Event-driven Stock Market Forecasting Using a Transformer-Based Model

Jignasu Pathak
Department of Computer Science
jignasu@vt.edu

Amartya Dutta
Department of Computer Science
amartya@vt.edu

Nabayan Chaudhury
Department of Electrical and Computer Engineering
nabayanc@vt.edu

Abstract

Predicting stock prices is a well-studied time-series forecasting problem, and has been attempted to be solved by many approaches with varying levels of success. Since stock prices have local randomness, the forecasting problem becomes quite challenging. Recent events have also shown that public sentiment and dialogue can affect how stock prices vary. As such, we tackle the problem of stock price forecasting driven by public events by evaluating datasets on a modern transformer architecture for Long Time Series Forecasting and comparing our results with a standard auto-regressive prediction model. In our work, we have implemented a Transformer on Stock Price Data, using both single features as well as multiple features. Literature on the implementation of transformers on Stock Price Data is relatively sparse and we are one of the few to do it. Further, we create our own Event-Driven Stock Price Dataset for evaluation. Codes of our analysis can be found at the following link: https://github.com/Mister-JP/AML_2022_Informer-2020-model

1 Introduction

Stock price data is very volatile and usually depends a lot on public sentiment, which means usually the stock prices are decided based on the what traders think they should be valued at: if the traders believe the stock price is higher, stock will go down. And with this inspiration we have taken data from a Twitter API[2]. Prediction is easier to for shorter time series with accuracy being higher, but long term forecasting still poses a lot of challenges, with performance dropping rapidly for longer sequence lengths as has been observed with more traditional Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTMs). Recent work has shown how LSTMs fail when the predicted sequence length is increased four-fold. The transformer network has shown excellent modeling capabilities for long-range interactions and dependencies in sequential data and is therefore very attractive for time series modeling. We demonstrate the performance of a modified, low memory and computational cost transformer on real world stock market datasets.

2 Related Work

Related work in using transformer models for stock market predictions using deep transformer models have shown promising results. [7] presents a comparative study of classic prediction models including RNN, Convolutional Neural Networks (CNN), LSTM and Transformer models operating on a wide variety of stock market datasets inclusing CSI 300 and S&P 500, which shows that ttransformer

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

models clearly outperform the competition in predictive ability, but they do not handle an event-driven dataset with event embeddings that might impact prices in non-traditional ways. [9] discusses Natural Language Processing (NLP) methods for forecasting stock prices only from discussions on Twitter and other media, and discusses a wide variety of models for prediction, and it is notable that there is not a lot of work in event-driven stock price prediction using Transformer networks. [10] compares the performance of transformers usng different evaluation metrics for time-series prediction.

3 Problem Definition

Our problem of stock price forecasting is a multivariate long sequence time-series forecasting (LSTF) problem, with long input and output sequence lengths. As such, most approaches to solving this problem involve a standard encoder-decoder architecture, where the encoder encodes the input sequence to a hidden representation vector, and the decoder takes the input sequence and the hidden state representations to output its predictions.

4 The Informer Model

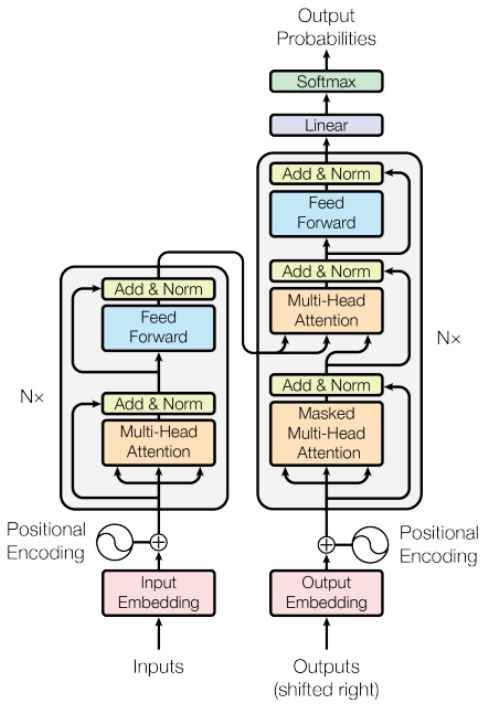

Figure 1: Vanilla transformer model architecture with encoder (left) and decoder (right)

4.1 The Vanilla Transformer

The vanilla transformer [1] introduced in 2017 has shown very good results in capturing long-term association dependencies among time series data. It comprises a stacked encoder-decoder architecture with residual connections. It employs a self-attention mechanism, where the input sequence is converted into a sequence of queries, keys, and values, and attention is calculated as a distribution over the weighted value set.

