Efficient and Effective Methods for Predicting Stock Market Prices

Adrianna Liu, Stanley Wu, and Patrick Yoon

1 Abstract

Artificial Intelligence is often paired with big data and big companies. It can seem daunting to perform stock market predictions without access to reliable and large data sets, or massive amounts of computing resources. However, we show that neither are necessary to achieve reasonable results. Specifically, we show that using the simple algorithms Long Short Term Memory, Decision Tree Regressor, and Hidden Markov Models can achieve surprisingly strong results even with only five hundred days worth of stock and news sentiment data.

2 Introduction

Over the past decade, machine learning has made significant progress in predictions with deep neural networks that consists of many complex layers and weights. It has become common to throw data into these large neural networks, train for a day on powerful GPUs, and spit out very impressive results. With a massive amount of data, this seems like a very desirable approach, and the stock market offers exactly this kind of opportunity. Opening prices, closing prices, and volume, are numbers that have been recorded dating back to the 1970s, with datasets in the thousands (Kaggle is one of many resources that provides such open source data). It however remains difficult to predict the stock market, partly due to the amount of factors that influence the stock market, and also how quickly and aggressively the stock market can fluctuate. This makes deep neural networks with massive amounts of stock data difficult to maintain and use, since constant expensive updates are required, making this approach ineffective for those without access to virtually unlimited resources.

A common approach to deal with ineffective models is to find data to augment with the current dataset in hopes of improving the model with this additional information. For the stock market, it is heavily influenced by the news, making it a perfect addition to predicting stock prices. We focus on news data (particularly news sentiment), since it has high influence on the market, and can be a source of extra information to predict the stock market. We chose to perform our analysis on Google due to its large presence in the economy, and subsequent higher influence from media platforms and news. However unlike open and closing prices which date back multiple decades, it is difficult to gain access to open source news data, as they are usually locked behind price barriers of large data companies. A standard approach of throwing a large neural network at stock prediction is limited by computing power and data availability.

Motivated by these concerns, we approach the modeling of the stock market using three less expensive, and more "quick-and-dirty" methods. Specially, we analyze the effectiveness of Long Short Term Memory (LSTM) using Keras (done by Stanley Wu), a hand written Decision Tree (done by Patrick Yoon), and a prewritten Gaussian Hidden Markov Model (done by Adrianna Liu), on limited data and compare their results when trained solely on Google's stock market data (baseline model), with when they were trained on Google's stock market data and some augmented news sentiment data (news model). To align with motivations, we build our models using stock data obtained from an open source Kaggle dataset, and news sentiment data was found on an open source OpenML page (links will be provided in the methodology). Due to the nature of finding news sentiment dataset as explained above, our news dataset consisted of around two years worth of news data, leading to around 500 days of stock market augmented data.

LSTM is well known for its performance with time series regression, so it will act as our basis for our two other approaches. A commonly used classification and regression technique is the decision tree approach, a strategy we chose due to the high dimensionality of our data (especially when augmented with news). A Hidden Markov model was chosen since it is a simpler version of recurrent neural networks, making it ideal since an RNN performs better on larger datasets (a luxury we do not have). We show the results of using these simpler regression models, and compare the changes of each baseline model with their respective news model. We also interpret these results with respect to their effectiveness in stock market usage, mainly focusing day to day accuracy, trend prediction, and overall effectiveness.

3 Methodology

Since news data and stock market data were not obtained together, some preprocessing was necessary to arrive at data suitable for training. The stock market dataset was a standard dataset found on Kaggle here consisting of high, low, open, close values among others features. The news sentiment dataset was found on OpenML here, and is derived from the entertainment media company Mashable, consisting of a variety of features from polarity to the weekday in which it was published (i.e. Wednesday ... etc). Since features such as weekday, number of videos ... etc, do not have relation to news sentiment, they were removed. In the resulting dataset, articles of the same day were grouped together, and feature values were averaged across articles of the same day to achieve a per-day assessment of news sentiment based on all the articles of each day. This dataset was then merged with the Google stock data according to date, leaving a dataset of about 500 rows in which each row represents a day of both news sentiment features and stock data. This was then split into a training set of 300 points, validation of 100 points, and testing of 100 points (Roughly 60, 20, 20 split). Notice that order was maintained because in time series analysis, each day leads to the next. A MinMaxScalar was then fit over the training points, this scaler will be used on the validation and testing set before they are passed into the trained model. This is standard preprocessing that is used in each model approach except for HMM, with more specific specifications in each to fit their respective models. More about where to find code for this can be found in the Appendix.

