COMP 551
Project write-up for
Mini-Project 3
Classification of Textual Data

Yajiv Luckheenarain, Anna Decterov, and Eric Guan

The goal of the project was to implement the naive Bayes algorithm from scratch and BERT with pre-trained weights, and compare their performances on the IMDB review dataset. The report details the process of acquiring and pre-processing the data, as well as implementing the two algorithms. For the naive Bayes method, the text data was turned into numerical features using the bags of words representation with the scikit-learn function CountVectorizer. For BERT, the input text was tokenized and converted into numerical features using the BertTokenizer from the transformers package. The results of the experiments showed that BERT outperformed naive Bayes in terms of accuracy. However, Bert also had a much longer training time. The report also provides implementation details and suggestions for further improvement.

I. Introduction

NATURAL Language Processing (NLP) is a subfield of artificial intelligence that focuses on enabling computers to understand and generate human language. It involves the use of various techniques and algorithms to process, analyze, and manipulate natural language data. NLP has numerous applications in different fields, including language translation, chatbots, speech recognition, and sentiment analysis. In our project, we implemented two models learned in class: Naive Bayes and BERT (Bidirectional Encoder Representations from Transformers). Both models were trained on the IMBD reviews dataset for sentiment analysis.

The sentiment of an IMDB movie review can be predicted using the Naive Gaussian Bayes machine learning algorithm. Based on the frequency of tokens, or lack thereof, in the review, the algorithm determines the likelihood that a specific review falls into the positive or negative category. It is based on the supposition that each token's recurrence in the review occurs independently of all other words and that the word's frequency follows a Gaussian distribution. By applying Bayes theorem, the model can determine the likelihood that a review will be positive or negative given the frequency of tokens in the review.

The model finds the log-likelihood probability that a review is positive or negative given the tokens it contains and classifies that review in the class that maximizes said probability. Finally, the review's sentiment is then labelled as positive or negative depending on the before-mentioned class it is now part of.

BERT makes predictions that are more accurate since it considers the relationships between words in the review as well as their position in the review. The input word sequence is transformed into a vector representation of tokens and their frequencies. In this project, we used the pre-trained BERT model "bert-base-uncased". We then fine-tuned said model by computing the weights it associates with tokens, their frequencies, and their relative position in a review. These computations were done using the Adam optimizer. We used the default configurations of this BERT model from TFBertForSequenceClassification python class. This model uses 12 layers, where each layers contains 12 attention heads. Finally, each attention heads were limited to 512 tokens. This means that each attention head contains a 512 x 512 self-attention matrix whose purpose is to analyze with greater complexity the relationship between tokens of a review and their relationship to the sentiment of the overall text.

Fig. 1: Word Clouds of positive and Negative reviews

II. Datasets

This dataset includes reviews of movies and the sentiment (positive and negative) associated with them. It was used to train, test, and validate our models. The core data set contains 25 000 reviews for the training set and 25 000 reviews for the testing set labelled positive or negative by human annotators.

The dataset contains an equal number of positive and negative reviews. The length of the review in character and words does not seem to have a correlation to the reviews' sentiment as seen in Fig. 2. As expected, Fig 1 shows that the words used in the reviews are different based on their associated sentiment. It is obvious that words are used as features in our models. Fig 2 also shows that punctuation symbols have an impact on the outcome of the review. It would be a good idea to use a tokenizer that keeps punctuation as a token. Even though negative reviews seem to have more upper-case words, we found that it would be more straightforward to generalize all words to be lower-case since that was done in most examples we found [**saha2021basics**].

For the Naive Bayes model, we instantiate a list of stop words containing words related to the movie industry which don't pertain significantly to sentiment such as punctuation and "a". Those tokens are then removed from the review and a matrix is created where each row is a different review and

