Sentiment Analysis of Dow Jones Using Multi-Head Attention

Yusuf Elbana
yae13@rutgers.edu

Abstract

In this paper, I uncover how sentiment analysis can help predict movement in the Dow Jones Industrial Average using a multi head attention model trained on Reddit posts. I explore different mechanisms for aggregating contextual information such as mean and weighted sum pooling, in order to enhance the model's performance in capturing relevant sentiment from the Reddit posts.

1 Introduction

1.1 Question

How effectively can multi-head attention predict the trend for Dow Jones Industrial Average (DJIA) based off sentiment analyzed from Reddit posts?

In the last two assignments, we implemented binary classifier models (ham,spam) using Naive Bayes and Logistic Regression. However, neither approach was capable of capturing the contextual relationships between tokens. I plan to deploy a model that can learn the contextual dependencies which in theory should boost predictive power. However my model will be predicting an index's trend, which is seen to be almost "random".

Earlier in the year, the transformer architecture was briefly discussed, which inspired me to utilize parts of it in the model.

2 Motivation

This project is important because it taps into an age-old question: is it going up or down? I personally found the Transformer architecture to be quite daunting at first, especially since I had come across it when learning about models like GPT. The fact that one can design a model capable of understanding the context of words is fascinating, and I hope to leverage that capability to effectively predict the index's movement.

During research, I came across several examples of transformer based models being used for sentiment analysis tasks like movie reviews. See [2] and [4]. Many people scrape platforms like Twitter to track posts from influential people such as Elon Musk, and make investment decision in crypto based on the perceived sentiment of those posts. Although I specifically focus on the DJIA, it raises the question of how effectively can social sentiment act as a financial signal?

3 Method

3.1 Data

The dataset consists of the top 25 Reddit posts scraped from the "r/worldnews" subreddit, within the period 2008 to 2016. Kudos to [1].

3.2 Form

The data can be viewed as tabular, as it's format is CSV. The features consist of the text within each post, as well as the post ranking (number). Date is not a feature since the goal is to predict DIJA's *future* trends.

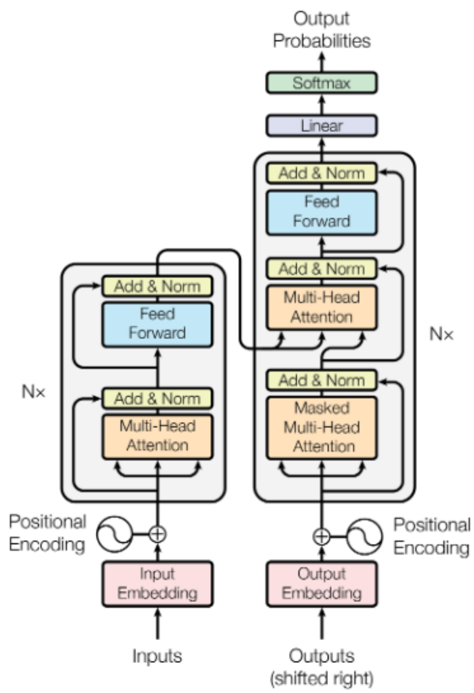


Figure 1: Encoder (left) & Decoder (right) [3]

3.3 Model

The model is a simplified transformer classifier, inspired by GPT. However, it is not designed for next token prediction. Instead, the model is suppose to perform sentiment analysis to determine whether DIJA went up or down on a given day based on the entire sequence of tokens across 25 individual Reddit posts. The difference fundamentally changes the setup. Fanfei Meng and Chen-Ao Wang [2] explored mean pooling to aggregate token information for their sentiment analysis model, which I initially implemented for the both the token and post dimensions. Since mean pooling averages the tokens and posts equally, I believe the model lost predictive power as certain posts/tokens may carry more relevant information, leading to lackluster results. To address this I experimented with other forms of pooling mechanisms analyzed by Jinming Xing, Dongwen Luo, Chang Xue, and Ruilin Xing [4]. For now, the term "pooling" will be used generally when describing the model's flow. I will later explore the results achieved from the various pooling methods.

3.3.1 Feature Space

To reiterate, for each day in the range from $2008 \rightarrow 2016$, there are 25 Reddit posts, each containing a variable amount of tokens. To handle this variability, a unique padding token is used to ensure equal sequence lengths. As a result, the model expects an input tensor of B, P, T, C where B is batch size, P is the number of posts per day, T is the padded sequence length of each post, and C is the embedding dimension.