3.1 LSTM

An LSTM model is a popular choice for time series predictions, so we consider it the model to compare to for our two other approaches. We followed some guidelines on using Keras for our LSTM from these writeups from machinelearningmastery.com here, and prepared our data according to how LSTM takes as input as mentioned in this machinelearningmastery.com article here. Since our data contains a number of features about prices and news sentiment per day, the input is taken in as a sequence of past days and their features to predict the next day's "Middle" value, which is calculated as the average of that day's "High" and "Low" value. To accommodate for LSTMs expected inputs, a create_dataset was implemented with a look_back input that controls the number of previous days (rows) used to predict the next day's (row) Middle target value. The model was then created to take in an input shape corresponding to this number of look_back days. For example, if there were 30 features, and the look_back day was 2, create_dataset would create a list of list of look_back amount of list of features. This would then be reshaped so that each point has shape (look_back, features). In the creation of the model, 32 units was chosen as the number of units, and the last Dense unit returns a value of dimensionality 1, representing the predicted scaled middle value of the next day. Since we are using the LSTM model to perform a regression task, mean squared error was used as the loss function, with the adam optimizer since it works well with LSTM. In hyper parameter tuning, the validation set was used for comparison in different epochs, batch sizes and look_back values for the LSTM model.

3.2 Decision Tree

We create our own decision tree with help from articles from machinelearningmastery.com here and towardsdatascience.com from here. For a regression decision tree, the main idea was finding a best rule to split a node at each level of the tree. The inputs of the decision tree are the features used to predict the value of a stock. These features include the prices of when the stock opens/closes in the market and the highest/lowest prices of the stock of a given day. Along with those variables contain the number of shares and volume number, which make up the baseline model. We also included a separate model based on using news as a factor. This news model contains the same features as the baseline in addition to eleven other features, some of which represents an average number of articles published in a specific genre for a given day. These features are: lifestyle, entertainment, business, social media, tech, and world. Some other features include the average global and absolute subjectivity, sentiment polarity, and rate of positive/negative words of all the articles published on Mashable on a specific day. The outputs of both models would be the average stock price of the day after the given day. To create a decision tree, the algorithm starts off at the root of the tree and determines what the most impactful feature is and what the threshold is in order to create the split. In order to conclude what the best feature is, we iterate through each of the features, then for each feature we iterate through each value in the dataset and split the dataset at that value. From there, we calculate the residual sum of squares (RSS) of both ends of the split and add them together.

$$RSS = \sum_{\text{left}} (y_i - \hat{y_L})^2 + \sum_{\text{right}} (y_i - \hat{y_R})^2$$

By calculating the RSS for every value in every feature, we check to see what our smallest RSS value is, which corresponds to the best feature and threshold for that node. We continue to move down the tree recursively until we either hit our maximum depth or there are no values left in the dataset to split. In order to determine the optimal maximum depth of the decision tree, we used our validation set (of 100 points) to tune it. We created multiple decision trees with varying maximum depths and evaluated the r2 on the validation set.

We accounted for multiple factors to determine which depth was most optimal, including the time it takes to create the decision tree, overfitting, and accuracy on the validation set.

3.3 HMM

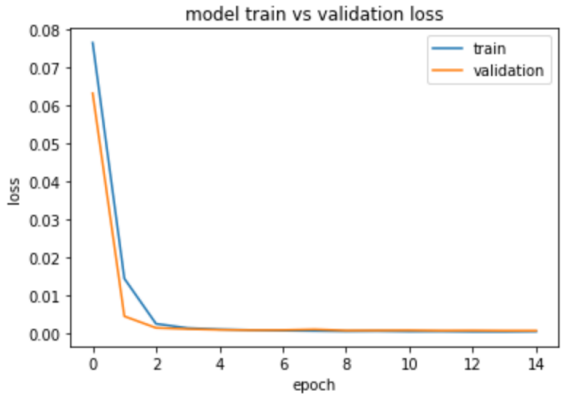
For a Hidden Markov Model, the main idea is transitioning from one state to another. In the stock market, the price of a stock often depends on factors that the investor cannot see. In other words, those would be the hidden variables. In addition, the transitions between all the factors that affect the stocks depend on the company itself — the company's decisions and policies, its financial state, etc. All of these factors can directly affect the price of the stocks, which is why it also acts as the observed data. As always in machine learning, the data was split in training, validation, and testing, and we also scaled the data with a MinMaxScaler. For features, there is stock market data for the opening price, closing price, highest price, and lowest price. We decided to use someone else's implementation of an HMM model, specifically citing an article from rubikscode.net written by Ankara Ankan and Abinash Panda that can be found here. In this paper, it was noted that these features should be used to calculate the closing prices for a day, given the opening price as well as the data from a certain amount of days before, called the latency, of which we decided to use 10. We decided to go a bit further, and use the calculated closing prices to actually find the middle prices.

