. Mathematically, for a binary classification it can be calculated as:

$$\text{logloss} = -(y\log(p)+(1-y)\log(1-p))$$

4 Exploratory Analysis

As stated above, the data for the problem is extracted from Kaggle Competition that can be downloaded from [here](#). The main data used for modeling is extracted from 'train.csv' file, while 'test.csv' contains only comments for which the labels need to be predicted. The results predicted for the data from 'tes.csv' file will be used to get the final score at Kaggle.

As an input, the train data contains raw text of comments posted on social media and six of the labels as the output represented as binary labels(1,0), indicating either presence of the label for the comment or no presence. A quick look at the data is seen in the picture below:

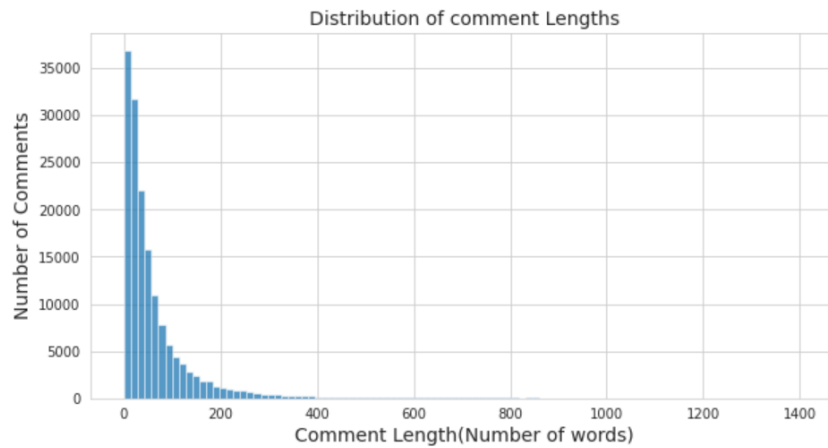
	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

Figure 3 First 5 rows of the data

The data all together has 159571 rows and 8 columns. The 'id' columns is just an identifier for the comments. Column 'comment_text' represents the raw comments and columns from 'toxic' to 'identity_hate' are the six toxic labels. As a first step, checking for any missing values in the data, there are no any missing data found.

4.1 Length of the comments

To get understanding of how long are the comments, we can first see the distribution for the length of the comments. As a measure for the length, we can use word count in the comments. With a rough estimation of word counts in the comments through word split (using space as separator), we see that majority of the comments have text length up to 200 words. There are few comments that have very long lengths.



4.2 Distribution of Labels

To get an idea of the most occurring labels, the counts of each label can be calculated. The count of the labels, show that 'toxic' is the most frequent occurring label, while 'obscene' and 'insult' are also high, but the counts are very few for 'severe-toxic' and 'threat'

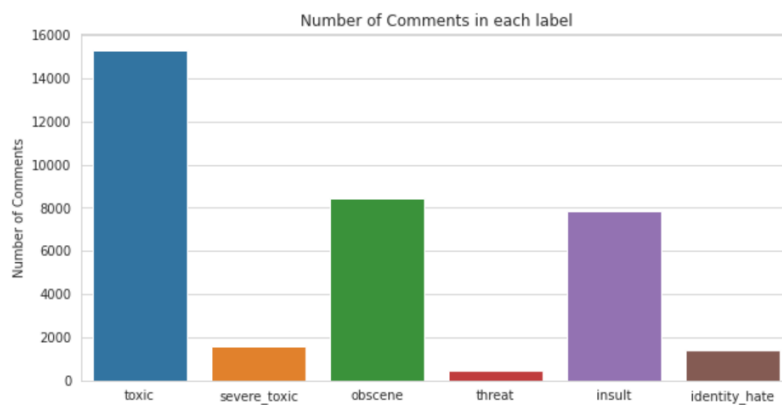


Figure 4 Count of Labels

Since the task is multi-label task, each comment can have multiple labels, therefore, we can also see the distribution of comments having multiple labels. Majority of the comments have only single label, and there are also many comments having two and three labels, but comments with all the six labels are extremely low as seen in plot below.

