GITHUB LINK: https://github.com/johneckberg/CS2611-final-project

CSC2611 Final Report

DJ and John

A. Abstract

The objective of this project was to create a model that could accurately predict the sentiment of a given text. To do this, the model was trained on a Twitter dataset, consisting of four columns: a unique text ID, selected text from a tweet, the entirety of the text from a tweet, and sentiment of the tweet – the latter two being the only two relevant to the training. Using a BERT model, we were able to achieve a precision and recall of roughly 80%.

B. Introduction

Text sentiment analysis is a Natural Language Processing technique where text is analyzed and labeled against a set of criteria. The analysis consists of breaking down the text into components, such as sentences, phrases, tokens, and parts of speech, as well as looking at the general context in which each of these terms/phrases are placed and then assigning each a sentiment score. In this case, our approach was to determine whether a given text was "positive," (0) "negative," (2) or "neutral" (1).

With Twitter being a hub of mixed sentiments, we thought a dataset based on tweets posted would be an interesting database to train our model. The dataset used was one provided by the site Kaggle, containing over 27 thousand tweets, established for Kaggle competitions. This dataset contains four columns: the first specifies a unique text ID, the second includes text taken from tweets, the third is made up of selected from the previous column, and the last details the corresponding sentiment label for the text listed. For this specific problem, only the second and last column were actually used for the training and testing of the model.

To train and test the model, we had to first get a hold of the pre-trained data, which was recorded by Google, where they divided the pre-training across multiple GPUs over the span of 72 hours. We then had to proceed with fine-tuning, which played a significant role in boosting accuracy, precision, and recall. Finally, we had to assess whether the model was overfit and whether improvements to the results could be made.

C. Baseline Model

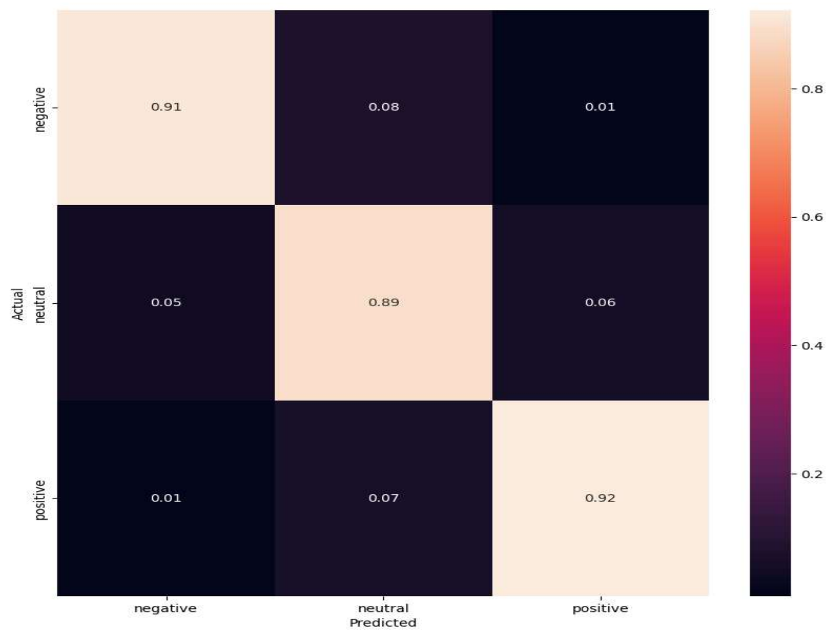
A good direction for this task would be to learn similar representations for the words that appear close to each other more frequently in our set of documents or sentences. However, there are a couple things to be careful about. Sometimes the meaning can be totally different (for example, bank as a financial institute vs bank of the river) The word bank has *multiple word senses*. Ideally, we want to learn **sequence-level semantics.** If a model could take in a whole sequence of words, we could "cover up" a set of words to predict. This lets us learn similar representations for words that appear in the same context more frequently, rather than just learning words that are close more frequently. This means we can learn sentence level information rather than just word level information. This is called Masked Language Modeling. This objective is used to train a Bert model.

A Bert model is a Transformer encoder. Transformer models are a type of neural network architecture designed for sequence-to-sequence tasks, such as natural language processing and machine translation. The nice thing about sequence-to-sequence models is they work by converting an input into a latent representation, then into an output. If we can just take this latent representation, we have a sentence embedding model.

The key component to the transformer encoder is self-attention. We can think of self-attention as a mechanism that enhances the information content of a token by including information about the token's context. In other words, the self-attention mechanism enables the model to weigh the importance of different tokens that make up an input sequence and learn to adjust their influence on the output. This is especially important for language processing tasks, where the meaning of a word can change based on its context within a sentence or document.

The Bert model [Devlin J, et. Al. 2018] is trained on two tasks; the MLM task we just discussed, and NSP, or next sentence prediction. This is done by first, passing two sentences into the bert model and then adding a special CLS token to the beginning of the total sequence. Then this CLS token is used to predict if the two sentences are next to each other (like two sentences on a wiki page). These two tasks are then learned on billions of sentences of unstructured text data from Wikipedia and other sources. The original Bert model uses over 3 billion training sentences and took Google over 72 hours on 64 TPU's (Google's custom hardware for large ML models). This pre-trained knowledge is then used for downstream tasks via fine-tuning.

In the context of our classification problem, we simply add a linear layer to the output of the CLS token to act as a classification layer. This model was fine-tuned on our Twitter dataset with cross-entropy loss, for 3 epochs, with a learning rate of $2*10^{-5}$ and a batch size of 16 (8 on 2 gpus). We got a precision of 0.79 and a recall of 0.79.