3.4 Implementation

3.4.1 Preprocessing

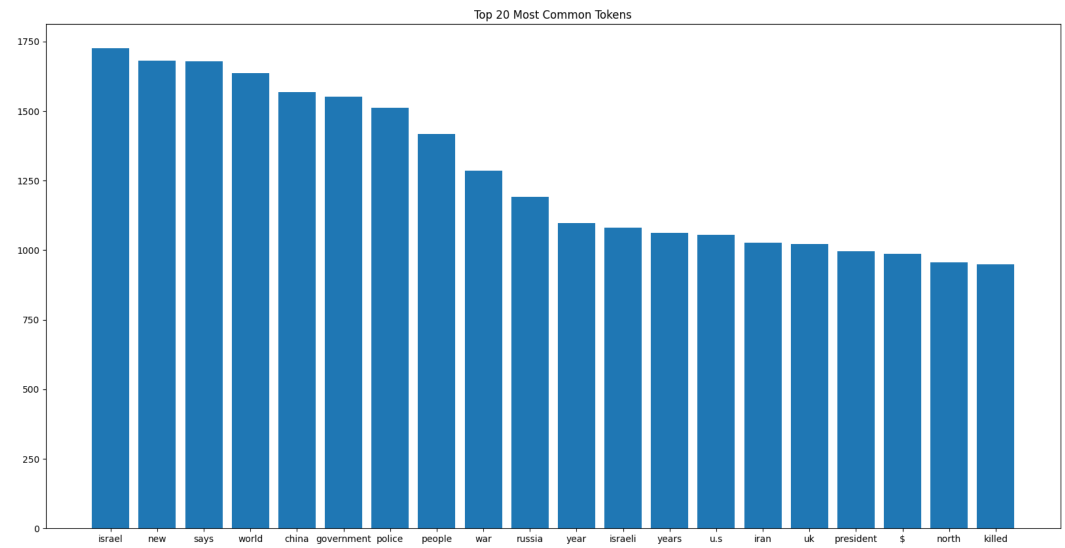


Figure 2: Top 20 Most Frequent Tokens

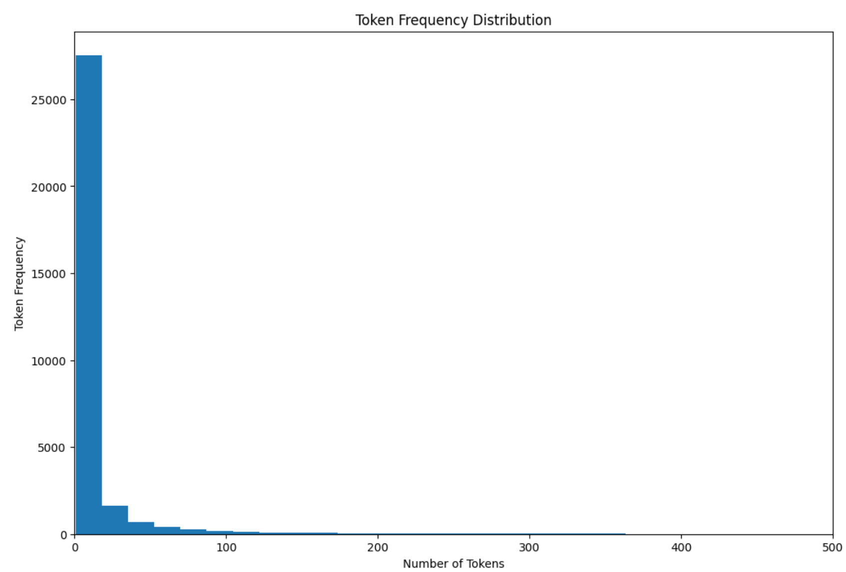


Figure 3: Token Frequency Distribution

I began by dropping any days that did not contain exactly 25 Reddit posts. I had to clean the raw text appeared to scraped as binary string with extraneous quotes surrounding the text (ex: b"”<text>’"). I then used spaCy to clean the text by removing punctuation, whitespace, and stop words. I want to note that stop words are sometimes included while doing sentiment analysis, but I chose to filter them out as the noise they would cause would outweigh the potential context gained.

I split the dataset into 80% training and 20% testing. I ensured that the training dataset was evenly split for the binary labels, as I learned in the most recent lab that a model may would skew towards the favored label. From this training set, I built a vocabulary by tokenizing all posts and assigning each token a unique token index.

3.4.2 Tensor Creation

To prepare tensors for the model, I padded each post to match the length of the longest post. The dimensions are described in 3.3.1.