For every data-point $X \in \mathbb{R}^d$ we calculate Queries (Q), Keys (K), and Values (V) given by

$$Q = XW^Q, Q \in \mathbb{R}^{d_{model}} K = XW^K, K \in \mathbb{R}^{d_{model}} V = XW^V, V \in \mathbb{R}^{d_{model}}$$

where d_{model} is the model dimension of the transformer.

The scaled dot-product attention score is calculated by

$$Attention(Q, K, V) = softmax(QK^T/\sqrt{d_{model}})V$$

Attention can be calculated over multiple heads, allowing parallelization of operations and learning attention over multiple representation subspaces, increasing the association information learned by the model.

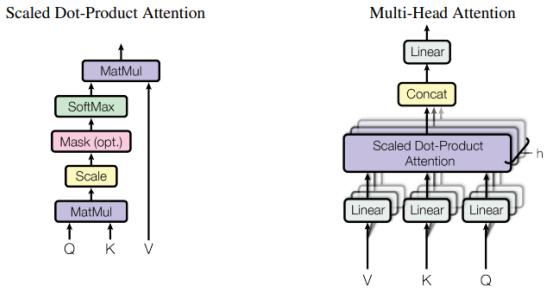

Figure 2: Scaled dot product attention (left) and multi-headed attention(right)

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_H)W^O$$

where

$$head_i = Attention(QW^Q{}_i, KW^K{}_i, VW^V{}_i)$$

where the projections are matrices of parameters, $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$, h being the number of attention heads.

This attention calculation operation provides computational benefits over standard time-series models like RNNs and LSTM networks which operate linearly. Along with the attention scores, the model also incorporates positional and temporal embeddings, which capture the temporal information at each time step of the data.

4.2 Problems with the Vanilla Transformer and improvements

The vanilla transformer runs into the following problems for LSTF problems:

- **Self-attention computation**: Self-attention computation is a quadratic process, so the complexity of calculation is $O(L^2)$ per layer.
- **Memory complexity is too high**: For N stacked attention layers, the memory usage has the complexity of $O(N.L^2)$, which prevents scalability.
- **Decoding speed limitations**: Since transformer outputs are decoded dynamically, the speed of long-range predictions falls dramatically.

The Informer model [5] addresses these limitations by observing that the generated self-attention maps tend to have sparsity, due to patterns in the generated attention, i.e a few pairs of query-key dot products dominate the attention distribution while others generate trivial attention. Thus, if it is possible to reduce the complexity of the attention map, it would theoretically allow us to calculate attention faster.

The efficient attention mechanism starts with the observation that the attention for the i^{th} query can be written using a kernel smoother to represent a distribution as:

$$Attention(q_i, K, V) = \sum_j \frac{(k(q_i, k_j)v_j}{\sum_l k(q_i, k_l))} = E_{p(k_j|q_i)}[v_j]$$

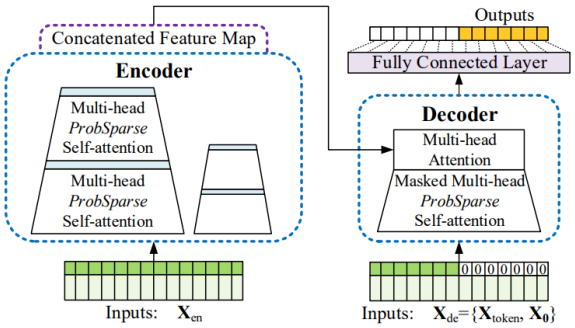

Figure 3: The model of the informer architecture, with the encoder(left) and decoder(right)

where $k(q_i, k_j)$ selects the asymmetric exponential kernel $exp(q_i k_j^T/\sqrt{d_{model}})$. Here, we see that the attention of the i^{th} query on the entire set of keys is defined as a probability distribution $p(k_j, q_i)$, and the dominant attention causing pairs of queries and keys pull this distribution away from a uniform distribution $q(k_j, q_i)$, which is essentially equal to $1/L_K$), where L_K is the length of the sequence of keys. Thus, by calculating the KL-divergence of these two distributions, we can obtain a metric that is defined as the sparsity measure for the i^{th}-query.

For the distributions p and q,

$$KL(q||p) = ln \sum_{l=1}^{L_K} e^{q_i K_l^T/\sqrt{d}} - \frac{1}{L_K} \sum_{j=1}^{L_K} q_i k_j^T/\sqrt{d} - lnL_K$$

On removing the constant term, we obtain a metric that is defined as the *sparsity measure*,

$$M(q_i, K) = ln \sum_{j=1}^{L_K} e^{\frac{q_i K_j^T}{\sqrt{d}}} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{q_i k_j^T}{\sqrt{d}}$$

which ensures that if the i^{th} query has higher attention value, the first term will be higher and is therefore more *diverse* and has a greater impact on the data. We set a sampling factor $u = c.lnL_Q$, and only sample the top u queries according to the sparsity measure, which makes attention calculation $O(lnL_Q)$, and memory usage for each layer becomes $O(L_K lnL_Q)$.