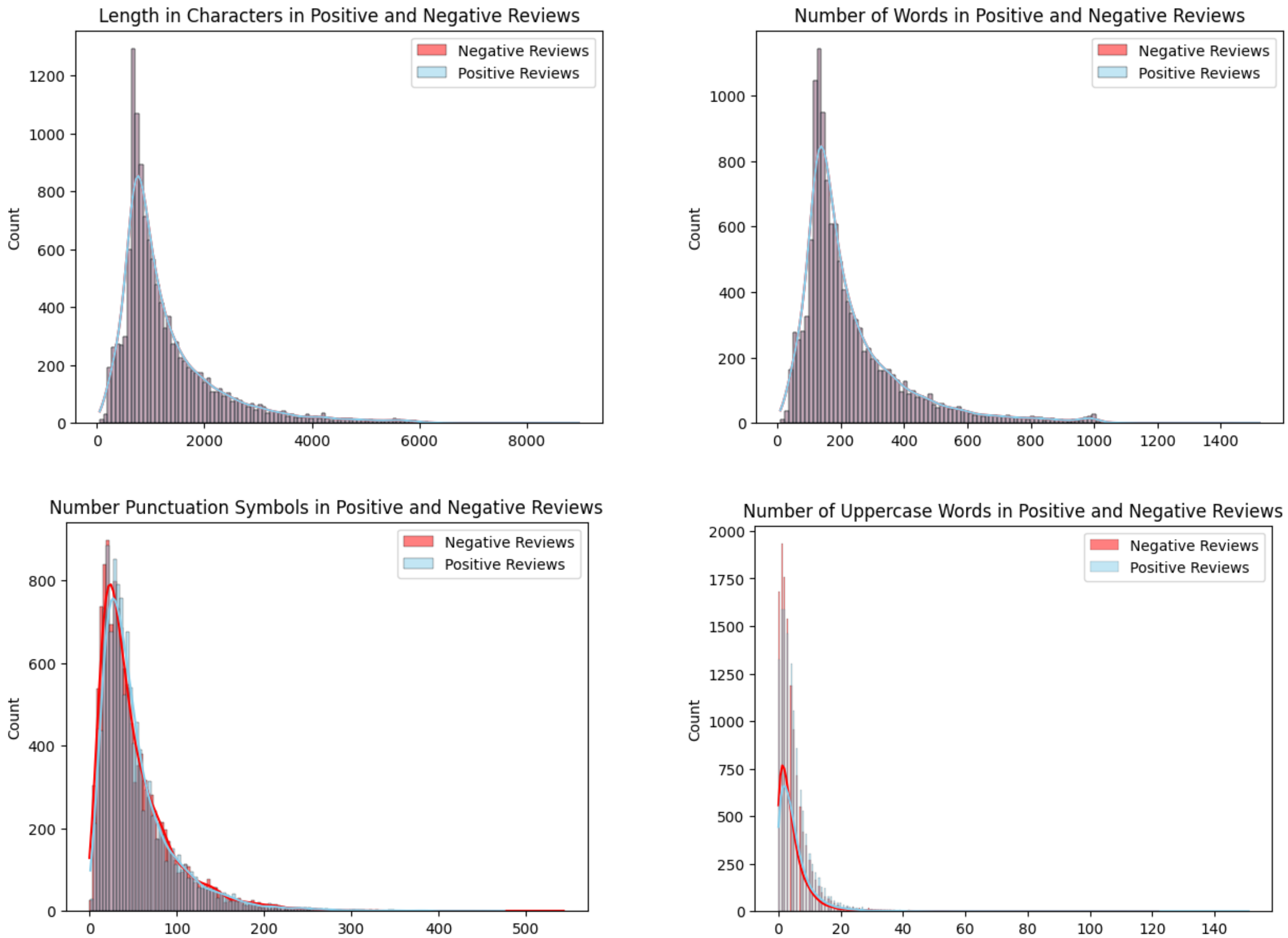

Fig. 2: Analysis of the Reviews in the Dataset

each column contains unique words of said reviews. Each cell of the matrix contains the frequencies that each word appears in their respective reviews. Lemmatization is added in order to combine similar words to minimize variance and simplify the model's training.

As for BERT, the data needs to be tokenized in order to be inputted into our model. As opposed to the Naive Bayes model, BERT can take sentences as input sentences. This allows the model to understand the context where each token is split using special tokens such as [CLS] and [SEP] which respectively indicate the start and end of a sentence. These tokens are converted to numerical IDs based on a vocabulary file.

