Generalized Sentiment Analysis

Josef LaFranchise, Sashank Reddy Vasepalli, Shreya Sudhanva

Why we picked this project up:

We set out to build a versatile sentiment analysis model, aiming to provide accurate predictions on different types of data. The main advantage of such a model is its ability to adapt easily to new data, having lesser biases about certain traits in the data, and the savings in time and cost from not having to train new models for specific tasks. The domain of generalizable modeling is interesting due to the difficulty involved in training them effectively, and because of the impact these models can have on a broad range of fields.

Traditional machine learning models are heavily biased to the data they are trained on, and often fail to generalize to new types of data. For example, if a sentiment analysis model was trained on unregulated amazon reviews, and then was asked to determine the sentiment of a research paper on the relation between bananas and radioactivity, it's likely to fail rather than succeed most of the time. Hence, we set out to train multiple transformer models on different datasets, and combine this with a fine-tuned Bidirectional Encoder Representations from Transformers (BERT) to obtain a versatile model.

Our Data:

For our task, we made use of 3 different sentiment classification datasets. Each of these datasets is filtered to contain two labels, Positive (1) and Negative (0). The first dataset was the Twitter Sentiment Analysis dataset from Kaggle, and was the largest dataset at around 1.6 million tweets. This was the most general of the datasets and was therefore the one we used to train our models. The second dataset, ChatGPT sentiment analysis, was a collection of tweets that were specific to discussion of chatgpt, as from Kaggle, with each tweet having a positive, negative, or neutral label. Since our transformer model was designed for binary classification, preprocessing was performed on this dataset to remove tweets with a neutral label. After this preprocessing step, the dataset contained around 200,000 tweets. The third and final dataset we used was the financial phrasebank dataset from Hugging Face. This dataset contains sentences from financial news along with sentiment labels. Like the chatgpt dataset, preprocessing was performed to normalize the labels. After normalization, the size of the dataset was 4844 sentences.

For the BERT model's training, we cleverly combined all three datasets to prevent the domination of any singular dataset. We sampled 16000, 14780, and 17543 rows from the tweet sentiment, financial phrasebank, and chatgpt datasets respectively. This was saved as the "combined_output_csvSmall.csv" file. We ended up with 18,301 samples for Class '0'/Negative and 16,612 samples for Class '1'/Positive.

Prior to using these datasets, we have preprocessed them to convert labels into a machine-friendly format [0, 1], and used nltk's word tokenizer to break down sentences into tokens and convert them into integers. Finally, we also created a data_generator to feed the data in batches to our ML models (specifically for BERT).

Our Models:

To establish a baseline, we trained sklearn's logistic regression model on the tweet dataset and evaluated its accuracy on the datasets. For the features, we took the counts of the first 10000 tokens as a vector for each datapoint. We obtained the results shown in the table below.

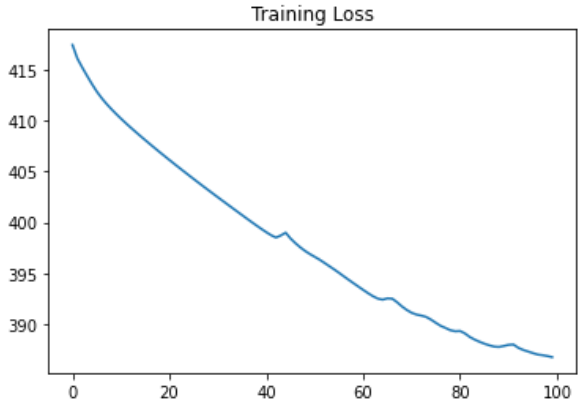
Dataset	Accuracy
Tweet Train	0.897
Tweet Test	0.758
Chatgpt Dataset	0.474
Financial Dataset	0.632

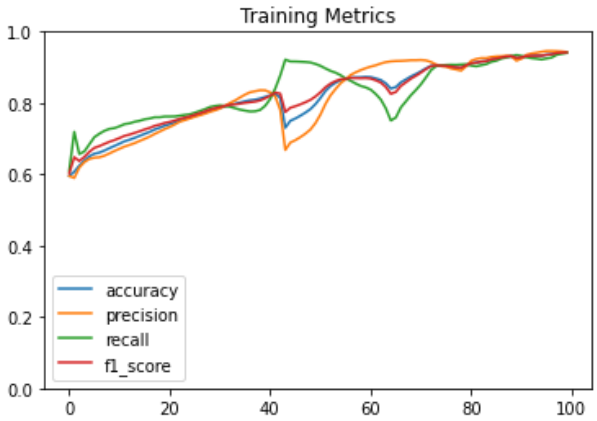
For creating our transformer model from scratch, we leverage the pytorch library. We based our model off of the encoder-decoder model presented in the paper "Attention Is All You Need" [1], disregarding the decoder part of the model as it was not necessary for the task sentiment analysis. Each encoder layer consisted of a multihead self-attention layer followed by a position-wise feed-forward layer. The output of the final encoder layer is flattened and passed to a linear layer followed by a sigmoid layer to get the final classification label. The parameters of the final model were as follows: an embedding size of 52, 4 attention heads, 4 encoder layers, and a feedforward layer size of 128.