While attention calculation is $O(LlnL$, computing the sparsity measure can also be performed with a lower complexity than $O(L^2)$, from the max-mean measurement, which can be shown to be:

$$\overline{M}(q_i, K) = \max_j \{\frac{q_i k_j^T}{\sqrt{d}}\} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{q_i k_j^T}{\sqrt{d}}$$

and by randomly sampling $U = L_K lnL_Q$ dot-products we can obtain the above approximation. This can be further sampled to select a sparse query vector $\overline{Q}$, which we then use in attention calculation as:

$$Sparse\ Attention(Q, K, V) = Softmax(\overline{Q}K^T/\sqrt{d_{model}})$$

4.3 Encoder

The input sequence is first reshaped into a matrix given by $X_e^t n \in \mathbb{R}^{L_x \times d_{model}}$, which is then fit into a distilling operation that extracts the Values(V) which have a dominating impact on the attention. This process uses dilated convolution, inspired from the theory behind Dilated Residual Networks, and can be written as:

$$X_{j+1}^t = MaxPool(ExponentialLinearUnit(Conv1d([X_j^t]_{AttentionBlock})$$

where j and $j+1$ refer to the j^{th} and $j+1^th$ layers respectively. Addition of the maxpooling layer of stride 2 allows us to downsample the input to a half-slice, which reduces the memory usage to $O((2-\epsilon)LlogL$ where ϵ is some small number. The output of the final attention block is then concatenated to form a feature map, which is then used as the hidden representation vector fed to the decoder.

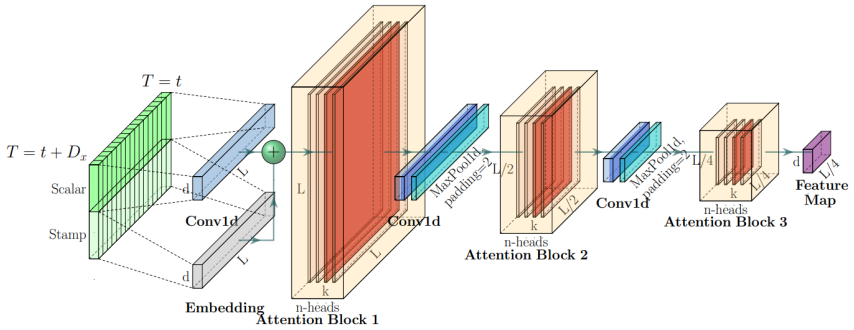

Figure 4: A single stack of the encoder

4.4 Decoder

The decoder is a standard transformer decoder that uses generative inference, where the decoder input is an embedded token given by:

$$X_{de}^t = Concat(X_{token}^t, X_0^t) \in \mathbb{R}^{(L_y+L_{token})\times d_{model}}$$

where X_{token}^t is a start token taken from the input data, and X_0^t is the target token, initialized to be 0. This token and the encoder input is passed through a masked-multiheaded attention block, and the attention output is fed to a fully-connected layer that predicts the output sequence. The masked attention scores are set to $-\infty$, so that information is not leaked to the predictor, and thus prevents auto-regression.

The chosen loss function is the *Mean Squared Error*, which is calculated at the decoder output with respect to the predicted target sequence, and propagated through the entire model.

5 Experiments

5.1 Stock Price Data

In finance, market data is price and other related data for a financial instrument reported by a trading venue such as a stock exchange. The stock's price only tells you a company's current value or its market value. So, the price represents how much the stock trades at—or the price agreed upon by a buyer and a seller. If there are more buyers than sellers, the stock price will climb. If there are more sellers than buyers, the price will drop. The ability to predict the behavior of such data is important from both a Financial as well as Machine Learning point of view. We use the Informer [5] model for all our Stock Price Prediction tasks.

5.1.1 S&P 500

Standard & Poor's 500 Index, created in 1957 is widely regarded as the best single gauge of large-cap US equities. It is a market-capitalization-weighted index of the 500 largest publicly traded companies in the U.S. and covers approximately 80% of available market capitalization.
We consider the S&P 500 Index Data because it is a widely popular dataset for stock price prediction. We first want to compare the Informer model's performance on a dataset that varies simply based on Stock Price feature values.