III. Results

A. Experiment 1

After training both models on the IMDB review training dataset, we compare the accuracy of both models on the testing set. As shown in *Table I*, the accuracy of Naive Bayes is 87.73% compared to the higher accuracy of BERT of 90.62%. In its training, the Bert Model reported a loss of 0.23. We plotted the accuracy of the BERT Model at each training step within one epoch. In the beginning, the accuracy was heavily fluctuating, but it settled down and started steadily increasing after 100 steps of training. BERT's training was done using an Adam optimizer, and gradient descent with a mini-batch of 8. Our graph also shows that the accuracy could possibly improve even higher with more epochs, but for the purposes of this experiment, one was good enough due to the long run-time of the model.

TABLE I: Accuracy of Both Models

Model	Accuracy(%)
Naive Bayes	87.73
BERT	90.62

For the Naive Bayes model, we were able to deduce the tokens with the most important weights. In other words, the model was trained to believe that these tokens had the highest influence on the sentiment of the review they were found in. On *Fig. 4*, the 20 most important weights are illustrated. The weights were sorted by determining which weights had the highest impact on the probability that a review had a positive or negative sentiment. A word's presence in a review was also determined to be more important than its absence by a factor

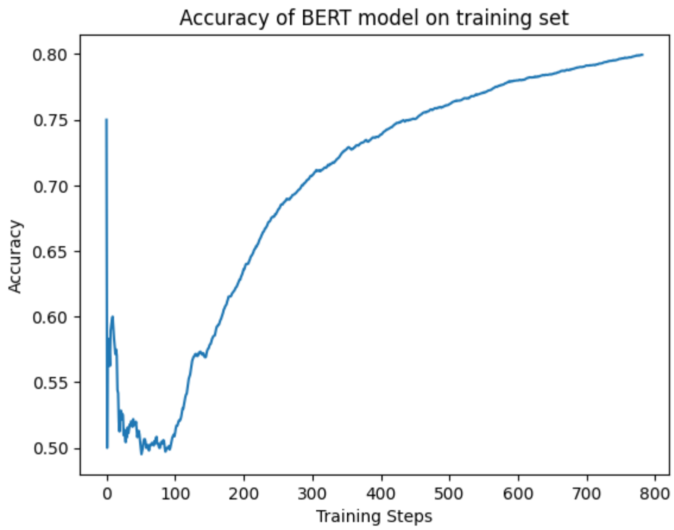

Fig. 3: Validation accuracy of Bert Model

of 100.

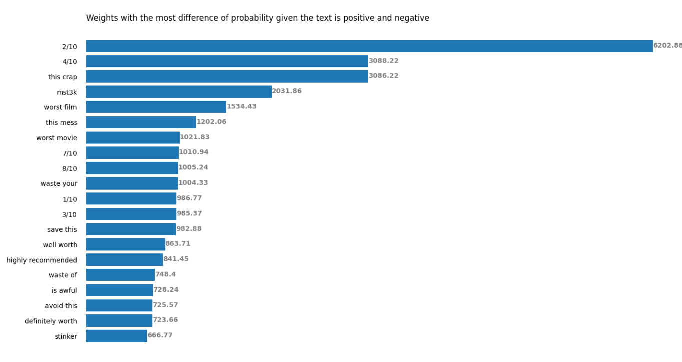

Fig. 4: Tokens with the highest weights for sentiment analysis

The confusion matrix of the Naive Bayes model was generated as seen on *Fig. 5*. We achieved an accuracy of 0.88, a precision of 0.88, a recall of 0.87, and an f1 of 0.88.

B. Experiment 2

The BERT model consists of 12 layers. Each layers contains 12 attention heads, and each attention heads contains a 512 x 512 self-attention matrix. Each attention matrix's purpose is to associate a weight between pairs of tokens. This weight is responsible for classifying the review's sentiment by considering not only the tokens of the review, but their relative position.

