A Comprehensive Analysis Of Twitter Tweets Using SOTA Models

Raja Batra, Eli Rejto

***Abstract*— Sentiment analysis is a technique used to identify the tone of a piece of text. This report outlines various machine learning approaches to conducting sentiment analysis on Twitter Tweets. The specific models built and tested are Transformers, LSTM's, and the foundation models GPT-3.5 and GPT - 4**

I. INTRODUCTION

In a digital world, sentiment analysis is an important tool that can be used to extract emotions from text. This paper explores several methods for deciphering sentiment within Twitter's 280 characters. Utilizing Kaggle's Sentiment-140 dataset [3], categorized into positive and negative tweets, our research employs advanced models to interpret Twitter sentiments: Transformers, LSTM's, GPT 3.5 and GPT 4. We start by exploring the dataset and then detail the construction of the transformer and LSTM models. Our focus then shifts towards investigating the impact of in-context learning and prompt input sequences on foundation models.

II. DATA

A. Dataset and preprocessing

The dataset utilized in our analysis was the kaggle sentiment 140 dataset [3]. The dataset had 6 fields: target, ids, date, flag, user, and text. For the purpose of our research we filtered the data to contain the text, which was the text of a tweet, and target, which was the polarity of a tweet. As shown in figure 1, the dataset contained 1.6 million tweets that were evenly split between positive and negative tweets.

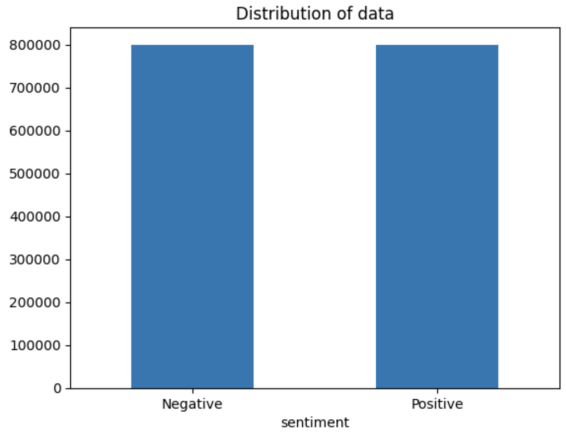

Figure 1. Distribution of Tweets

The first step in processing our data was adjusting the labels. Each tweet had a target which is a 0 if it is negative and a 4 for positive. The adjustment made was redefining the positive label to have a value of 1 instead.

To process the text of the tweets, we utilized various techniques. For the transformer model, we attempted word level tokenization as well as subword tokenization. For the LSTM, we used word level tokenization and for the foundation models, we used the raw text.

The first technique we utilized was word level tokenization. To do this we processed each individual tweet so that all the URLs were replaced with 'URL', emojis with 'emoji,' and usernames with 'user.' We then lemmatized the text and removed stopwords from the text using the open source python package nltk [6]. After, the remaining text was lowercase and filtered to only contain alphanumeric characters. For example, the following tweet "@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it." was converted into "user url aww bummer shoulda got david carr third day." Processing the 1.6 million tweets took a total time of 88 seconds using the google colab cpu.

Once the text was filtered, we built a vocabulary for the text using the pytorch feature torchtext [4]. With torchtext, a dictionary of the 20000 most common words in our dataset was created. Each word had a corresponding index. Using this vocabulary, the text in each tweet was mapped to its corresponding integer. The list of integers was padded so they were all the same length and ultimately converted to a pytorch tensor so that it could be stored in a pytorch DataLoader.

The second technique utilized was subword tokenization. Subword tokenization is the process of breaking down words into smaller units based on their frequency. This was done following a hugging face tutorial to build a subword tokenizer from scratch [2]. The first step was to normalize the text using the prebuilt BertPreTokenizer[1]. Our tokenizer was then trained on the text file wikitext-2 [1] to create a dictionary of 25000 tokens. We then encoded our twitter texts using this tokenizer so that the text was mapped to the new subwords we created. For example, the text "let's test this tokenizer" was converted to ['[CLS]', 'let', "'", 's', 'test', 'this', 'tok', '##eni', '##zer', '.', '[SEP]']. Using the encode function of the tokenizer, it took 144 seconds to process all of our tweets. The tokenized string was then mapped to the tokens corresponding integer. The encoded text was converted to a pytorch tensor and stored in a pytorch dataloader along with its label.