Data Preparation The S&P Data is constantly being updated but since we wanted a fair comparison, we extracted the same values as [7], which ranges from 1st January 2010 to 31st December 2020. We extracted the values from Marketwatch. The data extracted has Date, Open, High, Low, Close, Adj Close, and Volume values ranging from 1st January 2010 to 31st December 2020. Each of the values is normalized using the formula below:

$$x_t = (x_t - \mu)/\sigma \tag{1}$$

The dataset is split into two individual datasets, one with only Dates and Close price values and the other with all the values.
Fig 5 shows the trend of the Closing Price value over the years.

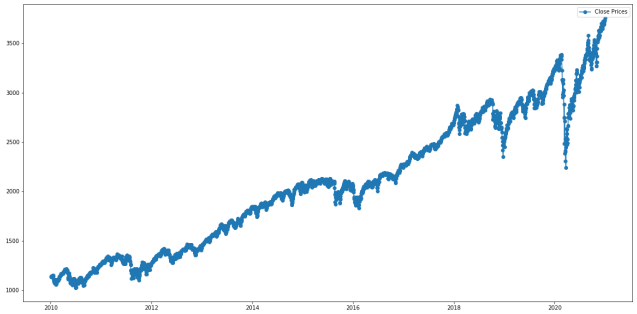

Figure 5: Closing Price of S&P 500 over the years

Single Feature Most works predict Stock Price values using a Window method that learns from its previous values to predict future values of the Closing price. In [7], they used the Deep Transformer for predicting the Stock Price values and achieved a State of the Art score in MSE and MAE value. A comparison of their training parameters with ours is given in Table 1

Hyper-parameters	Batch size	Learning rate	Epochs	Dropout
Deep Transformer	16	0.0001	1000	0.1
Informer	32	0.0001	20	0.05

Table 1: Hyper-parameter values comparison with Deep Transformer

The decoder input sequence length was kept constant at 8 throughout this experiment. Furthermore, we also vary the Input sequence length into the encoder and the predicted sequence length and record how they affect the model's performance. These experimental results are shown in Table 2

Encoder Input Length	Prediction Sequence	MSE	MAE
25	10	0.194	0.39
25	15	0.179	0.37
30	10	0.178	0.367
30	15	0.121	0.41
50	15	0.209	0.392
32	2	0.064	0.2

Table 2: Encoder Input Length and Predicted Sequence and how it affects performance

Even though, we ran it 50 times the less number of epochs as compared to [7], when we used the same number of Encode and Prediction Sequence (32,2) they did, we achieved a good MSE score. We compare the performance of our implementation with various other approaches in the Results section.

Multiple Features Most works that have been performed on Stock Market Prediction use only a single feature, this is especially true for Transformer based methods, which are relatively new to this. Therefore, as additional work, we also predict the Close prices of S&P 500 using multiple

feature values and evaluate its performance. We check the correlation between each feature and the Close price in order to decide which of them will be a good fit for the Multi-feature prediction. This can be seen in Fig 6. Since our target value is Close Price and the Volume of stocks has a negative correlation, we drop it off for our prediction task. Thus, the features used as Input are Open, High, Low, and Adj Close.

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999839	0.999738	0.999600	0.999600	-0.026733
High	0.999839	1.000000	0.999632	0.999768	0.999768	-0.020745
Low	0.999738	0.999632	1.000000	0.999800	0.999800	-0.037105
Close	0.999600	0.999768	0.999800	1.000000	1.000000	-0.029631
Adj Close	0.999600	0.999768	0.999800	1.000000	1.000000	-0.029631
Volume	-0.026733	-0.020745	-0.037105	-0.029631	-0.029631	1.000000

Figure 6: Correlation between different feature values in S&P 500 Dataset

We perform several experiments with different parameter values. These results are discussed in Table 4 in the Results section. The decoder input sequence length was kept constant at 16 throughout this experiment.

5.2 Event Driven Dataset

After having shown our experimental results on Single and Multi-feature Stock Prediction, we extend it further to generate our Event driven Dataset.

5.2.1 Amazon Stock Price Data + Tweets

Dataset Preparation We used the Twitter API for scrapping the Tweet data and performed sentiment analysis on the tweets obtained. We have used the Tweeteval[6] pre-trained model on the Twitter data. Tweeteval [6] was introduced to train a Language Model, especially for Twitter sentiment analysis, and get accurate results as previous benchmarks were good but social media has fast-paced and unique characteristics which makes it hard to get accurate results on tweets. The most difficult part about collecting tweets was the keyword as "Amazon" led to advertisements by sellers and not the sentiment about the Amazon stock price. Multiple efforts later, we found the keyword "AMZN report" to obtain people's thoughts on Amazon stock performance. We also tried different approaches including adding the official profit and loss of the company in the data as well as the past reports for each row. And we have compared the results in the result section.
The stock gained from the Yahoo website. We parsed the data and combined it with date-wise sentiment analysis data. We also embedded the weekly, monthly, and yearly data in the dataset by segregating the data on 7 days, 30 days, and 365 days in each column. We normalized each column and obtained a dataset with information about general public sentiment a day before, a week before, a month before, and a year before in order to predict whether the stock price would increase or decrease and by what percentage. Time series usually tries to find the trends from past data and tries to predict, but in order for it to predict the attribute needs to be leveled and the predictions will be made accordingly. In our case, we have taken multiple approaches to compare the results based on the different attributes, like Close Price which increased from 30$ to 140$ in the past 5 years as well as the difference between the prices. And the final attribute which gave the best result was the percentage change in the closing price, which was 1-2% which was constant over the period of 5 years.

