Leveraging Market Sentiment to Optimize for Stock Volatility Prediction

Aakash Gupta

March 16, 2025

Abstract

In this paper, I propose three models in which I leverage Market Sentiment from the years 2015 to 2020 to predict daily stock volatility. The objective is to modify existing stock volatility models to incorporate market sentiment and evaluate their predictive accuracy compared to the baseline.

1 Replication

The data and the processes can be found on ”https://github.com/aakashg2/MarketSentiment”

2 Introduction and Motivation

There are numerous financial models that forecast volatility, stock price, and returns. Many of these models use predictors such as past returns, volatility, and prices to predict future values, but there is no general model that incorporates market sentiment as a predictor of future stock price volatility. Although market sentiment can be extremely abstract (open to many different interpretations), it can be used to predict behavior shocks. Using market sentiment, one can quantify public sentiment about the economy and use it to predict future shocks; something that conventional financial models cannot account for.

Market confidence and sentiment play an important role in determining stock volatility. News headlines about potential recessions and even talks of market crashes can trigger shocks in the stock even though they may not actually occur. However, in most existing models, market sentiment is usually never considered when predicting stock price volatility.

In this paper, I propose three models that are modified to take into account different interpretations of market sentiment and evaluate them to see how well they perform relative to the baseline models in that space. The main goal of this paper is to promote the importance of market sentiment into existing financial models to bolster their predictive accuracy.

Section 2 gives a brief description of the data and the data cleaning process. Section 3 uses a Lasso model to forecast stock volatility. Section 4 uses a modified GARCH model to forecast volatility. Section 5 uses an LSTM neural network to predict stock volatility.

3 Description of the Data

3.1 Source of the Data

In this paper, I use two datasets: A dataset that contains all of the articles from the New York Times website from the year 2015 to 2020. As the New York Times is a popular News Site with 11.4 million users and is said to have a general audience, it will be safe to assume that these articles will be seen

by the general public.
In total, there are approximately 70,000 articles in total with 30 articles per day from January 1, 2015, to December 31, 2020.
The second dataset are the Stock prices from the S&P 500 from 2015 to 2020. The reason I chose this stock was because it is widely considered as the measure for the

Below are the first 6 rows of the article data frame and stock data frame respectively.

3.2 Descriptive Statistics

As my entire dataset was a collection of all the articles from 2015 to 2020, I opted to filter out for the "important" articles; namely the articles that I thought would influence public sentiment about the market. As such I filtered only for articles that contained certain keywords that reflected its relation to the market or potential conflicts that would affect market sentiment.

The list of words were

`lockdown, covid, virus, terrorist, dispute, fears, fear, crash, business, finance, war, automation, stocks, conflict, sanction, sanctions, leaders, crisis, tension, hikes, panic, recession, tumble, turmoil, uncertainty, volatility, inflation, layoff, decline, bankruptcy, covid-19, China, restrictions, tariffs, business, finance, world, politics, your-money`

I only included negative words as positive words or positive words are not known to decrease volatility. It is a theory that the market reacts greatly to bad news/sentiment, which is why I solely wanted to focus on articles that produced such sentiment.

After the filtering process, I was left with approximately 14,000 articles from 2015 to 2020. On average, there were around 10 articles a day with no missing articles on a single day.

4 Lasso Model

Before I began any complex models, I elected to choose a naive model and that would observe if certain words would cause a change in stock volatility over time.
However, I could not use a simple OLS model as having over 10,000 covariates regressed on just 1400 datapoints would not not due to the curse of dimensionality. Furthermore, I also knew that only small fraction of words would invoke a response in the response which is why I elected to use a simple Lasso Model. As I had over 10,000 covariates and the assumption of sparsity, this would be a candidate model. Additionally, the results of Lasso would also serve as a way to perform descriptive analysis and test the claim that news headlines influence stock volatility.

4.1 Construction of the Data

Consider the model:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + X_t \Gamma + \epsilon_t$$

y_t is the stock volatility at time t, which in this case is the absolute difference between the opening price and closing price at time t.

$$y_t = |\text{Stock Opening Price} - \text{Stock Closing Price}|$$

X is a sparse matrix that only contains binary values. It has 1422 rows, which are the total number of trading days from 2015 to 2020. It also has 10762 columns which are the total number of unique words in all of the market related articles from 2015 to 2020.

X_t will hence be a 1x10762 dimension sparse matrix and it will span the total number of unique words that the titles contain in the dataset of article titles. At time t, words that show up in titles will be marked as 1 and 0 otherwise.

$$X_{ti} = \mathbb{1}[w_{ti} \in W_t]$$

W_t are the sets of words that consist of the title of the article at time t.