B. Data Split

To train the transformer and LSTM models, we used 20 percent of our data (320000 tweets). This smaller dataset was randomly selected while maintaining an even distribution of positive and negative tweets. From this subset, we used the sklearn library [5] to split our dataset into 80 percent training, 10 percent validation, and 10 percent test. The encoded text and labels were stored in their respective pytorch dataloaders with 128 tweets per batch.

III. System Design

The next section will explore the three different model approaches we used. Our first approach was a custom transformer, followed by an LSTM, and ultimately foundation models.

A. *Transformers*

The first approach attempted was building a transformer model. Our model utilized an embedding layer, followed by a positional encoding layer, transformer layer, and linear layer. The transformer layer utilized the architecture from the paper "Attention is all you need"[1] as shown in Figure 2.

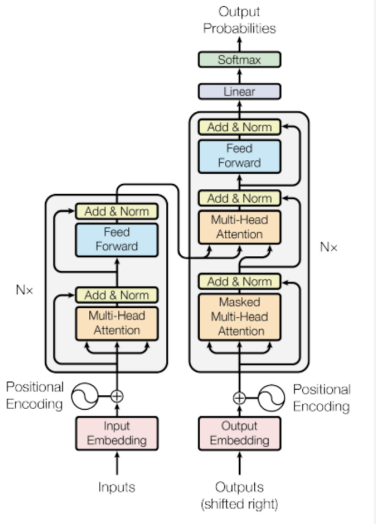

Figure 2. Transformer Block Diagram [1]

Using our training and validation data, we tuned our model and ultimately settled on a model size with 5 heads, 2 layers, a hidden dimension of 80, and an embedding dimension of 40. In addition, during training, we implemented gradient clipping, weight bias, and added dropout layers.

We first trained the model on my processed data that utilized word level tokenization. As seen in figure 3, the model was able to train, as the loss decreased and eventually converged.

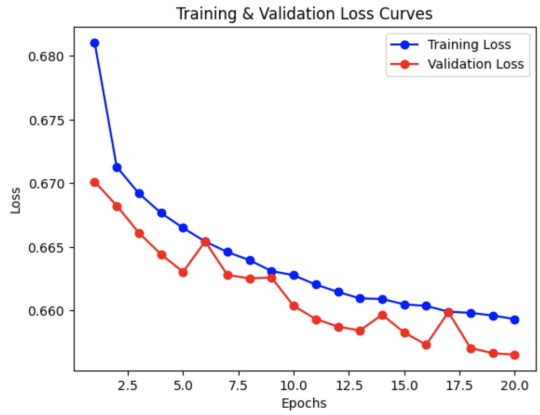

Figure 3. Transformer loss curve with word tokenization

The model took around 79 seconds per epoch to train. While the word level encoded text was able to train, it had a relatively low accuracy. The model achieved its highest validation accuracy of 61.12% and test accuracy of 61.01%.

Our intuition suggested that there were too many words that were only seen a few times. As a result, it was difficult to train the model prompting us to attempt encoding our tweets with subword tokenization. In our second attempt, we utilized the same transformer model, but with the subword tokenization.The training curve for our second attempt at training the transformer is depicted in Figure 4. This model was faster, taking 54 seconds per epoch. As well, we were able to achieve a much higher accuracy. Our validation data reached 80.81% accuracy and our test data had an accuracy of 78.08%.