Method The model Informer[5] has a parameter that allows one to predict based on multiple as well as single inputs to predict another single output. This means the model gets trained on multiple features and uses the information to predict a single-valued output. In our case, we want to use multivariate data which includes both the stock prices as well as the Twitter sentimental analysis data

to train our model and predict only the stock's closing price based on the input provided to the model. To notice the difference in the amount of data fed into the model we have also tried a range of attributes from 102 to 27 for seeing the difference in the performance based on the complexity of the model. In the Informer model, we tuned the hyperparameters: sequence length, label length, and prediction length. These are the first features we altered from the previous paper to various values varying between 10-200. The possible set of hyperparameter values was taken from [1]. In [5], the authors have compared the results for varying prediction lengths and inputs. In our case, since stock price data is volatile and less predictable we have not tried to estimate the price for more than 50 days ahead, as the accuracy was decreasing drastically.
Going forward, we record the amount of time it takes to train the model, with a limited amount of data. The majority of time was taken for the model with higher dimensions and we have used the google cloud platform for faster training of the model, with the size of the model varying from 512 to 4096 we have found a huge difference in the time it takes to train the model. Since the change in accuracy was limited to a certain extent it made us think of the method of our model, which we selected based on different factors like time taken overall over the accuracy.
After the experiments conducted, we realized that Transformers need a great deal of parameter tuning and Data processing to achieve the best results. This aligns with [1], where the authors stated that a very simple linear model can outperform all of the previous models on a variety of common benchmarks and it challenges the usefulness of a Transformer for time series forecasting. To further build upon our work, we compared it to results generated by the ARIMA model which are discussed in the Results section.

6 Results

6.1 S&P500

The results of single feature detection in the S&P 500 dataset can be seen below in Table 3. Our model doesn't perform the best but with this we were able to validate its performance on the Stock Price dataset. We saw from the results that having a lower prediction and relatively larger input sequence generates better results because the model learns more and predicts less.

Model	MSE	MAE
RNN	0.142	0.284
LSTM	0.1208	0.2676
WLSTM	0.1067	0.247
GRU	0.1	0.239
WLSTM + Attention	0.054	0.19
Deep Transformer	0.014	0.081
Informer	0.064	0.2

Table 3: MSE and MAE value comparison for single feature detection

As for Multi-feature prediction, we can see from Table 4 how the MSE and MAE values vary with the different parameter values. There is a huge room for improvement, but from these, we get an idea of how varying the parameters affect the model performance. For a lower input and prediction

Encoder Input Length	Prediction Sequence	MSE	MAE
64	24	0.194	0.39
72	48	0.31	0.5
100	50	0.29	0.49

Table 4: Encoder Input Length and Predicted Sequence and how it affects performance for Multiple Feature to Target Prediction

sequence lengths, we achieved better results while when the difference between them decreased the results got worse as well.

6.2 Amazon Dataset

The results from the best hyper-tuned parameters are given below. We have performed experiments on various different values because the nature of the paper is experimental. We have tried several approaches with trial and error. And we have used the closing price to generate new attributes and experiment with them for checking the model performance.

In Table 5 we have presented Mean Square error(MSE) and Mean Absolute Error(MAE) as a way to find differences in prediction and the ground truth of the original values.

