TDDE16

TEXT MINING PROJECT

ON

Sentiment Analysis Of IMDB Movie Reviews

Author:
Chetan Kandya (cheka108)
cheka108@student.liu.se

January 12, 2023

Abstract

Sentiment Analysis is a powerful Natural Language Processing (NLP) technique and has evolved significantly over the last decade due to the explosive growth of user generated data and advancement in the field of machine learning. It has many different use-cases in the real world, like: social media monitoring, brand monitoring, customer service and feedback analysis. In this project, different machine learning models are trained and evaluated on the IMDB movie reviews dataset (with 50,000 movie reviews), for the task of sentiment classification. The Multinomial Naive Bayes model and linear Support Vector Machine model performed well but could not outperform the BERT model which achieved an average F1 score of 0.94 and 94% accuracy and the RoBERTa model which achieved an average F1 score of 0.96 and 96% accuracy. Finally, an error analysis was performed on the misclassified reviews of the BERT and RoBERTa model, which could help to further improve the overall performance.

1 Introduction

Sentiment Analysis is the task of correctly identifying the sentiment from raw text. There are several types of sentiment analysis, the most popular ones are:

1. Fine-grained sentiment analysis
2. Emotion detection
3. Binary sentiment analysis
4. Aspect based sentiment analysis
5. Multilingual sentiment analysis

The focus of this project is to explore different machine learning and deep learning techniques to perform binary sentiment analysis to detect the positive or negative sentiment from a given text.

Sentiment analysis can be used by different businesses to understand the emotions and sentiments of their customers. By correctly identifying customer's sentiment through their feed-backs, reviews and social media conversations, the companies can make well-informed decisions and provide their customers with better services.

2 Theory

This section will discuss about different machine learning and deep learning techniques used in this project for the task of sentiment analysis.

2.1 Machine Learning techniques

2.1.1 Multinomial Naive Bayes model

Naıve Bayes is a probabilistic classifier that uses Bayes' theorem to classify input data into different categories by calculating the posterior probability for each class and selecting the most likely case (Bayes and Price, 1763).

Multinomial Naive Bayes classifier is a specific version of the Naive Bayes classifier which uses a multinomial distribution for each feature in the model. To train the model, the text is first converted into a numerical representation, such as a tf-idf representation or a a bag of words representation. The model is then trained on a labeled dataset by estimating the conditional probability of each class given the feature representation and the prior probability of each class.

2.1.2 Support Vector Machines

Support Vector Machines (SVMs) is a classification algorithm that aims to find the optimal boundary (hyperplane) that separates the different classes in the feature space. Data points closest to the hyperplane are called support vectors. The data points are classified based on the distance vector measured from the hyperplane (Cortes, 1995). The algorithm is then optimized by maximizing the distance between the boundary and the closest points.

2.2 Deep Learning techniques

2.2.1 Bidirectional Encoder Representations from Transformers (BERT)

BERT is a transformer language model which can be fine tuned for different natural language processing (NLP) tasks. BERT was created by Jacob Devlin and his colleagues from Google in 2018 (Devlin et al., 2018).

BERT model architecture has multiple bidirectional encoder layers and self-attention heads. The implementation is almost identical to original transformer implementation described in (Vaswani et al., 2017) and released in the tensor2tensor library [1].

BERT was pre-trained on two tasks:

Language Modeling: the model was trained to predict 15% of the masked tokens from context.

Next Sentence Prediction: the model was trained to predict if the input sentence was probable or not given the first sentence.

[1] https://github.com/tensorflow/tensor2tensor

The pre-training of BERT model was computationally very expensive as it was done on unlabeled data extracted from the Books-Corpus (Zhu et al., 2015) with 800M words and English Wikipedia with 2,500M words. But a pre-trained BERT model can be fine-tuned for different tasks with fewer resources.

The original English-language BERT has two models, **BERT-Base** (12 encoders with 12 bidirectional self-attention heads, 768 hidden units and 110 million parameters) and **BERT-Large** (24 encoders with 16 bidirectional self-attention heads, 1024 hidden units and 340 million parameters). But nowadays, many different pre-trained BERT models are available [2].

2.2.2 A Robustly Optimized BERT Pretraining Approach (RoBERTa)

RoBERTa is an extension of the BERT model and has similar architecture. It was created by Yinhan Liu and his colleagues in 2019 to optimize BERT (Liu et al., 2019).