However, rather than directly using the given opening, closing, high, and low stock prices to train the model, Ankan and Panda recommended calculating the fractional changes as that creates a percentage and makes it easier to model. To actually predict the stock prices, you first need to train the HMM model from the opening, closing, low, and high observations. As recommended by the article, we decided to represent the distribution as a multinomial Gaussian. This was done by using the hmmlearn package and its fit() method to calculate the parameters. In predicting the closing price, this assumes that you know the opening price as well. Using the fractional change of opening and close prices that you calculated before, you can then compute the closing price that maximizes the probability of that observation. Then, the wanted middle price is simply (open + close) / 2. In tuning, we tested the validation set with a different number of hidden states instead of the default 4 to see if the predictions would change. In this case, it appeared that altering the number of hidden states did not change much.

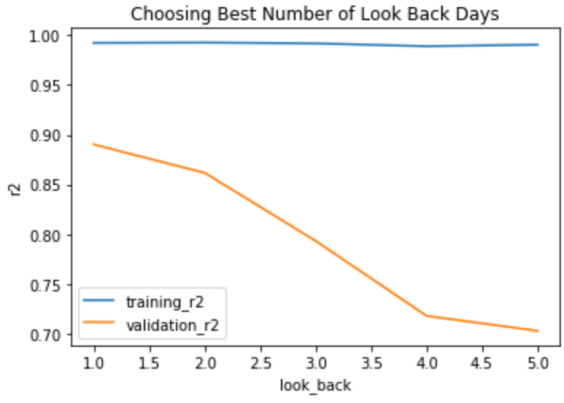
4 Experiment and Results

4.1 LSTM

The main experiment we conducted on LSTM, was to see how well the baseline model performed against the news model. To do this comparison, we used both mean squared error as well as r2 score. Mean squared error was used as the loss function for the model, and r2 was used to score against the decision tree (allowing us to compare the effectiveness of LSTM vs. decision tree). As seen below, loss converges almost immediately (couple of epochs), due to the small size of our dataset.

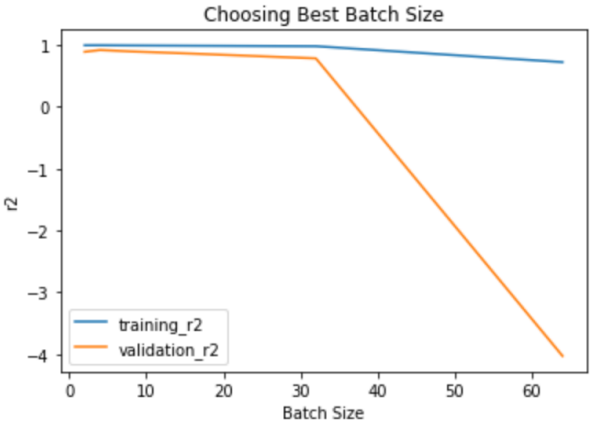


This convergence is quick for both training and validation, so much to the point that validation MSE is less than training MSE (again results of a small dataset). Because of this, the r2 score is going to be more informative on the model's performance. When we compared the baseline performance to the news performance (with original hyper parameters), we see that there is not much difference between the r2 scores. This suggests that news sentiment isn't making a large impact on the regression, but that it is also not making a negative impact on the regression. Since they both perform identically as well (see Table 1 for full results), the rest of hyper parameter tuning was performed on the model augmented with news sentiment.