As for our LLM, we import the Tensorflow pre-trained BERT model using the 'transformers' library. BERT, a transformer-based model, learns bidirectional contextual relationships in language by considering the context of words from both directions within a sentence. It also comes with a BertTokenizer, which helps us further by allowing us to generate input sequences in the correct format for the model's fine-tuning. BERT contains 109,482,240 trainable parameters. However, to perform fine-tuning on the model, we will be freezing the entire BERT model's weights to prevent it from losing important values from the backpropagation of our FFNN used for fine-tuning. The FFNN has 2 consecutive layers of 50 ReLU nodes, followed by a sigmoid to perform binary classification.

Results:

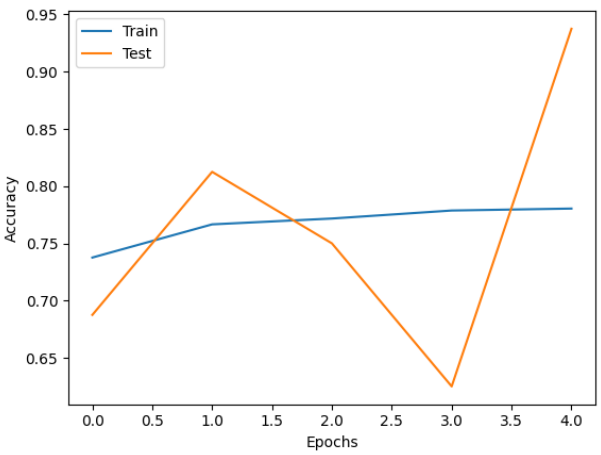
We trained the transformer model for 100 epochs with a batch size of 64. The test metrics are shown in the figures below.

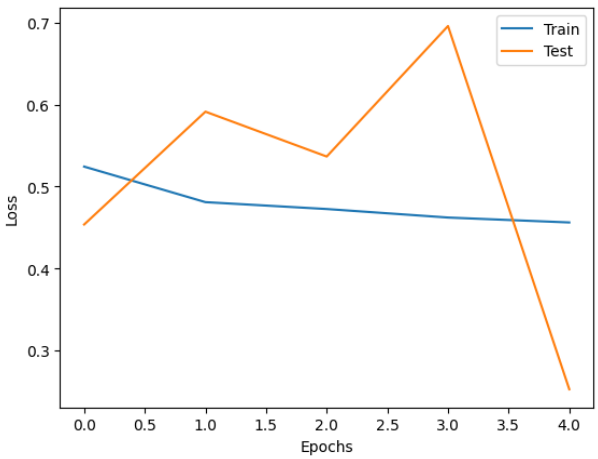




Dataset	Accuracy	Precision	Recall	F1
Tweet Train	0.928	0.942	0.914	0.928
Tweet Test	0.656	0.675	0.616	0.644
Chatgpt Dataset	0.478	0.315	0.445	0.369
Financial Dataset	0.465	0.723	0.396	0.512

With our transformer, we scored higher than the baseline on the training set but performed worse on the test set. This suggests that our model may have overfit on the training data. For the ChatGPT data set, our model performed slightly better than the baseline. We expected that our model would perform better on the Chatgpt dataset than on the financial dataset since the former consisted of tweets like our training set while the latter did not. While this did happen, the difference in performance was not that great.





We trained our BERT model for 5 epochs. After the fine-tuning, we achieved an accuracy of 93.75 % on the validation dataset. This was after we cherry-picked the best-performing model based on validation loss. The test accuracy of the BERT model was significantly greater than all the previous transformer models combined, despite being trained on a mixed dataset. This is most likely due to the pre-trained knowledge representations that BERT contains.

Dataset	**Accuracy**	**Loss**
Train	0.9375	0.2525
Test	0.8012	0.4191

Conclusion:

After our experiments, we proved that LLMs are indeed powerful tools in the field of AI. They excel at almost any task, as long as they are sufficiently fine-tuned for it. We also show that the accuracies achieved by the LLM model on a harder mixed dataset and with fewer training epochs further strengthen our case of utilizing them for generalizable tasks.

Future Work:

Currently, we have used BERT as our Large Language Model. However, BERT was introduced by Google back in 2018 and is considered an older technology due to the speed of advancements in the field of AI. Our original plan was to use LLaMA 2.0 as our LLM and then use QLoRA (Low-Rank Adaptations) to fine-tune it. However, we had a lot of compatibility mismatches with the model and the libraries it was built on and ended up being unable to train it locally. Hence we would love to incorporate larger LLMs like LLaMA into our system in the future.

Additionally, we would also love to explore the concept of ensemble modeling. By efficiently combining the outputs of both the domain-specific transformers and a 70B+ parameter, generalized language model, we are sure to attain accuracies nearing the human baseline.

Works Cited:

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uskoreit, Lion Jones, Adian N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. *arXiv preprint arXiv:1706.03762*, 2017.
[2] Language Modeling with NN.Transformer and TORCHTEXT. PyTorch. https://pytorch.org/tutorials/beginner/transformer_tutorial.html

[3] Arjun Sarkar. Building a Transformer with PyTorch. Datacamp.com. August 2023.
[4] Luca Massaron. Fine-tune Llama 2 for sentiment analysis. November 2023.
Fine-tune Llama 2 for sentiment analysis | Kaggle