Liu found that BERT was significantly undertrained and proposed the following modifications: (1) training the model for longer time, with larger batch size and using more data; (2) removing the next sentence prediction objective; (3) training on longer sequences; and (4) dynamically changing the masking pattern applied to the training data.

Overall, RoBERTa outperforms BERT for many different tasks including (GLUE) General Language Understanding Evaluation benchmark (Wang et al., 2018) and (SQuAD) Stanford Question Answering Dataset (Rajpurkar et al., 2016).

Similar to BERT, the original English-language RoBERTa has two models, **RoBERTa-Base** (12 encoders with 12 bidirectional self-attention heads, 768 hidden units and 125 million parameters) and **RoBERTa-Large** (24 encoders with 16 bidirectional self-attention heads, 1024 hidden units and 340 million parameters). However, there are many different pre-trained RoBERTa models available [2].

2.3 Related Work

The authors of the paper (Chiorrini et al., 2021) used BERT to perform sentiment analysis on Twitter data and evaluated the performance of the trained model on real-world tweet datasets. The BERT model achieved an accuracy of 92%.

(Devika et al., 2016) compares various different techniques like SVM and Naive Bayes for the task Sentiment Analysis by analyzing various methodologies.

(Bingyu and Arefyev, 2022) analyzed the performance of the model developed by (Thongtan and Phienthrakul, 2019) with different amounts of training data (subsets of the IMDB dataset) and compared it to the Transformer-based RoBERTa model and came to a conclusion that RoBERTa performs better.

(Tan et al., 2022) proposed a model that combines a RoBERTa model and a LSTM (Long Short-Term Memory) model for sentiment analysis. The RoBERTa model was used to map the words into a compact meaningful word embedding space while the LSTM model captures the long-distance contextual semantics effectively. The proposed hybrid model outshines the state-of-the-art methods by achieving F1-scores of 93%, 91%, and 90% on IMDb dataset, Twitter US Airline Sentiment dataset, and Sentiment140 dataset, respectively.

(Alaparthi and Mishra, 2020) used the BERT model to perform sentiment analysis on IMDB movie reviews dataset and achieved 92.3% accuracy and an average F1 score of 0.9231.

3 Data

The dataset used for training and evaluating different models in this project is called the IMDB movie reviews dataset. This dataset contains 50,000 movie reviews and each review is labelled with a 'positive' or 'negative' sentiment. The overall distribution of labels is balanced as there are 25,000 reviews for each sentiment.

3.1 Cleaning the data

The data was read from a file called 'IMDB dataset.csv' into a data-frame using pandas. All the duplicate rows were dropped and the sentiment label for each review was converted to integers i.e. 'negative' was converted to '0' and 'positive' was converted to '1'. To analyse the data better, all the reviews were processed using a custom function called 'preprocessing', which updated each review with the following changes:

- Changed 't to 'not'.
- Removed @name.
- Removed stop-words.
- Removed trailing blank spaces.
- Removed all the html tags like:
 etc.

[2] https://huggingface.co/transformers/v2.9.1/pretrained_models.html

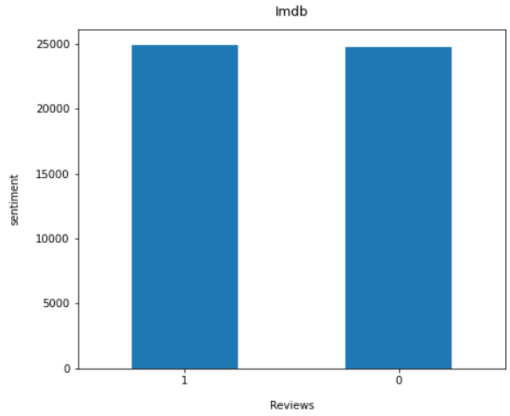

Figure 1: Data distribution for IMDB movie reviews

3.2 Data Analysis

To get a better understanding of the data, the positive and negative reviews were analysed separately. Before starting the analysis, the data was pre-processed to remove all the stop words, html tags, trailing blank spaces, specail symbols etc.

3.2.1 Positive Reviews

The longest positive movie review was of 9185 words and majority of the reviews had somewhere between 500 to 800 words. Figure 2 shows the top 10 words with the lowest idf values from the positive movie reviews.