	Training		Validation		Testing	
	Baseline	News	Baseline	News	Baseline	News
LSTM	0.99	0.99	0.91	0.91	0.96	0.95
Decision Tree	1	0.99	0.90	0.89	0.91	0.89
HMM	1	0.98	0.95	0.73	0.98	0.81

Table 1: *Each model's r2 scores for predictions on the training set, validation set, and testing set.*

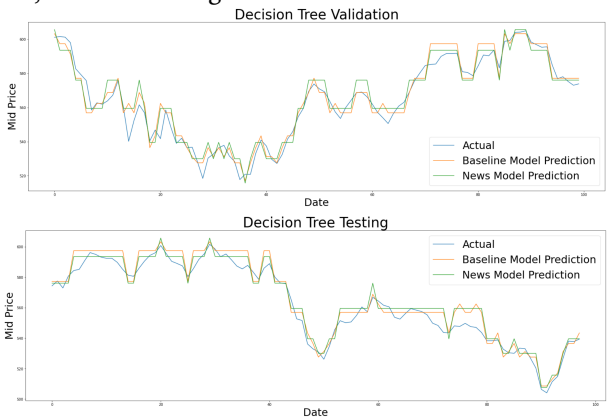


When testing the performance of different look_back days, epochs, and batch sizes, we found that 10-15 epochs, look_back day of 1, and that a batch size of 2-8 led to the best performance. These seem to be natural conclusions since the size of our dataset meant having more look_back days, more epochs, and higher batch size would lead to more overfitting.

Taking a look at the above figure, both models performed very well (baseline vs news), yet it is important to notice that there is a "lag" in both models predictions. This is likely a byproduct of the way LSTM predicts from constructed data points but it is nonetheless very important since day to day stock trading relies on accurate predictions to the day. Overall, LSTM is a basic, fast to train time series model that is often used for time series predictions. It performs very well (training/validation/testing scores) regardless of whether or not news data is supplemented, however in terms of utility in stock market trading in either case, it does not offer much utility beyond a simple regressor. While it fits the data very well, the time lag and overall "noisiness" of the output makes it tough to use in conjunction with trading. While it is possible that this can be explained by the low amount of data being used, given its strong performance on not only the training set, but validation and testing set as well, it is safe to assume that more data will likely show the same results, but also likely show the same flaws.

4.2 Decision Tree

We created separate decision trees to represent our baseline model and our news model. We check all the decision trees with a maximum depth between three and fifteen and conclude that the optimal depth for the baseline model is 4 while the optimal depth for the news model is 5. From here, we are able to predict any output given a row of feature variables. With both decision trees, we were able to see how well they performed by comparing their r2 scores (see Table 1). We can also compare their performance by graphing each of the decision trees' predictions compared to the actual values. Below, we show this for the validation set, and the testing set.

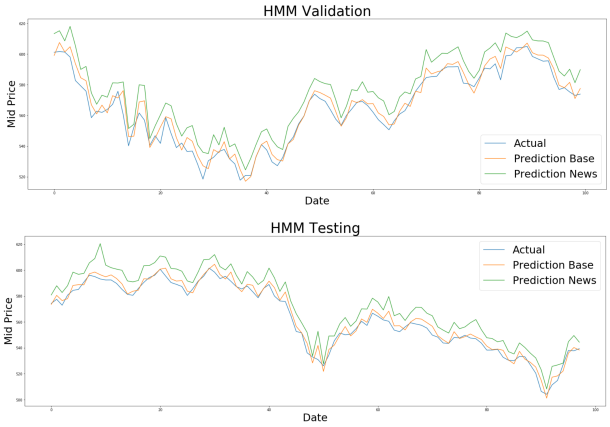


One of the main distinctions with both decision trees is that the graphs tend to be quite rigid and almost

"blocky" at certain points. This is primarily due to the fact that the regression tree we built isn't actually able to calculate continuous values. With the way we set up our split function, when we are at our maximum depth, we take the average y value of the remaining training set and deem that as our prediction. Therefore, even before we start predicting, we know that the predictions are predetermined in our decision tree. The more levels we add to the tree, the more prediction values we will have, but this also increases the likelihood of overfitting. Due to this, decision trees lack the ability to be consistently accurate in predicting day-to-day values, as two rows of features with similar but not exact values can output the same prediction. However, we can clearly see from the graphs that a decision tree is excellent at predicting the overall trend of the prices. While the predictions tend to be on the stiffer end, we can visualize the overall pattern with ease. Considering all these factors, we are hesitant to conclude this algorithm as efficient. The decision tree is able to predict overall trends, but not being able to accurately predict a day-to-day price hinders its overall usability. On top of that, in order to train a larger decision tree takes an absurd amount of time. Finding the best feature and threshold for a node takes a polynomial amount of time, and we are splitting a node into two children until we hit the maximum depth. This results in $O(2^n)$, which is incredibly slow compared to other algorithms. Therefore, compared to our other algorithms and even more advanced algorithms, decision trees are quite inefficient.