4.2 Results of the Lasso Model

Before we get into the results of the Lasso model, I needed a baseline model that I could compare my MSE score to. I elected to regress it against a simple AR(2) model which is just a regression of volatility against two time lags.

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 t_{t-2} + \epsilon_t$$

Words	Coefficients
China	-0.0756
Coronavirus	0.248589
Debt	-0.079963
Iran	0.037635
Trump	0.014818
Border	0.590347
Debate	0.264783
Markets	0.420555
Recession	0.207803
Stock	0.402308

These were the top 10 words that contributed to the volatility increases. The main words that increased volatility were: markets, recession, stocks, Coronavirus, and border.
Looking at the list of article titles, we see that these results corroborate our hypothesis that certain words to influence market sentiment through volatility.

Model	Test MSE Score	Train MSE Score
Baseline AR(2) Model	5.04941	1.23759
Lasso Model	1.41011	0.691066

By splitting the model 80-20 by training and testing data, I produced a simple forecast generated by the Baseline and Lasso model to get a superior MSE of 1.41 compared to the MSE Score of 5.05 for the baseline model, showing that even a naive model can outperform the baseline by approximately 5 magnitudes.
One important observation is that the training error is drastically lower for than the testing error, which is common but showcases the biasedness of the model.
In general, the Naive Lasso model provides credence to the possibility that market sentiment can be used to capture shocks in the economy.

5 Modified GARCH Model

A GARCH model is a popular model that can be used to capture the variance of the stock in different time intervals. As a result, it would be a perfect candidate model that could be modified to handle market sentiment. However, one obstacle would be how it would incorporate market sentiment. Our previous method would not work as we would run into the curse of dimensionality and hence we could have to reduce the covariates to a minimal value.

The modified model will be

$$\sigma_t^2 = \omega + \alpha_1 \sigma_{t-1}^2 + \beta_1 R_{t-1}^2 + \beta_2 R_{t-2}^2 + \gamma X_t + \epsilon_t$$

$$\sigma_t^2 \sim N(0, Var(P_t))$$

where P_t is the variance in the price of the stock which is volatility.

I added an exogenous component that would includes the negativity sentiment for all the articles published on day t. The transformer I used was ProsusAI in the BERT ecosystem. This is a transformer that is specialized in reading financial data or financial articles. Hence, sentiment analysis will be best simulate investor behavior. For example, the sentence: "Apple Lays off 10% of workforce" would output a 95% negativity score. Meanwhile, the sentence, "Apple reaches record profits this quarter" would output an 88% positivity score.
The reason I decided to only include the negativity sentiment but not the positivity sentiment or the neutral sentiment was because in the financial world, volatility shocks are not caused by positive or neutral news; they are caused by bad news. Even if there was good news such as "Amazon enjoys record profits", there would not be a volatility shocks such that the volatility would rapidly decrease but instead there would be a long term decrease in volatility. However, if there was bad news such as "Covid-19 lockdown measures will likely last for a month", then there would be an immediate volatility shock on day 1. To complement this ideal, the value of X_t would be the 80th percentile of negativity sentiment to give the effect that investors will only look at the very bad news to determine their confidence in the economy.

5.1 Introduction to Sentiment Analysis

Sentiment Analysis is an NLP technique that can quantify the portion of emotional tone inside a text. There are many variations of Sentiment Analysis you can use but the most popular method is Tokenization.

Tokenization is the process of breaking down sentences into a vector of words so that they will be easier to analyze.

"schools reopen worldwide despite coronavirus" $\xrightarrow{Tokenizer}$ [schools, reopen, worldwide, despite, coronavirus]

Next, the model will remove unnecessary words such as prepositions and neutral words to save on computational time and only focus on the emotional words. The resultant output will be called a word embedding

After this preprocessing, the word embeddings are then placed into a transformer which do most of the heavy lifting. Transformers are functions/blackboxes that push our word embeddings through encoders and decoders to output a sentiment score. This is illustrated in **figure 2**.
An encoder is a function that takes a word embedding and outputs a numerical matrix representation of that input. Next, the decoder will take that output and will generate a sentiment output. In most transformers, the encoders and decoders are stacked on each other many times to ensure accuracy.

5.1.1 Encoders

The architecture of an Encoder is quite complex and can be summarized in 4 processes illustrated in **Figure 2**.

The first step: Input Embeddings is the process of turning word embeddings into vectors using embedding layers. These embedding layers will capture the semantic meaning of the words and convert them into numerical vectors. Traditionally, each word in the word embedding will be turned into a vector of size 512.