Figure 18 shows the word cloud of the positive movie reviews.

	term	idf
18955	film	1.515256
35805	movie	1.521464
21965	good	1.676134
31168	like	1.787076
53912	time	1.913983
22377	great	2.019822
47344	see	2.027091
51163	story	2.075847
58526	watch	2.103204
8775	character	2.115021

Figure 2: Top 10 words from positive reviews

3.2.2 Negative Reviews

The longest negative movie review was of 5664 words. Whereas, the majority of the reviews had somewhere between 500 to 800 words. Figure 3 shows the top 10 words with the lowest idf values from the negative movie reviews.

Figure 19 shows the word cloud of the negative movie reviews.

	term	idf
33906	movie	1.363075
18124	film	1.543989
29475	like	1.646208
3476	bad	1.767813
20928	good	1.804423
51306	time	1.930830
55725	watch	1.956524
44950	see	2.092799
51025	think	2.105927
8267	character	2.123827

Figure 3: Top 10 words from negative reviews

After looking at the terms with the lowest idf values in the complete dataset, it is safe to conclude that the model may perform better if the stop words related to movies are identified and removed.

4 Method

This section describes the methodology used to train and evaluate different models used in this project.

4.1 Multinomial Naive Bayes model

In order to train and evaluate a Multinomial Naive Bayes model, the following steps were followed:

1. **Preprocessing the data for Multinomial Naive Bayes model**
   Before fitting the model, all the reviews were passed through the preprocessing function mentioned in the previous section in order to remove all the stop words and html tags.

2. **Vectorizing all the reviews in the training data**
   All the reviews in the training data had to be vectorized before they can be used for training the model. For vectorizing the data, TfidfVectorizer [3] from the sklearn library was used. The TfidfVectorizer converts a collection of raw documents to a matrix of TF-IDF features.

3. **Fitting the Multinomial Naive Bayes model**
   The MultinomialNB classifier was imported

[3] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer

from the sklearn library to fit the model. To make things simpler, a pipeline[4] from sklearn library was used. The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters. In this case, the vectorization and fitting of the model were combined into a single step.

4. **Evaluating the model on the test data**
   For the evaluation, a function called classification_report[5] is used. This function reports per-class precision, recall, F1 score and overall accuracy.

5. **Optimizing the model**
   The model was then optimized by doing a hyperparameter search to find the optimal hyperparameters. However, manually tweaking the hyperparameters of the various components of a vectorizer–classifier pipeline can be very hard. Therefore, a grid search with 5-fold cross-validation was implemented using the class GridSearchCV[6] provided by the scikit-learn library.

4.2 SVM

For fitting and evaluating a linear support vector machine model, all the same steps for fitting a multinomial naive bayes model were followed. The only difference in fitting a SVM model is that a SGDClassifier is used instead of a MultinomialNB classifier. SGDClassifier[7] is a class provided by the sklearn library and by default, it fits a linear support vector machine (SVM). In order to fit the SVM model using the optimal hyperparameters a grid search with 5-fold cross-validation was implemented using the GridSearchCV class.

4.3 Fine-tuning A BERT Model

In this project, the 'bert-base-uncased' model with 110 million parameters is used. Uncased means that there are only lower-case letters in the vocabulary.

4.3.1 Hyperparameters used for the BERT model

For fine-tuning the BERT model, The authors (Devlin et al., 2018) recommended the following hyperparameters:

- **MAX_LEN = 512**
  Maximum length of each sentence.
- **BATCH_SIZE = 16**
  Size of each batch.
- **NUM_CLASSES = 2**
  Number of target classes. In this case there are two classes: 0 (negative) and 1 (positive).
- **LEARNING_RATE = 2e-5**
  Learning rate governs the rate at which the algorithm learns or updates the parameters estimates.
- **NUM_EPOCHS= 5**
  Number of iterations for training.

4.3.2 Preprocessing the data for BERT model

In order to get the data into the format in which it can be used as an input to the BERT model, the following changes are required:

1. Adding special tokens to the start and end of each sentence. At the beginning of every sentence a [CLS] token was added and at the end of each sentence a [SEP] token was added.
2. Padding and truncating all sentences to a single constant length (512). Padding is done with a special [PAD] token, which is at index 0 in the BERT vocabulary.
3. Splitting every sentence into tokens using BertTokenizer [8].
4. Creating 'attention mask' in order to differentiate real tokens from padding tokens. The "Attention Mask" is simply an array of 1s and 0s indicating which tokens are padding and which are not.