3.4.3 Model Flow

Since the task is binary classification rather than next token prediction, I opted to use binary cross entropy with a sigmoid activation instead of softmax cross entropy (widely used since vocabulary size = number of classes). The following is a high level overview of the model:

Embeddings: Token embeddings + positional encodings.

Attention: MHA is applied with Num_{heads} heads over $\text{Num}_{\text{layers}}$.

Normalization: Residual connections are added and the result is normalized.

Token Pooling: Pool over token dimension to get a "sentence" level representation.

Optional LayerNorm

Post Pooling: Pool over post dimension to get a singular representation for the day.

Optional LayerNorm

Classification: A final classifier layer that predicts the binary label.

More formally:

$$[MHA \rightarrow B, P, T, C] \rightarrow [Pool_T \rightarrow B, P, C] \rightarrow [Pool_P \rightarrow B, C] \rightarrow [Linear \rightarrow B, 1] \rightarrow Sigmoid(B, 1)$$

For the optimizer, I chose to use AdamW as it has been adopted for transformer based architectures such as GPT.

3.4.4 Training/Testing

During training, I chose not to shuffle the data, as doing so would disrupt the chronological order of the days. If I randomly got batches, the model would essentially be "cheating".

For each training epoch, accuracy, precision, recall, F1 score, and binary cross entropy loss will be calculated in order to gauge the model's progression.

For testing I will similarly calculate accuracy, precision, recall, F1 scores, and binary cross entropy loss in order to gauge the model's predictive power.

4 Results

As stated earlier, I explored different pooling mechanisms to reduce dimensions for classification

4.0.1 Weighted Sum Pooling

Described in [4], I found that weighted sum pooling to be the most appropriate mechanism for the token dimension. This method allows the model to quantify the importance of certain tokens dynamically, with scalars being calculated from a token's rich representation thanks to the MHA layer. In the proposal, I said I wouldn't apply a casual mask. However, I think that if token t was able to see future tokens $[t, ft_1, ft_2..ft_n]$, this layer would artificially scale the importance of t. Padding tokens are zeroed out before the softmax step.

$$WSP(X = B, P, T, C) = \sum_{i}^{T} w_i token_i \rightarrow Softmax(A, dim = 2) \rightarrow Sum(X{*}A) \rightarrow (B, P, C)$$

4.0.2 Mean Pooling

I opted to try mean pooling for the post dimension, as it may be beneficial to take weigh each post equally. I also implemented weighted sum pooling on the post dimension, the different results will visualized.

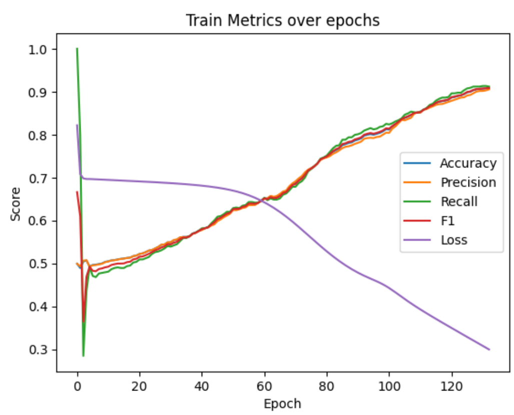
$$MP(X = B, P, C) = Mean(X, dim = 1) \rightarrow (B, C)$$

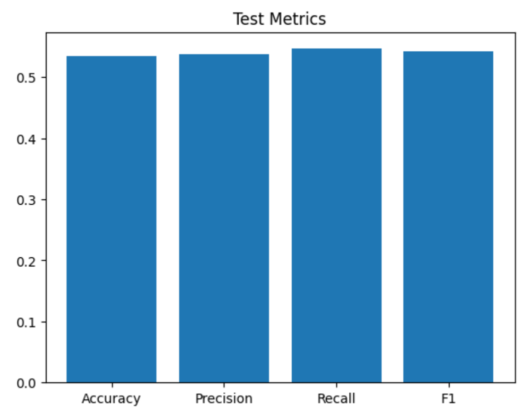
4.1 Analysis

During the training step for each $epoch \in Num_{epochs}$, I calculate the accuracy, precision, recall, F1, and loss. I thought line plots would be the most effective to see how the model progresses over these epochs. To visualize the test metrics, I will use a bar chart to compare.