D. Model Improvement Approaches

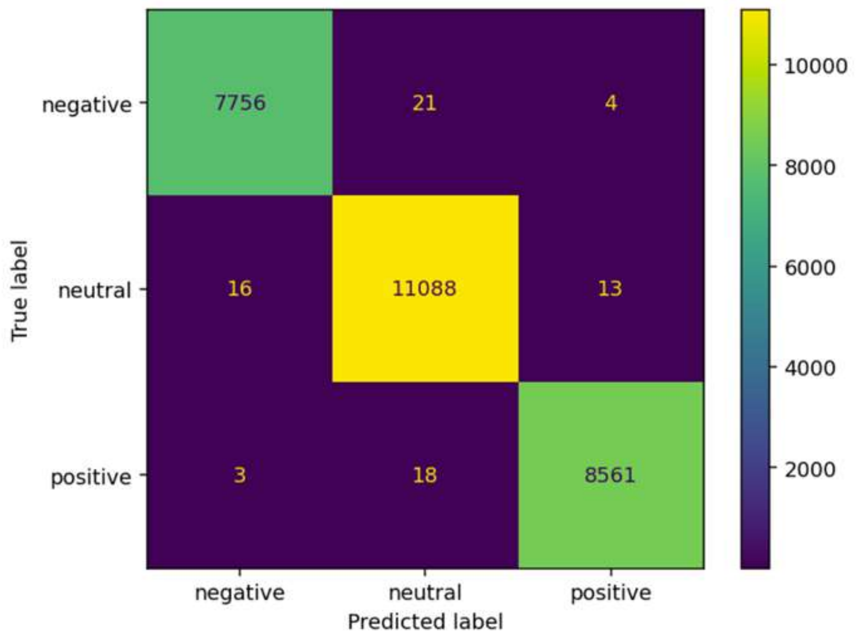
Next, we investigated ways to improve model performance. Bert models have been shown to perform better with more epochs. [Mosbach et. Al. 2020] found that despite achieving close to zero training loss, over-fitting is not an issue during fine-tuning. This means training longer can improve both performance and stability. Another point of improvement is that Bert models like larger batch sizes. A well cited paper on training transformers [Popel M, Bojar O, 2018] recommends 12k tokens per batch. We can't hit that on our hardware, but we try to get as close as possible. To investigate the effectiveness of these strategies, we perform a grid search on learning rates $2*10^5$ and $3*10^5$ batch size per device (4 devices) of 8, 16, 32, epochs of 3,10,20. We found no significant improvements when performing this grid search.

BERT pre-trained representations have also been widely studied using model probing methods, and it was also shown in [Mosbach et. Al. 2020] that the pre-trained features from earlier layers are more transferable than later layers. When re-initializing the last two blocks of the 6 block Bert network, we also saw no significant improvements. We can also try changing out the base model; we discussed the original version of a Bert model, but there are now many examples of Bert family models. We used both MPnet [Song K, et al. 2020] and base Bert, and saw no significant improvement.

Given our data size, this failure to learn could be caused by our model consistently getting stuck in a local minimum. However, even over 40 epochs, our loss continues to get smaller at a consistent pace, so this hypothesis does not make sense. The more reasonable conclusion may be that approximately 20,000 sentences are not enough to train up a 110 million parameter model to higher levels of performance.

E. Results/Model Comparison

After optimizations and fine-tuning were performed, the baseline (BERT) model achieved a fairly high accuracy of around 80%. On the other hand, the comparison (Random Forest Classifier) model was only able to reach an accuracy of just above 70%. The confusion matrix graph below illustrates the wholistic performance of our comparison model. The discrepancy between the two can be due to a multitude of factors.



For one, Random Forest Classifier models are typically unable to capture the semantic and syntactic nuances of natural language, such as word order, context, and ambiguity, something characteristic of the BERT model.

BERT models' high accuracy in tasks centered around Natural Language Processing comes from its "bidirectionality." Whereas most models can read a text from either left-to-right or right-to-left, BERT can read a text (or sequence of words) all at once, with no specific direction. (As discussed in the "Baseline Model" section, the process is very much fascinating). This allows it to understand the meaning of each word based on context from both sides of the word, an advantage it has over other models, including RFC. With our expertise, finding a model that could beat it would have been quite a surprise.

F. Conclusion

The goal of this project was to compare methods for classifying the sentiment of tweets and build the most performant model we could. To do this, we fine-tuned a base size Bert model for sentiment classification on a dataset of 26,000 tweets. We achieved an initial model recall of .79 and a precision of .79. We struggled to increase model performance past our initial benchmarks. We determined that this could mostly be attributed to the size of the data, and the cleanliness of the neutral category.

Regardless, we determined contextual representations perform better on this sentiment classification task, and leveraging pre-trained models is best for small datasets. While fine-tuning Bert models is straightforward, maximizing their performance can be challenging. Nevertheless, they demonstrate superior performance in sentiment classification tasks. To further explore, it would be valuable to conduct additional tests to assess whether utilizing the Bert large version enhances performance on our dataset. Additionally, a larger, meticulously labeled dataset comprising tweets and their corresponding sentiment labels would be necessary for further study.

Citations:

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. North American Chapter of the Association for Computational Linguistics.

Mosbach, M., Andriushchenko, M., & Klakow, D. (2020). On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines. ArXiv, abs/2006.04884.

Song, K., Tan, X., Qin, T., Lu, J., & Liu, T. (2020). MPNet: Masked and Permuted Pre-training for Language Understanding. ArXiv, abs/2004.09297.