Fortunately, all these steps are implemented via the tokenizer. After tokenizing the data successfully, a tensor dataset with 'Input IDs', 'Attention Mask' and 'Gold Labels' was created. This dataset was

[4] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[8] https://huggingface.co/docs/transformers/v4.25.1/en/model_doc/bert#transformers.BertTokenizer

further divided into three different datasets, 70% for training, 15% for validation and 15% for testing. Finally, three different data loaders were created using each dataset using a batch size of 16.

4.3.3 BERT Model

Firstly, the pre-trained BERT model had to be modified for classification and then the model had to be trained on the complete dataset until it is fine-tuned for the task of sentiment analysis. To make things simple, the huggingface pytorch library has provided multiple implementations of BERT, designed for a variety of NLP tasks.

In this project, one of the pytorch implementation of BERT called **BertForSequenceClassification**[9] is used. This is a normal BERT model with an added single linear layer on top for classification, which is used as a sentence classifier.

4.3.4 Optimizer

The default AdamW optimizer et al. (Loshchilov and Hutter, 2017) is used. AdamW is an Adam optimizer with weight decay regularization. Which means that the weight decay is decoupled from the optimization steps taken with respect to the loss function.

4.3.5 Training, Evaluating and Testing the Model

After setting up the model with the tokenizer and optimizer, the model was ready to be trained. For training the model, a function called **train** was implemented which performed the following steps:

1. Batch wise unpacking the data inputs and labels from the dataloader.
2. Clear out the gradients calculated in the previous pass.
3. Send the data onto the another device (GPU) for acceleration.
4. Forward pass (feed input data through the network).
5. Backward pass (back-propagation).
6. Update the parameters of the network with optimizer.step()
7. Monitor the progress.

For evaluating the model, a function called **evaluate** was implemented which performed the following steps:

1. Batch wise unpacking the data inputs and labels from the dataloader.
2. Clear out the gradients calculated in the previous pass.
3. Forward pass (feed input data through the network)
4. Monitor the progress and compute the loss on the validation data.

For testing the model, a function called **test** was implemented which performed the following steps:

1. Batch wise unpacking the data inputs and labels from the dataloader.
2. Clear out the gradients calculated in the previous pass.
3. Predict the class for the input data.

4.3.6 Saving the model

After the training is completed, the model with the best validation accuracy was saved and was used for testing.

4.4 Fine-tuning A ROBERTa Model

In this project, the 'RoBERTa-large' model with 340 million parameters is used.

4.4.1 Preprocessing the data for RoBERTa model

Similar to BERT, the RoBERTa model also expects the input data to be in a certain format. And for formatting the data, a RobertaTokenizer[10] is used, which performs the following tasks in order to format and tokenize the input data:

1. Adding special tokens to the start and end of each sentence. Unlike the BERT model, the special tokens for the RoBERTa model are: ['</s>', '<mask>', '<pad>', '<s>', '<unk>'].
2. Padding and truncating all sentences to a single constant length (512).
3. Splitting every sentence into tokens using RobertaTokenizer.
4. Creating 'attention mask' in order to differentiate between pad tokens and real tokens.

[9] https://huggingface.co/transformers/v2.2.0/model_doc/bert.html#bertforsequenceclassification

[10] https://huggingface.co/docs/transformers/model_doc/roberta#transformers.RobertaTokenizer

4.4.2 Hyperparameters used for the RoBERTa model

- **MAX_LEN = 512**
  Maximum length of each sentence.
- **BATCH_SIZE = 8**
  Size of each batch.
- **NUM_CLASSES = 2**
  Number of target classes. In this case there are two classes: 0 (negative) and 1 (positive).
- **LEARNING_RATE = 2e-5**
  Learning rate governs the rate at which the algorithm learns or updates the parameters estimates.
- **NUM_EPOCHS= 5**
  Number of iterations for training.

4.4.3 RoBERTa Model

Similar to BERT, the pre-trained RoBERTa model also had to be modified for the task of classification. In this project, one of the pytorch implementation of RoBERTa called **RobertaForSequenceClassification**[11] is used. This is a normal RoBERTa model with another linear layer added on top for classification, this layer is used as a sentence classifier.

4.4.4 Optimizer

The default AdamW optimizer et. al. (Loshchilov and Hutter, 2017) is used.

4.4.5 Training, Evaluating and Testing the Model

After setting up the model with the tokenizer and optimizer, the model was ready to be trained. For training, evaluating and testing the RoBERTa model, the **train**, **evaluate** and **test** functions were implemented similar to the BERT model.

4.4.6 Saving the model

After the training is completed, the model with the best validation accuracy was saved and used for testing.