The second step: Positional Encoding is put into place to capture the position of words in sentences. As the order of words can change the meaning of sentences, it is important that order is preserved.

The third step: Attention Mechanism is a process where the input sequence is divided into 3 vectors: a Query, a Key, and a Value. These three vectors are always linear combinations of the input sequence. Using these values, an attention matrix is which is a weighted sum of values based on the similarity of the query and key vectors. This and the original input is then passed through a feed-forward Neural Network to produce the final output which produces an output sequence. Using this method, the model will now be able to capture contextual clues and long range dependencies.
Typically in most transformers, encoders are stacked on top of each other, usually around 6 times to get a more refined result.

5.1.2 Decoders

The first step: Positional Encoding takes the input and passed it by the positional encoding layer to produce embeddings. These are then channeled into the first multi-head attention layer of the decoder, where the attention scores specific to the decoder are computed.
The second step: The Masked Self Attention Mechanism is similar to the self-attention mechanism but with the difference that it prevents positions from attending to subsequent positions. This ensures that each word in the sequence isn't influenced by future tokens.
The third step: Cross Attention is a mechanism that allows the model to attend to the information from the encoders output. This integrates information from the encoder to the information in the decoder.
The fourth step: A Feed Neural Network is similar to the fifth step in the Encoder portion.
The last step: A linear classifier is turns the input into a probability score of Sentiment. Using a softmax layer, it transforms the input into a Sentiment Score.

5.2 Results of the modified GARCH model

As $\hat{\sigma_t^2}$ is an unobserved value, I instead replaced it with an estimate of volatility, which is

$$|P_t^{After} - P_t^{Before}|$$

Many papers use this metric as a way to estimate volatility, and hence will follow the agreed upon conjecture.

As seen in **Figure 1**, we see that our negativity sentiment is completely insignificant by the central t test. Furthermore, the inclusion of the parameter in the exogenous increases the AIC and BIC scores, which shows the the inclusion of the negative sentiment parameter does not lead to a better model. The same results are reached after tuning the quartile of negative sentiment article titles each day.

One potential reason for this result is that despite sentiment scores being the best way to quantify, they are wholly inaccurate on some bases, which means that they may flag a Negative article tile as a neutral tone. To fix these issues, one will have to train a sentiment analysis tool on a similar dataset, but due to the massive computational costs and the potential biasness (it is not recommended to test data you trained with), I opted not to move any further with the sentiment analysis method to move to greener pastures.

6 Neural Networks

The reason I chose neural networks was due to the success of the Lasso Model. As quantifying sentiment was proven to be problematic due to the reasons I discussed above, it would be best to keep the

original essence of words. With this constraint in mind, the best model to utilize would be a neural network.
Along with this, I also think that the non-linear nature of the volatility of stocks would fit in perfectly into the features of the neural network. As stock volatility and stock movements in general are non-linear, conventional models such as Lasso and GARCH models fail by constraining movements to be linear. Due to this, I think that Neural Networks would be a second option to forecast stock volatility. Along with this, Neural Networks are one of the only other models other than Lasso that can take multiple inputs without suffering from the curse of dimensionality. Their architecture is built in such a way that it can filter out noise and pick apart the meaningful variance that influence stock markets. This is especially useful as we know only a handful of words in our unique list of words would influence stock volatility.
I opted to choose an LSTM Neural Network was because this models are adept at handling data which have time lags as my dataset is a time series.
As LSTMS have a memory component that allows them to retain and store long term information from past data to inform new predictions, this would be the best model to forecast stock volatility.
To keep things simple, my input would be the X_t matrix along with the two time lags of volatility. My output would be a fitted value of volatility.

6.1 Blueprints of an LSTM Neural Network

An LSTM is a variant of the Recurrent Neural Network that is specialized with time series data or outputs that are dependent on past data. In order to handle this dependency, the architecture of LSTM Neural Networks are different than traditional Neural Networks.

As seen in **Figure 3**, Recurrent Neural Networks have an input layer (which is 10764), hidden layers, and an output layer (which is 1). However, RNNs can access past information and use it to predict the present. For example, if a Feed Forward Neural Network was trying to process a sequence of 10 stock volatility values, it would forget the first value after processing the second value and hence would only have the second value to use to predict the third value. This is especially flawed when working with time series data; when the output is dependent on past data. RNNs solve this by adding short term memory, which means that past output would become their training data.

An LSTM increases the short term capacity into a long term capacity and adds new architecture to store this new memory in the form of gates. In short, each neuron/cell now contains an input gate, an output gate, and a forget gate. As illustrated in **Figure 4**, the input gate determines what information should be let in; the forget date determines what existing information should be deleted because it is not important, and the output gate determines what information (past memory) should be used to predict the current output. All three gates use sigmoid functions to evaluate information which enables them to do back propagation.