In this experiment, we analyze a mini-batch of 8 reviews. For each of these reviews, we feed them as inputs to the trained BERT model. Finally, we access a specific layer attention matrix of the model for each of these reviews. To access said attention matrix, we choose a specific layer and a specific attention head of said layer. In our case, we chose the second attention head of the second layer's attention matrix. We analyzed the self-attention matrix of two correctly and two incorrectly classified reviews. The three pairs of tokens with

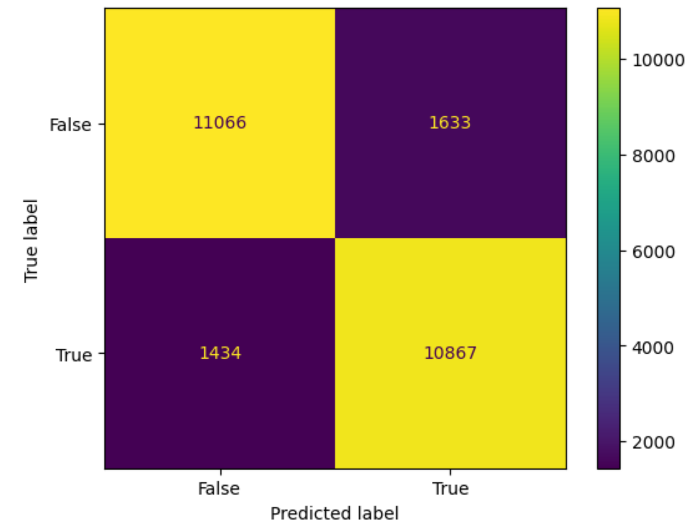

Fig. 5: Attention matrix for correct models

the largest attention weights were outputted for each of the analyzed reviews. The most relevant weights are dark green in the figures below. Each row and column of these matrices represented one of the 512 allowed tokens for each self-attention matrix. The corresponding pair of tokens associated with these highest weights can be found at the end of the newbert collab. An interesting example for a wrongly classified review comes with the pair of token "an excellent". This review had a negative sentiment. However, the last sentence of said review denoted that the movie had "an excellent soundtrack". Consequently, our BERT model wrongly associated this review with a positive review.

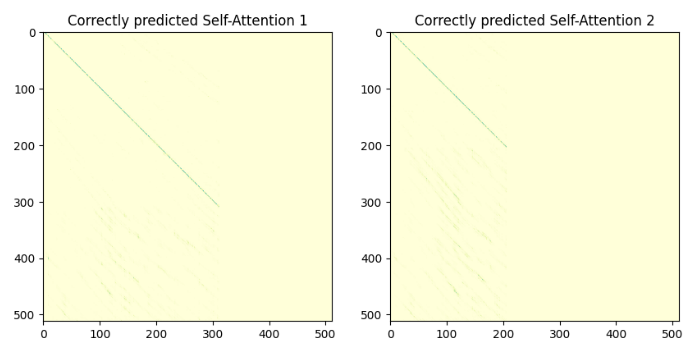

Fig. 6: Attention matrix for correct models

C. Extra Experiment 1

We wanted to examine the effect that varying the number of features has on the accuracy of the Naive Bayes model. We observe in *Figure 8* that as we increase the number of unique words included in the vocabulary list, the model's accuracy increases. The maximum number of features that we could train our model with was 8000 features, before we ran out of RAM, causing a crash. As a result, we used the highest amount of features possible when finalizing our Naive Bayes Model.