[11] https://huggingface.co/docs/transformers/v4.25.1/en/model_doc/roberta-#transformers.RobertaForSequenceClassification

5 Results

5.1 Multinomial naive Bayes Model

The Multinomial naive bayes model, when trained using the default parameters took very little time to train and gave an average F1 score of 0.86 and overall accuracy of 86% when evaluated on the test dataset. The complete classification report can be seen in Figure 4, which shows the precision, recall and F1 score of each class, along with the average accuracy and F1 score.

	precision	recall	f1-score	support
0	0.85	0.88	0.86	7390
1	0.87	0.84	0.86	7485
accuracy			0.86	14875
macro avg	0.86	0.86	0.86	14875
weighted avg	0.86	0.86	0.86	14875

Figure 4: Classification report for Multinomial Naive Bayes model trained using defualt parameters

Figure 5 shows the precision, recall and F1 score of each class, along with the average accuracy and F1 score of the multinomial naive bayes model trained using the optimal parameters.

The optimal parameters were found using grid search with 5-fold cross-validation and evaluated on the test dataset. The MultinomialNB model achieved even better results with optimal parameters. It got an average F1 score of 0.89 and overall accuracy of 89% when evaluated on the test dataset. Figure 6 shows the confusion matrix of the results.

	precision	recall	f1-score	support
0	0.88	0.90	0.89	7390
1	0.90	0.87	0.88	7485
accuracy			0.89	14875
macro avg	0.89	0.89	0.89	14875
weighted avg	0.89	0.89	0.89	14875

Figure 5: Classification report for Multinomial Naive Bayes model trained using optimal parameters

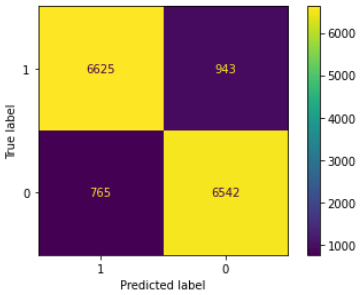

Figure 6: Confusion matrix for Multinomial Naive Bayes model trained using optimal parameters

5.2 SVM Model

The linear support vector machine model also took very little time to train and performed similar to the MultinomialNB model when trained using the optimal parameters. It got an average F1 score of 0.89 and overall accuracy of 89% when evaluated on the test dataset. Figure 7 shows the precision, recall and F1 score of each class, along with the average accuracy and F1 score of the linear SVM model trained using the optimal parameters.

	precision	recall	f1-score	support
0	0.88	0.90	0.89	7390
1	0.90	0.87	0.88	7485
accuracy			0.89	14875
macro avg	0.89	0.89	0.89	14875
weighted avg	0.89	0.89	0.89	14875

Figure 7: Classification report for SVM model

5.3 BERT Model

The BERT model took almost 5 hours to train on the provided IMDB dataset, but gave much better results as compared to the MutilnomialNB and the SVM model. The BERT model got an average F1 score of 0.94 and overall accuracy of 94% when evaluated on the test dataset. Figure 8 shows the precision, recall and F1 score of each class, along with the average accuracy and F1 score of the BERT model. Figure 9 shows the confusion matrix of the results.

	precision	recall	f1-score	support
0	0.95	0.94	0.94	3705
1	0.94	0.95	0.94	3733
accuracy			0.94	7438
macro avg	0.94	0.94	0.94	7438
weighted avg	0.94	0.94	0.94	7438

Figure 8: Classification report for BERT model

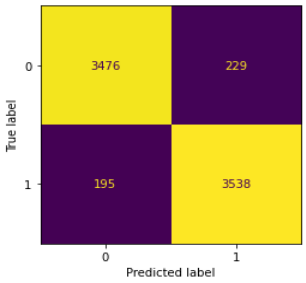

Figure 9: Confusion matrix for BERT model

5.4 RoBERTa Model

Similar to BERT, the RoBERTa model took approximately 7 hours to train, But produced the best results among all the models. It got an average F1 score of 0.96 and overall accuracy of 96% when evaluated on the test dataset. Figure 10 shows the precision, recall and F1 score of each class, along with the average accuracy and F1 score of the RoBERTa model evaluated on the test dataset.

	precision	recall	f1-score	support
0	0.96	0.95	0.96	3705
1	0.95	0.96	0.96	3733
accuracy			0.96	7438
macro avg	0.96	0.96	0.96	7438
weighted avg	0.96	0.96	0.96	7438

Figure 10: Classification report for RoBERTa model

Figure 11 shows the confusion matrix for the RoBERTa model.