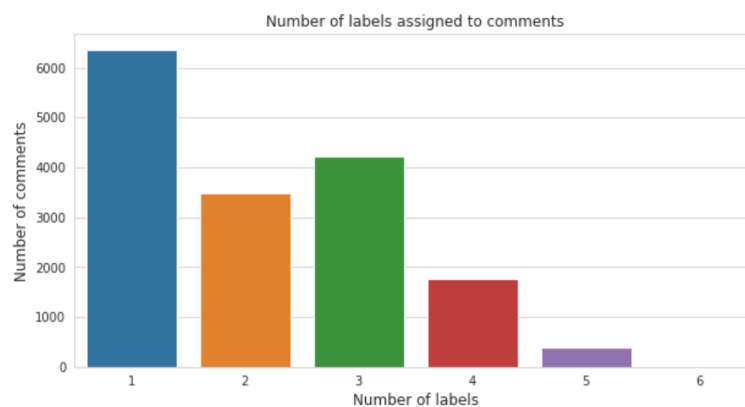
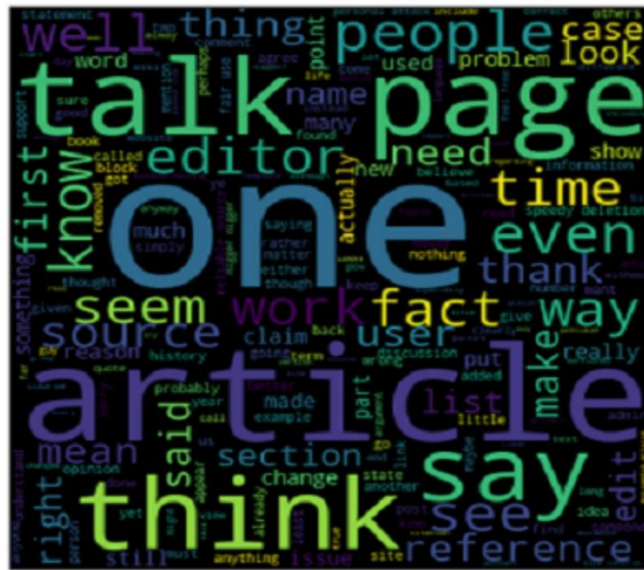


Figure 5 Counts of comments with multiple labels

In addition, from the distribution of the label counts we can also observe that majority of the comments, which is around 90% of the comments, do not have any labels assigned. This shows that majority of the comments are clean comments and do not show any toxic behavior. This distribution shows an imbalance problem in the data, hence AUC is a good metric to use for this problem.



From the word cloud, we see some frequently used words, but right now they do not make any significant distinction for classifying. However, they do look very similar to the words being used in online discussions or comments.



The correlation plot show that there is good relation between the labels 'toxic' and 'insult', 'toxic' and 'obscene', and 'obscene' and 'insult'. So, in the presence of one of the labels in the pairs for a

comment, it is likely that the comment will also have the other label.

5 Modeling Implementation

Before performing the modeling, data partition is performed to split the data into train and test set, in the ratio 0.7:0.3. The model will be trained on train split and will be evaluated on the test split.

5.1 Bag of Words

As an initial approach, we will use the bag of words representation to present the text for the modeling using TFIDF vectorization. Bag of words is a commonly used method for text or document classification, where it extracts features (unique words occurring in all the text documents) from text documents. Each document is then represented with the features, where it states the presence of the word (or wordcount) in the document. With TFIDF vectorization, instead of the simple word count, it represents a score that explains how relevant the word is to a document in all documents.

Before converting the texts to bag of words, text cleaning is performed by removing the punctuations, any numbers, stop-words and stemming the words. With the cleaned text, TFIDF vectorizer is applied to create the bag of words. TFIDF vectorizer is applied with n-grams of 1 and 2, i.e. it will create word features including a single word and two words. Also, the maximum features (vocabulary size) is restricted to 1000 words to control the sparsity in the data.

- Text cleaning is performed with the package 'textthero'. It provides easy to use text processing pipelines that makes text cleaning easy.
- For creating bag of words with TFIDF, 'TfidfVectorizer' from 'Sickit-learn' is applied.

5.2 Classification with Bag of Words

After constructing the bag of words, it is applied to train the classification models by applying the Binary Relevance and Classifier chain approaches. As stated above, Logistic Regression and Naïve Bayes models will be used as machine learning models for performing the classification tasks.

- To apply Binary Relevance problem transformation, 'scikit-multilearn' package is used.
- Classifier chain approach is applied with 'sklearn.multioutput' module from 'scikit-learn' package.
- For both the approaches, Logistic Regression with default parameters from 'scikit-learn' package is applied with solver as 'liblinear'. Similarly, for Naïve Bayes, 'MultinomialNB' from 'scikit-learn' is applied with default parameters.

After training the models on train data, they are evaluated on the test that with the three evaluation metrics stated above. The results of the classification is shown in the table below:

Model	Accuracy	AUC	Log loss
BR-LogisticRegression	0.979	0.960	0.0641
BR-MultinomialNB	0.976	0.955	0.071
Chain-LogisticRegression	0.979	0.950	0.0693
Chain-MultinomialNB	0.968	0.946	0.124

The results of text classification with bag of words and Binary Relevance and Classifier chain show that Binary Relevance with Logistic Regression (BR-LogisticRegression) is performing the best with better values for all the three evaluation metrics. Also, classifier chain is not performing better, which could be due to lack of any significant correlation between the class labels that was seen earlier from the correlation plot. However, the simplistic approach of Binary Relevance with Logistic Regression is already performing with good results for our benchmark. To see any improvements over this model, hyperparameter tuning with grid search is performed for the Logistic Regression with 5 cross validation for the cost parameter 'C'.