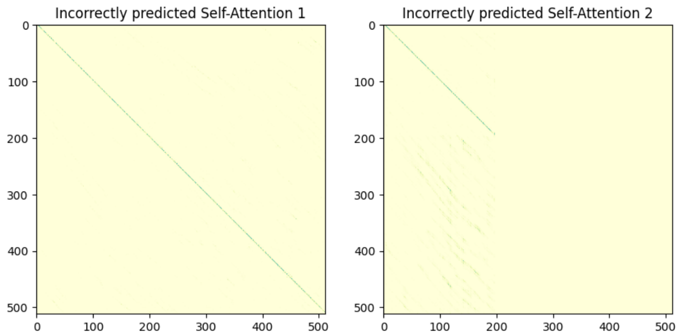

Fig. 7: Attention matrix for incorrect models

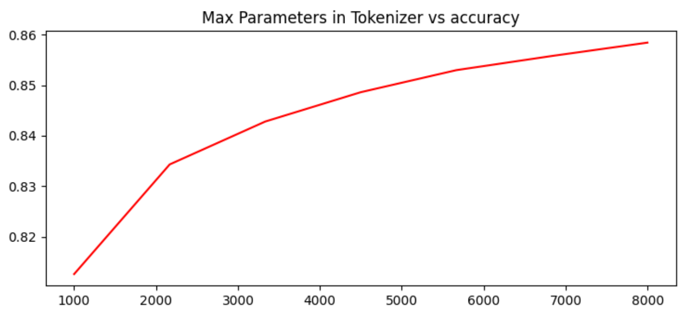

Fig. 8: Accuracy vs Number of Features

D. Extra Experiment 2

For another extra experiment, we observed the effect of varying the N-gram ranges in our Naive Bayes model. As illustrated in *Figure 9*, the accuracy is highest when using N-gram ranges of (1, 2) and (1, 3). At a close second, (2, 2) and (2, 3) also produce high accuracy.

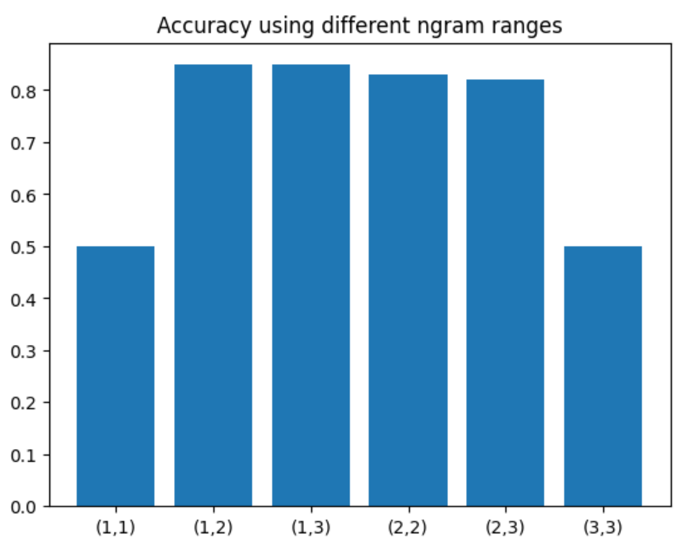

Fig. 9: Accuracy vs Different Ranges of N-grams

E. Extra Experiment 3

We originally trained our Naive Bayes model without lemmatizing its input tokens. In this extra experiment, we trained a new model using it. We found that it increased the performance of the model by around 0.6%. In fact, the model without lemmatization yielded an accuracy of 0.871Lemmatization is the process of reducing words to their base or dictionary form, which is called the lemma. A lemma is the canonical form of a word that represents its meaning, and it is typically the form that you would look up in a dictionary.

The goal of lemmatization is to convert words that have different inflected forms (such as ”run,” ”runs,” ”running,” and ”ran”) into a common base form (in this case, ”run”). This makes it easier to analyze and compare the meanings of words in a text because all variations of the same word are treated as the same word. To implement lemmatization in our Naive Bayes model, we used the Natural Language Toolkit (NLTK) in Python.

F. Extra Experiment 4

We modified the value of alpha which is a hyperparameter that modifies the prior n the Naive Bayes model to see if a different value would optimize its accuracy. The prior represents the probability distribution of the tokens and the sentiment associated with them that we think is the true probability before training the model. The value of alpha made no difference in our results as seen on *Figure 10*. That is likely due to training our model eventually yields low importance to the prior as the posterior probabilities dominate the model's predictions.