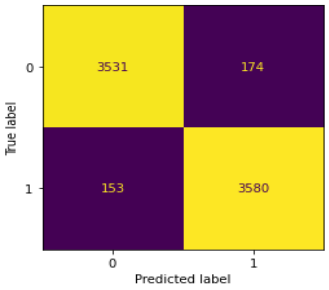

Figure 11: Confusion matrix for RoBERTa model

6 Discussion

6.1 Performance Comparison

The Multinomial Naive Bayes model and linear SVM model took very less time to train and got good results when evaluated on the test dataset. On the other hand, BERT and RoBERTa model took very long time and more computation power as compared to the MultinomialNB and SVM model but got significantly better results.

6.2 Performance evaluation of BERT model

As it can be seen from the figure 12 that the training loss is going down and the validation loss is going up, which means that the model is over-fitting. On the other hand, even though the validation loss is increasing, it can be seen from figure 13 that the validation accuracy is also increasing.

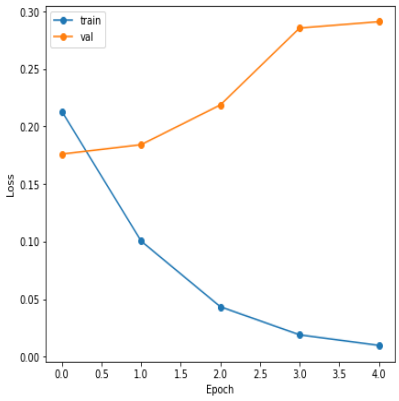

Figure 12: Training and validation loss of BERT model

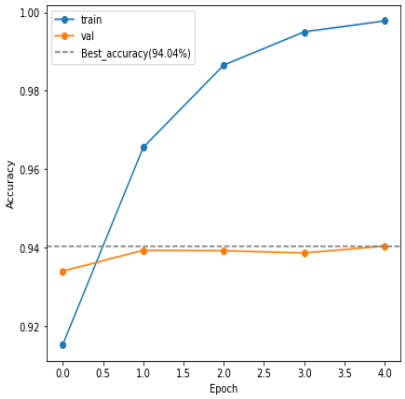

Figure 13: Training and validation accuracy of BERT model

In order to avoid over-fitting, I decided to retrain the BERT model by changing the **Learning Rate** from 2e-5 to 1e-6 and **Batch Size** from 16 to 8. And as it can be seen from the figure 14 and 15 that the training and validation loss are both decreasing and the training and validation accuracy are increasing, which implies that the model is not over-fitting.

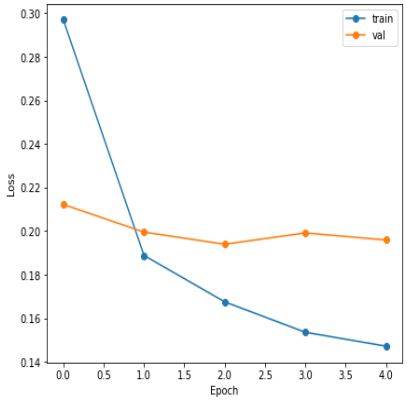

Figure 14: Training and validation loss of BERT model

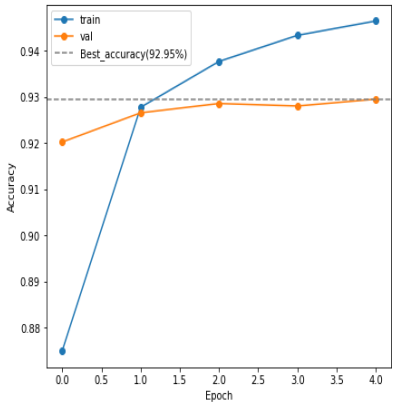

Figure 15: Training and validation accuracy of BERT model

6.3 Performance evaluation of RoBERTa model

Similar to the BERT model, the RoBERTa model was also over-fitting, as it can be seen in figure 16 that the training loss is going down and the validation loss is going up. But I did not have enough computation power to retrain the RoBERTa model by fine-tuning the hyper-parameters like learning rate and batch size.

But it can be noticed from the figure 17 that even though the validation loss is increasing, the validation accuracy is also increasing and has reached the point where with further training the validation accuracy will start to drop.