Predicting	Number of attributes	MSE	MAE	Epoch	Time sec	Loss
Close price nrml	102	0.0454	0.18842	1	40	0.5574
				2	40	0.5255
				3	72	0.487135
				4	159	0.4563
Close price nrml	27	0.04688	0.184	1	40	0.2677
				2	41	0.5609
				3	57	0.46
				4	156	0.4945
Close price diff	102	18.59	3.24	1	213	2.04
				2	210	2.04
				3	211	2.04
				4		
Close price diff	27	18.49	3.255	1	72	2.0644
				2	155	2.0219
				3	79	2.0437
				4	71	2.03576
Close price diff nrml	**102**	**0.046**	**0.161**	**1**	**66**	**0.106**
				2	**71**	**0.101**
				3	**72**	**0.103**
				4	**72**	**0.1**
Close price diff nrml	**27**	**0.045**	**0.1604**	**1**	**150**	**0.108**
				2	**73**	**0.103**
		bf		**3**	**58**	**0.09965**
				4	**40**	**0.1**
Close price diff perc	102	0.0013	0.0275	1	42	0.0323
				2	43.2	0.034
				3	42	0.015
				4	72	0.013
Close price diff perc	27	0.0014	0.028	1	423	0.038
				2	42.08	0.023
				3	41	0.059
				4	41	0.015
Close price diff perc nrml	102	0.046	0.161	1	74.12	0.091
				2	42.23	0.07959
				3	40	0.075
				4	40	0.075
Close price diff perc nrml	27	0.046	0.161	1	41	0.082
				2	40	0.07the 7
				3	40	0.075
				4	40	0.076
Auto-Arima		0.39	0.54			

Table 5: Performance of the Informer on different target values with different attribute values

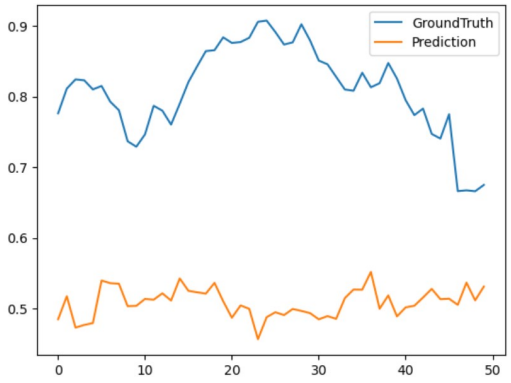

Figure 7: Prediction of actual stock price from the Dataset with tweets sentiment and income tax report making 102 attributes, and predicting Directly the close price.

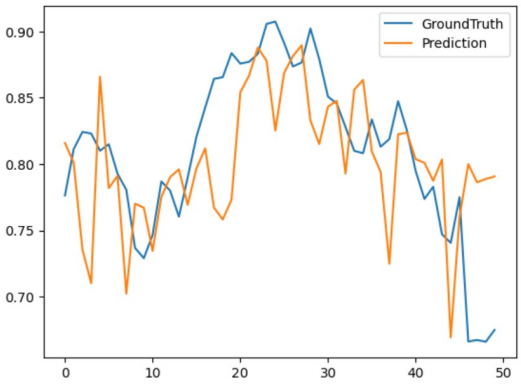

Figure 8: Prediction of actual stock price from the Dataset without official data on tax reports making 22 sentiment attributes, and predicting Directly the close price.

Figure 9: Prediction of Close price by transformer

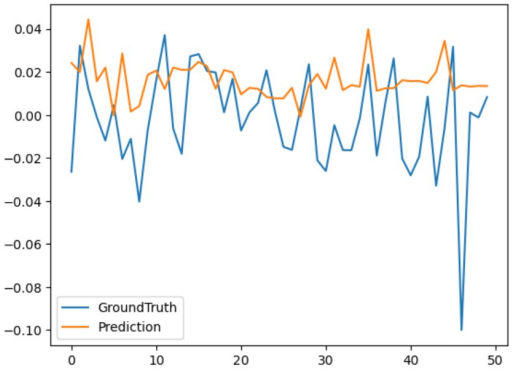

Figure 10: Prediction of normalised Close price difference from only 102 attributes of tax reports and sentiment analysis

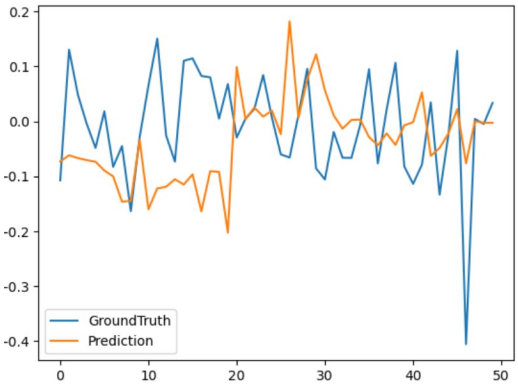

Figure 11: Prediction of Close price difference from only 22 attributes of sentiment analysis

Figure 12: Prediction of Close price difference

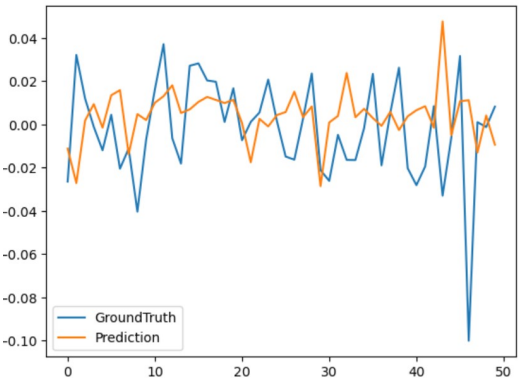

Figure 13: Prediction of normalized Close price difference from only 102 attributes of tax reports and sentiment analysis