6.2 Results of the LSTM Neural Network

I used the Optuna-based Tuning to train my Neural Network and optimized the learning rate, batch rate, hidden size, and dropout rate. Due to the strong computational requirements of Neural Networks, I could not use conventional techniques such as grid search and hence opted to use an Optuna-based Tuning algorithm to train my network.
This method shortens the complexity and Computational time of grid search by creating a search space (a set of hyperparameters) to optimize on. Using the Tree-structured Parzen Estimator, we can exploitation by getting the algorithm to focus on rich areas of the search space and choose new areas to avoid the local optima. It also prunes the tree in order to save time from tuning bad models.

Model	Test MSE Score	Train MSE Score
Baseline AR(2) Model	5.04941	1.23759
LSTM Neural Net	1.2472	0.9554

As seen in the tables, the Neural Net greatly outperforms the baseline AR(2) model and there is a small difference between the testing and training MSE scores, meaning that our model can be considered generalizable. As seen in the table, we see that the MSE score of the Lasso Model is slightly worse than that of the Neural Network when stratifying against the test data. However, due to a smaller difference between the testing and training error, we can deduce that the Neural Network is a better model due to its generalization potential.

7 Conclusions

In this paper, I investigated three models: the Lasso model, the modified GARCH model, and the Neural Network Model. Based on our results, we find that the LSTM Neural Network is the superior choice in both predictive capability and generalizability. The most likely reason for this outcome is the nonlinear nature of stock dynamics and the LSTM's ability to capture complex temporal dependencies.

The Lasso model, while effective in reducing overfitting and handling high-dimensional data, operates under the assumption of linear relationships, which limits its ability to capture the intricate, nonlinear patterns that characterize stock volatility. The modified GARCH model, designed specifically for modeling financial time-series volatility, also assumes a linear relationship and suffers from the inefficiencies of the sentiment analysis. This makes the addition of the sentiment score insignificant which causes it to fail.

In contrast, the LSTM Neural Network leverages its memory cell structure to retain and process long-term dependencies, allowing it to identify patterns that extend across time. Its ability to model both the sequential structure of volatility data and the nonlinear interactions between market sentiment and price movements gives it a distinct advantage. Furthermore, the LSTM's capability to handle noisy input data and dynamically adjust to shifting market conditions makes it more robust and adaptable than the other models.

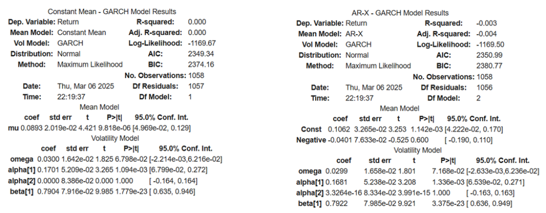
Overall, the superior performance of the LSTM Neural Network highlights the importance of using models that can accommodate dynamic and non-linear trends.

8 Limitations

There were some limitations that could impact the weight of the results. A potential candidate was handling articles that were published on weekends or days where there was no trading. Articles published during weekends or holidays were not considered which means that 570 days worth of articles were not taken into consideration when predicting market sentiment. As I did not want to include these articles on a Monday or the next trading day, I opted to simplify this by ignoring articles altogether.

A second potential limitation is knowing the date when the article was published. As the exact time the article was published is unknown, it is impossible to know if the article would be published before or after the market close on day t. Consequently, I make the implicit assumption that all articles are published while the markets remain open; an assumption that could very well be incorrect.