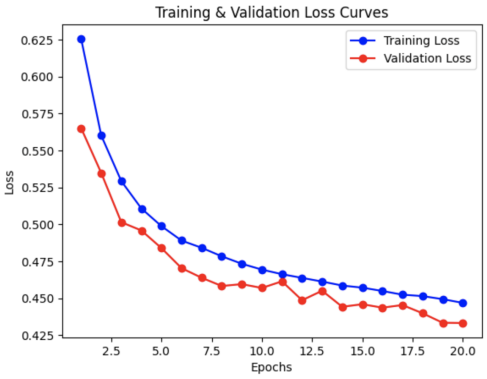

Figure 4.Transformer loss curve with subword tokenization

B. *LSTM*

The second approach we tested was building an LSTM, which is a recurrent neural network architecture. The model which is shown in Figure 5 uses an embedding layer, pytorch's LSTM layer, a linear layer, and a sigmoid activation. The model then returns the output from the last layer to determine the sentiment. To train the model, we experimented with gradient clipping, dropout, different loss functions, and various model sizes. We ultimately settled on our LSTM size to have 2 layers, an embedding dimension of 256, and a hidden dimension of 512.

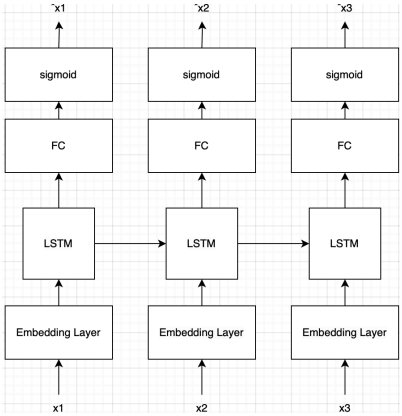

Figure 5. LSTM Block Diagram

This model successfully trained as shown in Figure 6, and took around 40 seconds per epoch to train. As well, it received a peak validation accuracy of 74.6% and a test accuracy of 73.76%

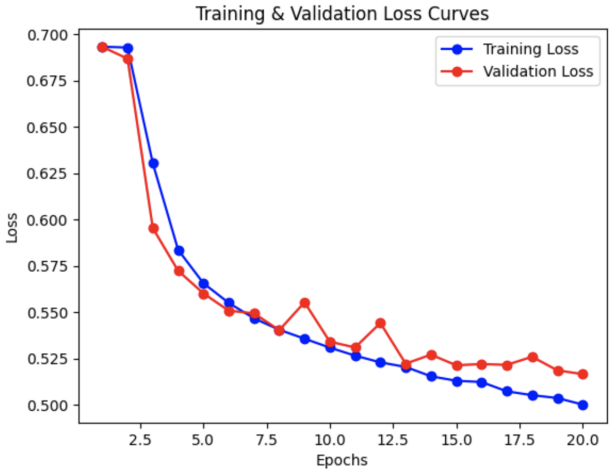

Figure 6. Training Curve for LSTM model

C. *Foundation Model*

The foundation models used for experimentation were GPT-3.5-turbo and GPT-4. Because of openAI's API request limits, the code asks GPT for sentiment analysis on a hundred tweets at a time. The API has limits on API requests per day and tokens per minute. This approach reduces API requests at the cost of using a lot of tokens per minute.

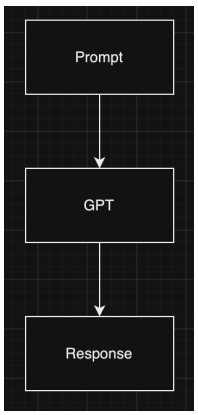

Figure 7. Foundation Model Block Diagram

It took several iterations to create a prompt that forced GPT to output only 1's or 0's for sentiment. The model responded well to prompts that included the words 'binary response'. The prompt that yielded the best results was:

Analyze the sentiment of the following tweets and provide a binary response: 0 for negative sentiment and 1 for positive sentiment. For instance, Tweet 1: 'Just had the best day ever, everything is falling into place! #grateful' should be responded to as 1. Tweet 2: 'Today sucks, everything is falling apart' should be responded to as 0. Ensure that the responses are strictly 0s and 1s. Now, evaluate the sentiment of the given tweets and provide a binary response: 0 for negative sentiment and 1 for positive sentiment. With this prompt, the foundation model was able to get the following results. The foundation model peaks at above the LSTM validation accuracy at about 85%, but has troughs as low as 46%. The overall accuracy of the in context learning without word randomization is aroun 67.3%. With in context learning the overall accuracy got ot around 67,8%. The next experiment done on the foundation model was to run sentiment analysis on tweets with random word order. The words included in the tweet are still the same but their order is randomized. With the same prompt, the following accuracies are achieved.