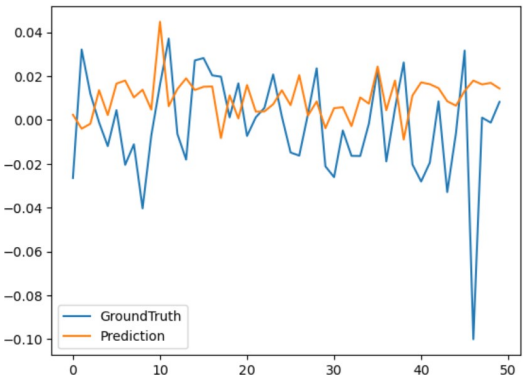

Figure 14: Prediction of normalized Close price difference from only 22 attributes of sentiment analysis

Figure 15: Prediction of normalised closed price difference on two different datasets.

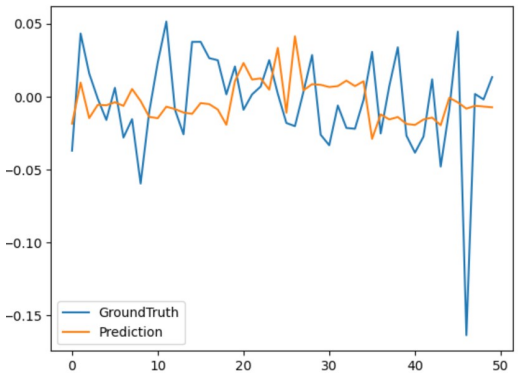

Figure 16: Prediction of Close price percentage of difference from only 102 attributes of tax reports and sentiment analysis

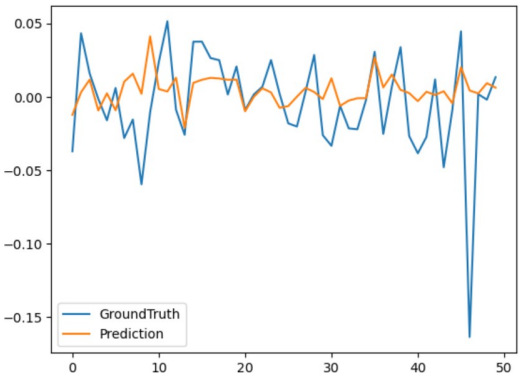

Figure 17: Prediction of Close price percentage of difference from only 22 attributes of sentiment analysis

Figure 18: Prediction of closed price percentage of difference on two different datasets.

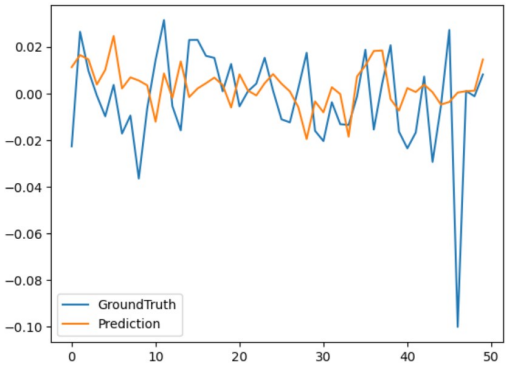

Figure 19: Prediction of normalised value of Close price percentage of difference from only 102 attributes of tax reports and sentiment analysis

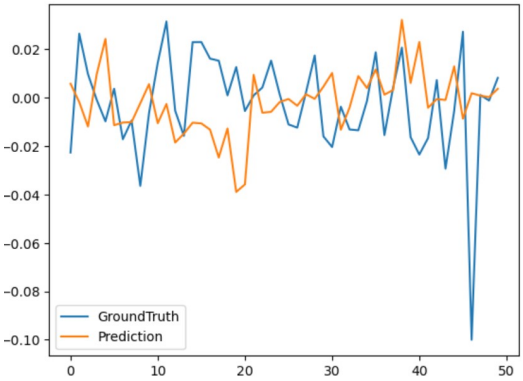

Figure 20: Prediction of normalised value of Close price percentage of difference from only 22 attributes of sentiment analysis

Figure 21: Prediction of normalised closed price percentage of difference on two different datasets.

As we can see in the results presented above the best performance is given by the normalised closer price difference compared to any other values being predicted and it is very close to actual graph.