4.1.1 Weighted Sum Pooling on T & P

About 130 epochs

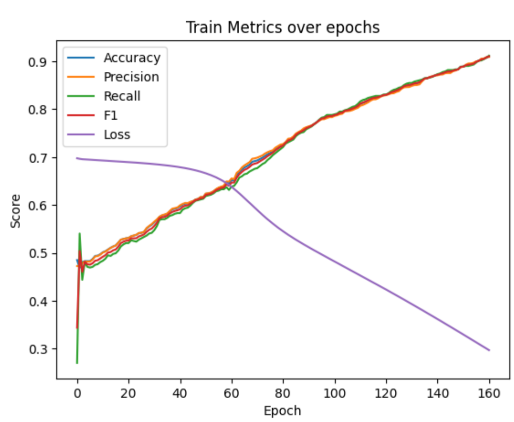




Accuracy:	53.3%
Precision:	53.8%
Recall:	54.6%
F1:	54.2%

4.1.2 Weighted Sum Pooling on T, Mean on P

About 160 epochs



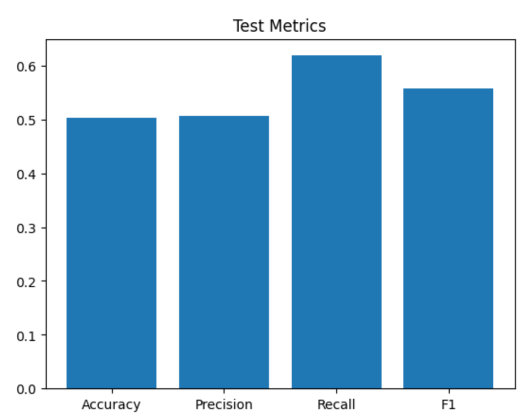


Figure 4: Testing Metrics

Accuracy:	50.2%
Precision:	50.6%
Recall:	61.8%
F1:	55.6%

Let $WSP_{TP}, WSP_T Mean_P$ denote the different pooling mechanisms; weighted sum pooling over both T and P, and weighted sum pooling over T with mean pooling over P respectively. Analyzing the training metrics, we see that WSP_{TP} has more fluctuations throughout epochs $20 \rightarrow 40, 80 \rightarrow 130$. $WSP_T Mean_P$ has fluctuations from epochs $0 \rightarrow 80$, and seems to stabilize throughout the remaining epochs.

Intuitively, $WSP_T Mean_P$ creates a more stable representation of the posts which led to a smoother learning pattern. We can see a steeper drop in loss around epoch $60 - 80$, at the same time the metrics stabilize. The model may have found the important tokens that are correlated with DIJA's price movement from the WSP on T layer, with mean pooling on P ensuring that certain posts don't skew the overall representation of a particular day. Essentially, the model may have found a generalization of the features that signify sentiment, ignoring noise from outlying posts.

For WSP_{TP} we see more fluctuations throughout the entire learning process, as both tokens and posts are being dynamically scaled based off their learned importance, creating more volatility in the model's learning.

Overall, $WSP_T Mean_P$ seems to catch when the market is going up more often, as it boasts a 62% recall score. For actual trading, both models could be used depending on a traders risk management. WSP_{TP} would be better for conservative traders who want to slightly lose less (precision) while gaining a little more (recall) consistently, due to it's balanced metrics. $WSP_T Mean_P$ would be better for traders focused on not missing up days (recall), with the inconvenience of more false positives (precision).

4.1.3 Comparison

During research of this data set, I found many random forest classification models that yielded metric scores of $\leq 50\%$ or lower when the chronological order of the dates was respected. If we consider this as a baseline, my implementation offers a slight edge for a trader.

4.1.4 Expectations/Notes

While I've never personally trained a model specifically to predict market movements. I've implemented a very simple GPT model, which seemed to work well for next token prediction. Based on that experience, I expected the metrics to be significantly higher. However, the noise and random nature of financial markets quickly became apparent. Without a clear benchmark for what metrics are respectable, I ended up experimenting with a variety of approaches to see what might work.

The dataset is also binary, containing only up or down labels without the a price feature. The smallest dip or gain would be labeled respectively as 0 or 1. Given this, if there was a third class for neutral/insignificant changes might have significantly improved the model's performance.

References

[1] Aaron7sun. Daily News for Stock Market Prediction. https://www.kaggle.com/datasets/aaron7sun/stocknews/data, 2016.

[2] Fanfei Meng and Chen-Ao Wang. Sentiment analysis with adaptive multi-head attention in transformer, 2024.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[4] Jinming Xing, Dongwen Luo, Chang Xue, and Ruilin Xing. Comparative analysis of pooling mechanisms in llms: A sentiment analysis perspective, 2025.