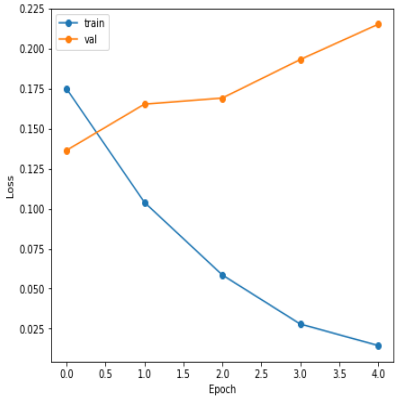

Figure 16: Training and validation loss of RoBERTa model

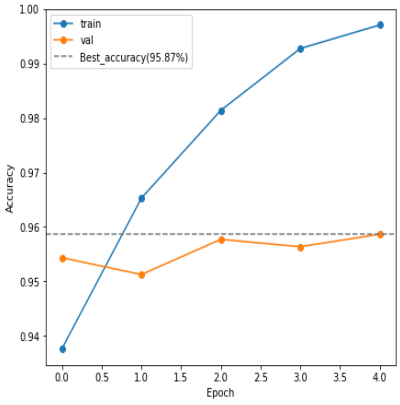

Figure 17: Training and validation accuracy of RoBERTa model

6.4 Error Analysis of BERT and RoBERTa Model

After training the BERT and RoBERTa model, the models with best validation accuracy were saved. These model were then used to perform sentiment classification on the test dataset. Although, BERT got 94% overall accuracy and RoBERTa got 96% overall accuracy on the test data, which is significantly better than the baseline model, it is always good to analyse the misclassified reviews in order to further improve the performance of the model.

For identifying the misclassified reviews, I wrote a function called error_report which returned all the **false positives** (1 identified as 0) and **false negatives** (0 identified as 1) reviews.

After analyzing the misclassified reviews, the following conclusions were drawn:

- Most miss-classified reviews had length much greater than 512 words, which is the maximum padding length of each review. Which makes them harder to classify.
- After reading some of the miss-classified reviews, it was be observed that most misclassified reviews did not talk about the movie or the experience of the person who is watching the movie, but is the summary of their whole day or any other event.
- RoBERTa seems to perform better than the BERT model on most of the edge cases.

7 Conclusion

After looking at the performance of all the models, it can be concluded that RoBERTa performed the best with an average F1 score of 0.96 and overall accuracy of 96%. Closely followed by BERT with an average F1 score of 0.94 and overall accuracy of 94%. And even though both BERT and RoBERTa requires long time and high computation power to train, they show a significant improvement in the results in comparison with the Multinomial naive bayes model and linear SVM model.

The BERT model developed in this project performs a little better than the model developed in the paper by (Alaparthi and Mishra, 2020) which got 92.3% accuracy and a F1 score of 0.9231 for the IMDB movie reviews dataset. Whereas, the RoBERTa model achieved the benchmark accuracy of 96%.

8 Future Improvements

- Combining the BERT and RoBERTa model to create a hybrid model and evaluate its performance on the same dataset.
- Identify and remove the domain specific stop words to further improve the performance.
- Implement other type of sentiment analysis techniques, like: Fine-grained sentiment analysis, aspect based sentiment analysis and Emotion detection.
- Investigate other machine learning and deep learning techniques for the task of sentiment analysis.

References

Shivaji Alaparthi and Manit Mishra. 2020. Bidirectional encoder representations from transformers (bert): A sentiment analysis odyssey.

Thomas Bayes and null Price. 1763. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s. *Philosophical Transactions of the Royal Society of London*, 53:370–418.

Zhang Bingyu and Nikolay Arefyev. 2022. The document vectors using cosine similarity revisited. In *Proceedings of the Third Workshop on Insights from Negative Results in NLP*. Association for Computational Linguistics.

Andrea Chiorrini, Claudia Diamantini, Alex Mircoli, and Domenico Potena. 2021. Emotion and sentiment analysis of tweets using bert. In *EDBT/ICDT Workshops*.

V. Cortes, C.and Vapnik. 1995. Support-vector networks. *Mach Learn*, 20:273–297.

M.D. Devika, C. Sunitha, and Amal Ganesh. 2016. Sentiment analysis: A comparative study on different approaches. *Procedia Computer Science*, 87:44–49. Fourth International Conference on Recent Trends in Computer Science & Engineering (ICRTCSE 2016).