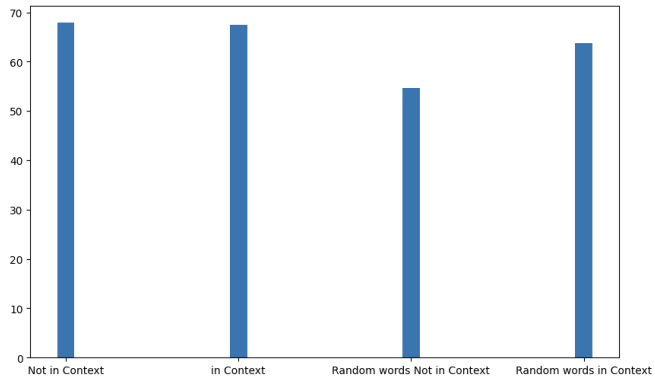

Figure 8. Bar graph of various GPT-3.5 experiments

As seen in figure 9, the word randomization has a noticeable effect on the accuracy of the GPT sentiment analysis as it reaches a peak accuracy of roughly 5-10% lower than so it can reasonably be assumed that word order is a feature in the foundational model**.** Interestingly, the in context learning has a greater effect on the word randomization prompts than the regular word prompts. The experiment on GPT-4 was a non in context learning experiment. We were unable to finetune the prompt for GPT-4 as it was far more expensive to run tests on. As a result, our one attempt performed poorly as seen in Figure 9.

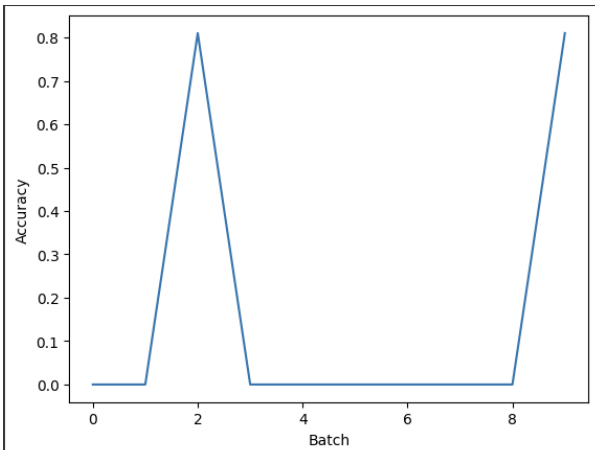

Figure 9. Accuracy of GPT-4 Model

IV. Results and Discussion

Model	Transformer w subword	Transformer w word	LSTM	GPT 3.5	GPT 3.5 (ICL)	GPT 3.5 random input sequence (ICL)
Test Accuracy	78.08%	61.01%	73.76%	67.3%	67.5%	63.8%
Training Time	54 sec/epoch	79 sec /epoch	40 sec /epoch	100 prompts /min	100 prompts /min	100 prompts/min