Another limitation that could impact the results was my rationale for filtering out certain articles if they did not contain any words or topics that could affect stock prices. Although I was quite thorough with the key words, there will still be some articles that will be left out. However, as the total list of articles over the past 5 years exceeded 70,000, it was the only feasible option to save computational time.



```
                 Constant Mean - GARCH Model Results
Dep. Variable: Return                 R-squared:        0.000
 Mean Model:   Constant Mean          Adj. R-squared:   0.000
 Vol Model:    GARCH                  Log-Likelihood:   -1169.67
Distribution:  Normal                 AIC:              2349.34
  Method:      Maximum Likelihood     BIC:              2374.16
                                      No. Observations: 1058
   Date:       Thu, Mar 06 2025       Df Residuals:     1057
   Time:       22:19:37               Df Model:         1
                          Mean Model
      coef    std err     t      P>|t|     95.0% Conf. Int.
mu 0.0893 2.019e-02 4.421 9.818e-06 [4.969e-02, 0.129]
                         Volatility Model
           coef    std err     t      P>|t|      95.0% Conf. Int.
 omega   0.0300 1.642e-02 1.825 6.798e-02 [-2.214e-03,6.216e-02]
alpha[1] 0.1701 5.209e-02 3.265 1.094e-03 [6.799e-02, 0.272]
alpha[2] 0.0000 8.386e-02 0.000 1.000      [ -0.164, 0.164]
 beta[1] 0.7904 7.916e-02 9.985 1.779e-23 [ 0.635, 0.946]
```

```
                    AR-X - GARCH Model Results
Dep. Variable: Return                 R-squared:        -0.003
 Mean Model:   AR-X                   Adj. R-squared:   -0.004
 Vol Model:    GARCH                  Log-Likelihood:   -1169.50
Distribution:  Normal                 AIC:              2350.99
  Method:      Maximum Likelihood     BIC:              2380.77
                                      No. Observations: 1058
   Date:       Thu, Mar 06 2025       Df Residuals:     1056
   Time:       22:19:37               Df Model:         2
                          Mean Model
            coef    std err     t      P>|t|     95.0% Conf. Int.
  Const   0.1062  3.265e-02 3.253  1.142e-03 [4.222e-02, 0.170]
Negative -0.0401 7.633e-02 -0.525 0.600      [ -0.190, 0.110]
                         Volatility Model
             coef       std err     t          P>|t|      95.0% Conf. Int.
 omega   0.0299       1.658e-02 1.801      7.168e-02 [-2.633e-03,6.236e-02]
alpha[1] 0.1681       5.238e-02 3.208      1.336e-03 [6.539e-02, 0.271]
alpha[2] 3.3264e-16 8.334e-02 3.991e-15 1.000      [ -0.163, 0.163]
 beta[1] 0.7922       7.985e-02 9.921      3.375e-23 [ 0.636, 0.949]
```

Figure 1: Baseline and Exogenous model

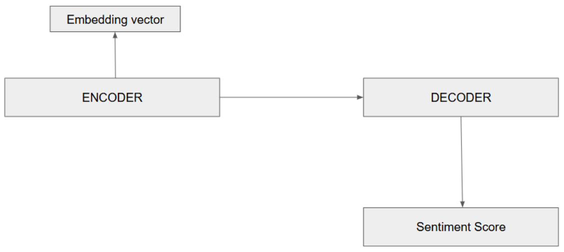


Figure 2: Transformers

9 Figures

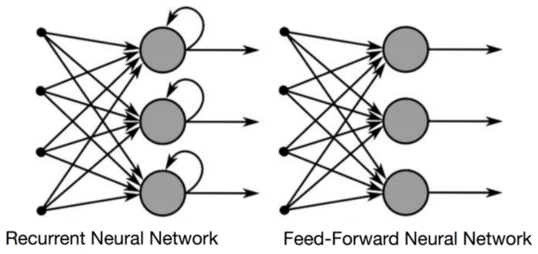


Figure 3: Recurrent Neural Networks

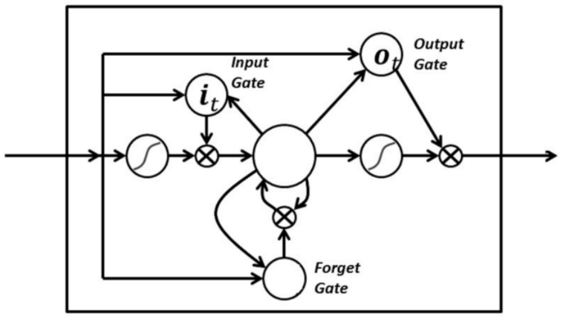


Figure 4: Memory Gates

10 Appendix

“What Is Self-Attention?” H2o.ai, h2o.ai/wiki/self-attention/.
Ph.D, Jacob Murel, and Joshua Noble. “Encoder-Decoder Model.”
Ibm.com, 2 Dec. 2024, www.ibm.com/think/topics/encoder-decoder-model.
“Medium.” Medium, 2025, medium.com/@sharifghafforov00/understanding-encoders-and-embeddings-in-large-language-models-llms. Accessed 10 Mar. 2025.
Donges, Niklas . “Recurrent Neural Networks 101: Understanding the Basics of RNNs and LSTM.” Built In, 2019, builtin.com/data-science/recurrent-neural-networks-and-lstm.
IBM. “Recurrent Neural Network (RNN).” Ibm.com, 4 Oct. 2021, www.ibm.com/think/topics/recurrent-neural-networks.
https://github.com/aakashg2/MarketSentiment