The major issue we faced in getting better performance was the trade-off between how to normalize the values while avoiding vanishing gradient. In some cases, if we were increasing the number of epochs, the overall prediction was coming out to be zero, and from our understanding, it is happening because the average change in percentage increase and decrease is close to zero but positive.

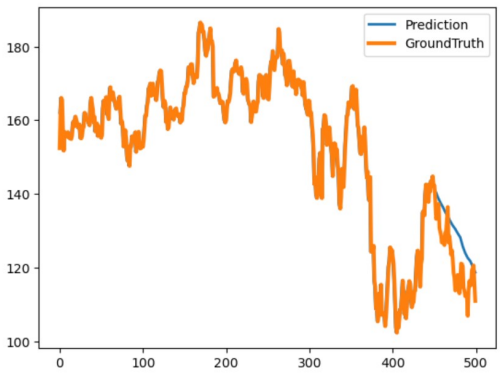

Figure 22: Prediction of final stock price from the transformer model

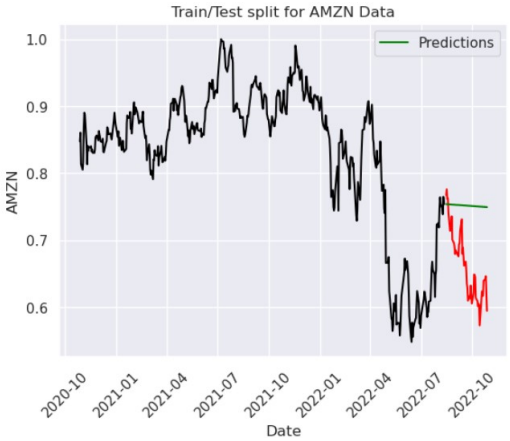

Figure 23: Prediction of final stock price from the ARIMA model

Figure 24: Comparison between prediction by ARIMA model vs our model for predicting the final stock price.

The above results shows how our model is predicting closer to the ground truth, and it was the result after a lot of fine tuning in the model and data we feed to the model. We have tried and tested 150+ combinations of hyper-parameters over the set of 10 datasets with different values being predicted and this is the results we came up with.

7 Conclusion and Future Works

From our experiments, it is clear that the Informer approach works reasonably well at predicting stock prices, when compared to standard auto-regressive approaches.

Future work includes:

- Comparison of the informer model with other deep-learning based prediction models over a wide variety of datasets.
- Creating a complete pipeline that combines event embedding generation from public domain APIs and stock price prediction on a more comprehensive dataset like StockNet.
- Improve the Informer model and create a novel transformer model, by possibly considering an autoregressive approach, and studying the impact of input masking on the results. Also possible to combine graph neural representations with the transformer network to obtain more detailed input subspace representations which might improve performance. It is also interesting to consider the possibility of a consolidated transformer framework that can both generate event embeddings as well as perform predictions.
- Recent work [8] has shown that transformer networks can be adapted as reinforcement learning agents, so it will be interesting to see possible performance benefits if a stock-price prediction model is adapted into something similar.

8 Contributions

Jignasu Pathak : AMAZN Data Experiment, Literature Review, Report Writing

Amartya Dutta : S&P Data Experiments, Literature Review, Report Writing

Nabayan Chaudhury : Data extraction, Literature Review, Report Writing, Verification of theoretical results

References

[1] Nie, Y., Nguyen, N., Sinthong, P., and Kalagnanam, J. 2022. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. arXiv preprint arXiv:2211.14730.

[2] Wu, H., Xu, J., Wang, J., and Long, M.. (2021). Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting.

[3] Subhash Chand Agrawal 2021 IOP Conf. Ser.: Mater. Sci. Eng. 1116 012189. Deep learning based non-linear regression for Stock Prediction

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., & Polosukhin, I.. (2017). Attention Is All You Need.
[5] Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence (pp. 11106–11115).
[6] Barbieri, F., Camacho-Collados, J., Neves, L., & Espinosa-Anke, L. (2020). Tweeteval: Unified benchmark and comparative evaluation for tweet classification. arXiv preprint arXiv:2010.12421.
[7] Wang, C., Chen, Y., Zhang, S., and Zhang, Q. 2022. Stock market index prediction using deep Transformer model. Expert Systems with Applications, 208, p.118128.
[8] Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I.. (2021). Decision Transformer: Reinforcement Learning via Sequence Modeling.
[9] Weiwei Jiang 2021. Applications of deep learning in stock market prediction: Recent progress. Expert Systems with Applications, 184, p.115537.
[10] Farsani, R & Pazouki, Ehsan & Jecei, Jecei. (2021). A Transformer Self-Attention Model for Time Series Forecasting. Journal of Electrical and Computer Engineering Innovations. 9. 1-10. 10.22061/JECEI.2020.7426.391.