Figure 10. Model Accuracies and training times

There were three machine learning based approaches for sentiment analysis in this paper: LSTM's, transformers, and foundational models. Originally the transformer approach involved word tokenization which did not yield very high accuracy since the tokenizer only had 1.6 million tweets to train on. On our test data set, the transformer was only able to reach an accuracy of 61%. To improve this, a subword tokenizer was implemented on our dataset. This vastly improved the accuracy of the transformer and made it the highest accuracy model of 81% on our test data set. The transformer model ultimately outperformed the LSTM model which was able to train to 74% on the test data set. Despite the lower accuracy, the LSTM model trained slightly faster than the transformer model. Using the same data, the LSTM took 40 seconds per epoch compared to the transformer which took 54 seconds per epoch. This was likely due to the smaller model size of the LSTM since transformers can leverage parallel computation to increase training speed. The biggest drawback of the foundational models was their inconsistency. Due to the approach of inputting one hundred tweets at a time, oddly if the foundational models determined a negative one sentiment for one tweet, it would often determine a negative one sentiment for all one hundred tweets in the batch. This was the biggest source of inaccuracies in the foundational models and despite extensive prompt engineering, was unavoidable. There was no prompt that fully avoided the negative one sentiment for an entire batch. While foundational models found some success, there were times when accuracy dropped below 50%. For this reason, the foundational approach was the least reliable of the three.

From our three approaches, it was clear that the most effective model was the transformer. Research has shown that humans tend to agree on sentiment around 80-85 percent of the time [8]. The transformer model was the only model to fall into this range. In addition to the transformer model, GPT-3.5 turbo was also able to receive a relatively high accuracy when utilizing in context learning. While the foundation model had a slightly lower accuracy than the transformer, it was much easier to implement and did not require training beyond adjusting the prompt.

In addition to evaluating the various approaches, we used our transformer model to conduct zero-shot sentiment analysis on Elon Musk news headlines. This model utilized the newsapi to extract the most recent news articles related to ElonMusk. These articles were the input to our model and used to determine the percentage of positive headlines. This model was deployed on a Flask website to give insight into how the news currently perceives Elon Musk. significantly without requiring much more training data.

Overall, this project taught us how to implement advanced models and adjust them to work with our data. We explored different processing techniques and training methods. As well, by making our project showed us how to use a model we make and deploy it on a website.The main challenges of this project were processing the data properly to input into the transformer model as well as training the transformer. Fortunately Prof. Tsai gave us the suggestion to use subword tokenization to improve the transformer accuracy.

V. Resources

The LSTM and transformer architecture was built using the Pytorch documentation and tutorials as references[4]. We also learned data processing techniques from the nltk, kaggle, and scikit documentation and examples [6][5][3]. The subword tokenizer was built using the hugging face tutorial [1]. The foundational model code was based off of the chat GPT API tutorial[**7**].

Acknowledgment

We would like to thank Professor Tsai for offering this awesome class and allowing us to improve our skills through this project.

Sources

[1] "Building a Tokenizer, Block by Block - Hugging Face NLP Course." *Building a Tokenizer, Block by Block - Hugging Face*, huggingface.co/learn/nlp-course/chapter6/8?fw=pt. Accessed 13 Dec. 2023

[2] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[3] KazAnova. "Sentiment140 Dataset with 1.6 Million Tweets." *Sentiment 140 Dataset*, kaggle, 13 Sept. 2017, www.kaggle.com/datasets/kazanova/sentiment140/data.

[4] "Language Modeling with Nn.Transformer and Torchtext¶." *Language Modeling with Nn.Transformer and Torchtext - PyTorch Tutorials 2.2.0+cu121 Documentation*, pytorch.org/tutorials/beginner/transformer_tutorial.html. Accessed 13 Dec. 2023.

[5] "Scikit-Learn." *Scikit-Learn*, scikit-learn.org/stable/. Accessed 13 Dec. 2023.

[6] "NLTK Documentation." *NLTK*, www.nltk.org/. Accessed 13 Dec. 2023.

[7] OpenAI. "OpenAI Platform Documentation Overview." OpenAI, https://platform.openai.com/docs/overview.

[8] "Sentiment Accuracy: Explaining the Baseline and How to Test It." *Lexalytics*, 29 Apr. 2022, www.lexalytics.com/blog/sentiment-accuracy-baseline-testing/#:~:text=Wel

l%2C%20for%20one%20thing%2C%20this,model%20trained%20on%20similar%20content.