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Kian Long Tan, Chin Poo Lee, Kalaiarasi Sonai Muthu Anbananthen, and Kian Ming Lim. 2022. Roberta-lstm: A hybrid model for sentiment analysis with transformer and recurrent neural network. *IEEE Access*, 10:21517–21525.

Tan Thongtan and Tanasanee Phienthrakul. 2019. Sentiment classification using document embeddings trained with cosine similarity. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 407–414, Florence, Italy. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books.

A Appendix

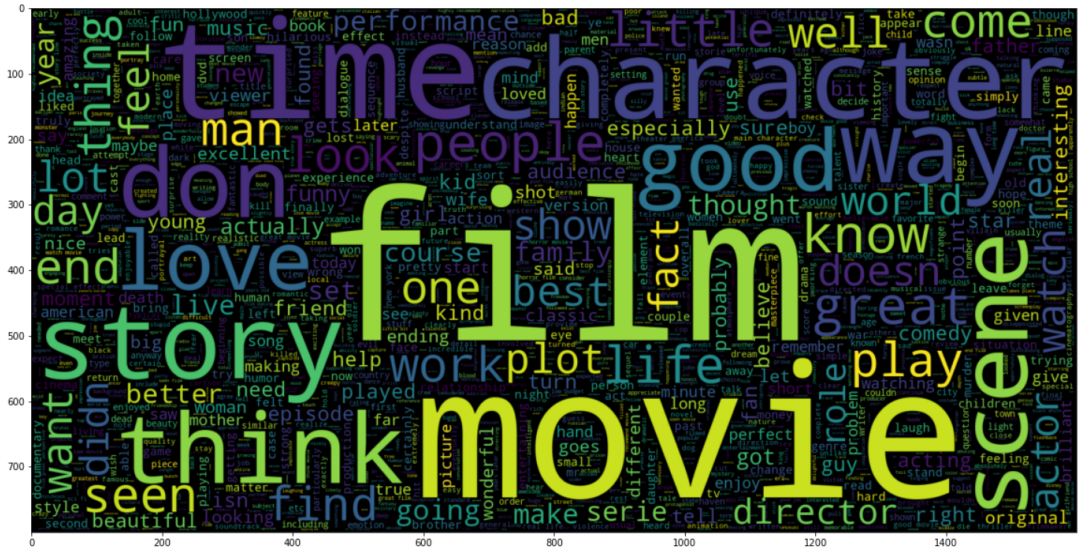

Figure 18: Wordcloud for positive IMDB movie reviews

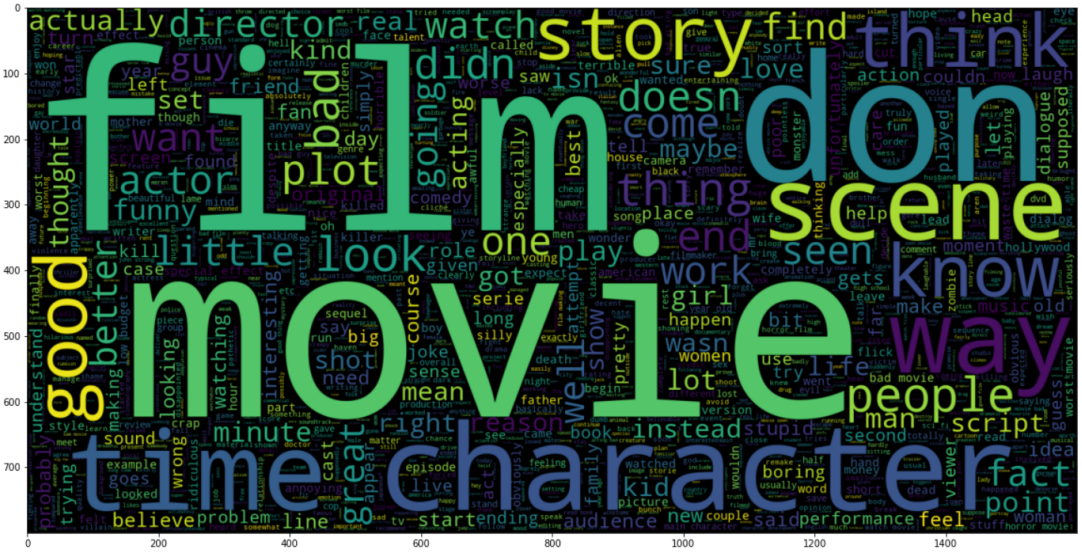

Figure 19: Wordcloud for negative IMDB movie reviews