- Grid search is performed for the best value of 'C' with six different values [0.0001, 0.05, 1, 10, 100, 1000]

The results from the tuning task reports the default value used for cost parameter 'C', which is 1, as the best parameter. So, the hyperparameter tuning did not give any improvements.

5.3 Classification with Word Embedding

Word embedding is type of word representation method that represents each word by real-valued vectors with a predefined vector length. These vectors are learned mostly through deep learning methods. With word embedding, words with similar meaning are represented with similar vector representation that allows for capturing the word meaning. We can train our own word embedding or use a pretrained word embeddings that has already been trained on a large corpus of text. For simplicity and computational purpose a pretrained word2vec model from Google is used to extract the word embedding. To represent a sentence with word embedding (sentence embedding), the average of word embedding of words in the sentence is computed.

- For downloading the 'word2vec' model, 'gensim' package is used.

Since, earlier Binary Relevance with Logistic Regression was performing the best, word embedding will be used to replace bag of words for training the model to see for any improvements in the results. The classification results with the word embedding seems to be very similar to the bag of words approach as seen below in the table, thus, giving no improvements. Hence, till here we can still consider Binary Relevance with Logistic Regression as the best performing model.

Model	Accuracy	AUC	Log loss
BR-LogisticRegression(word embedding)	0.975	0.965	0.071

5.4 LSTM Model

- The first step for data preparation for LSTM is to tokenize the words and represent them as integers, where each unique word is represented by an integer value. In addition, the vocabulary size needs to be specified that determines the number of most frequent words to use in the modeling. Before tokenizing, some text cleaning is performed to remove punctuations and any numbers. A vocabulary size of 10000 was used for the modeling
- After tokenizing the text, padding is performed. Padding ensures all the text sequence are of same length. With a given sequence length, padding assigns 0s at beginning of sentence or after the end of sentence to match the required sequence length, to examples that have sequence length less than the specified length. Similarly, it will truncate the examples that

have higher sequence length than the specified length. For padding, the sequence length was specified as 200, since, earlier we saw majority of the comments had roughly the text length up to 200.

- LSTM model was trained with 'Tensorflow' and 'Keras'

The architecture of the LSTM model used is shown in the figure below. It was finalized with some trial and error.

```
model = Sequential()
model.add(Input(shape=(None,)))

# Embedding layer
model.add(Embedding(input_dim=VOCAB_SIZE+1, output_dim=300,
                    input_length=PAD_LENGTH, mask_zero=True))

# LSTM layer
model.add(LSTM(units= 50, dropout=0.4, return_sequences=False))
# Dense layer
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(6, activation='sigmoid'))
```

Figure 8 LSTM Model

LSTM model was trained with 'Adam' optimizer and a learning rate of 5e-4, applying 'BinaryCrossentropy' as the loss function. Similarly, the model was trained with a batch size of 32 for 50 epochs. However, early stopping criteria was applied to prevent the model from over fitting. In addition, the model at the best epoch was saved and used for later evaluation. The early stopping criteria stopped the model at 6 epoch as no improvement was observed for the validation data. The learning curve for the model is shown in the figure below:

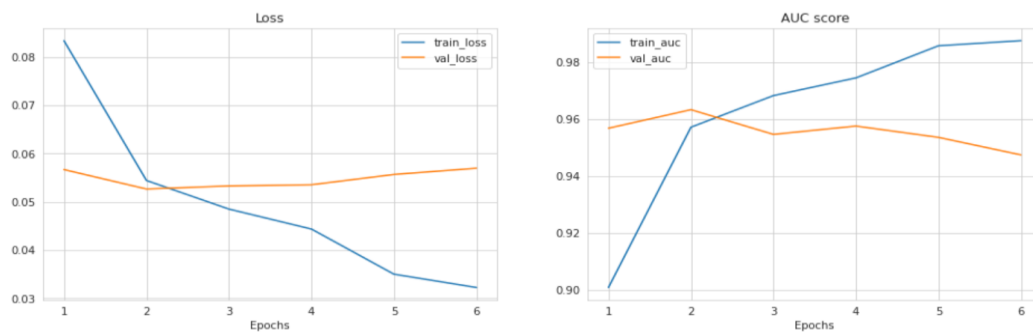


Figure 9 LSTM Performance Curve

The results from the LSTM model gave some improvement over the Logistic Regression model as shown below. This improvement was expected as LSTM as a deep learning model is quite powerful and useful when modeling sequence data and for text it also maintains the semantic meaning, unlike bag of words, where the word order is not maintained.