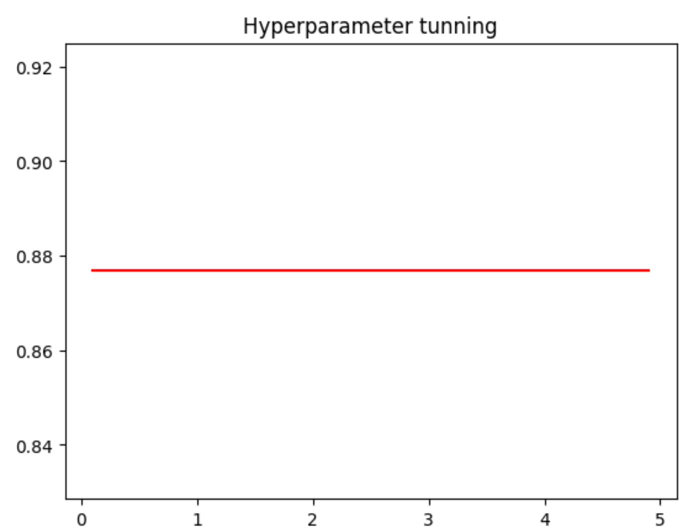

Fig. 10: Accuracy vs alpha

G. Extra Experiment 5

We experimented with different stop-words. To slowly improve our model, we chose our stop-words judiciously. Online resources [**saha2021basics**] recommend we use words such as ”all”,”in”,”the”,”is”,”and”. However, when we removed these words from our model, it reduced our accuracy by 0.5%. We are using bi-grams when training our model. As a result, we mostly tried to remove punctuation when training our final model.

IV. Discussion

We trained a Naive Bayes and a BERT model for sentiment analysis. The Naive Bayes model was written from scratch while the BERT model was fine tuned from a model with pre-trained weights. In this project, we obtained empirical evidence that BERT is more accurate in sentiment analysis for IMBD movie reviews. This is due to BERT's ability to grasps more complex relationships between words in a sentence when it comes to analyze the sentiment associated with sad sentence. On the other hand, Naive Bayes is only trained on the frequency of specific word and the sentiment of said word in a review. In theory, it sounds like both model should perform approximately the same, but sentences are more complex than a simple sequence of words. The position of each word in a sentence plays a large role in the feelings it conveys. For example the sentence "The customer service representative was very helpful and kind, but the product was terrible." Naive Bayes would probably classifies this sentence with a positive sentiment as the words "helpful" and "kind" are found in it. However, the context in which said words were used can clearly show that the sentiment is actually negative. As humans, we can understand this sentiment because we read not only the sentences, but the context in which they appear. BERT's infrastructure is designed to allow such complex relationships to be uncovered and trained.

These complex relationships between tokens that BERT understands are incomparably more computationally expensive to train. In this project, training our Naive Bayes model took approximately 5 minutes. Our BERT model, while it already had pre-trained weights, would take about 11 hours to fine-tune to the IMBD movie reviews dataset. Fortunately, using a GPU hardware accelerator, we could reduce this training time to about 11 minutes, assuming we had resources to spare.

Depending on the particular job and dataset being utilized, there may or may not be a performance difference between deep learning (BERT) and conventional machine learning (Naive Bayes) techniques. Deep learning techniques are typically more successful at problems involving huge volumes of unstructured data, such as speech or image recognition. On the other hand, traditional machine learning approaches might be more successful for problems involving structured data, such as tabular data or data that has already undergone preprocessing.

Several NLP tasks such as sentiment analysis, machine translation, and text synthesis, have seen substantial performance advances with deep learning techniques. These methods often require large amounts of training data and computational resources, but they can be highly effective for complex language tasks that are difficult to solve with traditional machine learning methods.

V. Conclusion

Sentiment analysis is the task of evaluating whether a text is associated to a positive or negative sentiment. Traditional Machine Learning techniques such as the Naive Bayes model can be used to complete this task. However, deep learning models such as BERT are more suited for these tasks. That is because NLP tasks often require models that can grasp the more complex underlying relationships between input features to the model. In sentiment analysis, the input features are the words found in the analyzed text. In this experiment. we concluded that BERT was more accurate for sentiment analysis on the IMBD review dataset. However, training the model requires much more resources for only a 3 percent improvement. Perhaps we would be able to reach a similar accuracy by exploring more parameters of the Naive Bayes model.

VI. Statement of Contributions

Anna designed and ran the experiments on the Naive Bayes mode. Yajiv designed and ran the experiments on the BERT model. Eric interpreted the results. We all wrote the report.

References

[1] Saha, S. (2021, April 2). Basics of CountVectorizer. Towards Data Science. Retrieved from https://towardsdatascience.com/basics-of-countvectorizer-e26677900f9c

[2] Jha, A. (2021, August 11). BERT testing on IMDB dataset- extensive tutorial. Kaggle. Retrieved from https://www.kaggle.com/code/atulanandjha/bert-testing-on-imdb-dataset-extensive-tutorial