Model	Accuracy	AUC	Log loss
LSTM	0.9816	0.9634	0.0527

5.5 BERT Model

As stated above, as a final model transfer learning with BERT is applied, since, it is a powerful model trained on large corpus of text and has been producing state of the arts results in many applications.

- To implement transfer learning with BERT, **Tensorflow Hub** is used, as it provides an extensive list of such pretrained models, including BERT and also easy to use code example to fine-tune the models.
- There are different variants of the BERT model, for this problem the selected BERT model is the small bert with uncased, which is available as:

'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-512_A-8/1'

- For BERT, since it is a pretrained model, the data has to be pre-processed according to the process used in the pretrained model. For that, tensorflow hub provides the necessary processing helper and we can simply make use of it. The text processor for the selected BERT model is available as:

'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3'

After specifying the appropriate model and the text processor from Tensorflow Hub, we can train the BERT model as a normal Keras model. The architecture for training the BERT model is shown below.

```
# create keras model
tfhub_handle_encoder = 'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-512_A-8/1'
tfhub_handle_preprocess = 'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3'

def build_classifier_model():
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
    outputs = encoder(encoder_inputs)
    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(500, activation='relu')(net)
    net = tf.keras.layers.Dense(6, activation='sigmoid', name='classifier')(net)
    return tf.keras.Model(text_input, net)
```

Figure 10 BERT Model

BERT model was trained with optimizer weight decay with adam 'AdamW' and a learning rate of $3e-5$. 'BinaryCrossentropy' was applied as a loss function and it was trained with batch size of 32 and epochs of 20. Again, here, stopping criteria was applied to stop the model from overfitting and the model at best epoch was saved for later evaluation. The performance curve for the BERT model is shown below.

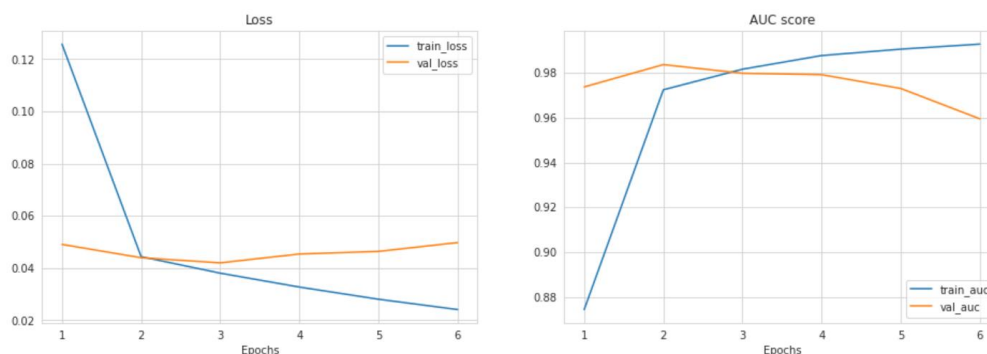


Figure 11 BERT Performance Curve

The results from the BERT model as expected gave an improvement over the previous models. As a pretrained on large corpus of text and being more sophisticated, it was expected to perform better. The results from BERT model on test data is shown below:

Model	Accuracy	AUC	Log loss
BERT	0.9836	0.9798	0.042

6 Conclusion

Different modeling approaches have been covered to solve the multi-label classification problem of classifying comments to the toxic labels. From the models applied, BERT model is performing the best with the mean AUC ROC score of 0.98 on the test data. However, the evaluation so far has been done on the test data that was extracted as a split from the original train data provided at Kaggle. As, there exists and completely different test data that does not have any labels and is used by Kaggle for final evaluation. The predictions on this test data is made with the models and the results are submitted to the Kaggle for the final evaluation. The results of the submissions made to Kaggle is shown in the table below.

Models	ROC AUC
Logistic Regression(Binary Relevance)	0.95875
LSTM	0.97128
BERT	0.98172

The results obtained from the Kaggle submission also shows that the BERT model is performing the best for solving the multi-label classification task. With a high score for the AUC, we can confidently say that the BERT model is suitable for classifying the comments to the toxic labels.

REFERENCES

[https://link.springer.com/article/10.1007/s11704-017-7031-7#:~:text=Binary%20relevance%20is%20arguably%20the,\(one%20per%20class%20label\).](https://link.springer.com/article/10.1007/s11704-017-7031-7#:~:text=Binary%20relevance%20is%20arguably%20the,(one%20per%20class%20label).)

<https://machinelearningmastery.com/what-are-word-embeddings/>

https://www.tensorflow.org/tutorials/text/classify_text_with_bert