4.3 HMM

Using a Hidden Markov Model, we tested two models, one without a news sentiment and one with. Predicting the stock prices without news sentiment was more straightforward and produced better results. In fact, looking at the r2 score in Table 1, we can see that for the training, validation, and testing data, the HMM worked significantly better when news sentiments were not included. In this version of the HMM we implemented, the main focus was calculating the most optimal fractions from the states of the previous ten days, which include the opening, close, high, and low prices. We then use them to compute the most probable state for the current day. These fractions are the features passed into the model. The fractional high ((high - open) / open) and low ((open - low) / open) computation remained the same for both models, but when we added in the news sentiments, we had to come up with a way to associate the news with price. Thus, we decided to multiply ((close - open) / open), the original fractional change computation, with the news sentiment data, because originally, the calculated frac_change value ((close - open) / open) was used to calculate the closing and middle prices. However, as seen from the r2 scores, this may not have been the best way to incorporate the news sentiment into the model.



As such, when you observe the plots for the testing data and the validation data, you see that the graph for the news model has a very similar shape to the baseline model, just higher up on the graph. However, going back to the baseline model, we see that the HMM performs rather well, and often overshoots the predicted price. This can be attributed to the fact that in the HMM model we used, the closing price tends to be predicted a little higher, which also results in a higher middle price. Looking more closely, there seems to be a delay when you observe the day-to-day modeling for the prices, as the prediction usually changes direction after the actual price. However, there are regions on the baseline graph that match with the actual graph better than others. This is related to the actual price changes, and whether it happens smoothly or rapidly. One specific example in the testing plot is that the baseline model actually predicted a rebound one day earlier, as seen in the section right around date 50. It is very interesting that the model was actually able to predict the rebound ahead of time, albeit incorrect. The model eventually rebounded again around the correct time, but this was an interesting observation as predicting rebounds in the stock markets are an important aspect when it comes to overall trend prediction. All in all, it appears that the model predicts less accurate prices when the changes in price are more rapid. As for the graph with the news sentiments, as we noted before, the shape of the graph was very similar to that of the baseline model, the prices predicted were just very high. In order to think of a way to resolve this issue, we noted that the entirety of the graph was about the same price higher than the prices of the baseline model. Thus, if we were to scale or translate the graph down a little, we would perhaps be able to see a better model comparison as well as a better r2 value.

5 Conclusions

Approaching the problem of predicting stock market prices with minimal data can be done in two ways. One is a regression/predictive approach with LSTM and Decision Trees, and the other with a probabilistic approach using HMM. Based on Table 1 of r2 results,

all three of these models fit extremely well on training data with r2 scores above 0.95. (This is expected of HMM and Decision Tree approaches). These models also perform very well on validation and testing sets, with r2 scores all above 0.8. However, due to the minimal amount of data, it seems adding news sentiment does not improve the models. In the case of decision tree and LSTM, there doesn't seem to be a difference in performance, but HMM suffers from this additional data. In addition, while LSTM does not take very long to train, the decision tree and HMM approach takes longer, but not so much so that it is unreasonable to use. We conclude that under data availability and computing resource constraints, the three simple models we propose perform well, and continue to do so regardless of if the model's training features included news sentiment data. We show that large many layered neural networks and massive datasets are not required to make good predictions about the stock market. Rather using simpler models such as LSTM, decision tree, and HMM can be very informative and accurate when it comes to predicting stock market prices.

6 Appendix

First, you will need to pull the code from our Github repository here. You will need to have Python installed along with these libraries: numpy, matplotlib, pandas, seaborn, hmmlearn, and scikit-learn, keras (the file "requirements.txt" can be helpful to install all at once). Then, please follow the corresponding instructions for each machine learning algorithm. For the model notebooks themselves, they are all in the folder called "models," named respectively to their model type. As for the initial data wrangling and datasets, they are in "models/dataset."

6.1 Decision Tree

After opening "Decision Tree.ipynb", you can proceed to run every cell in order. Any cell directly below a text in bold red font may be skipped as they are examples or code that take an incredibly long time to run and are insignificant to the actual experiment/result. If neither the code or the dataset has been modified, you should be able to run all the other cells quickly (except for two cells that create the decision trees which takes a little longer than instantaneous). You should get the exact same results as we did, as there were no instances of using any randomness in the experiment, including splitting the dataset and creating the decision tree.

6.2 LSTM

Similar to the Decision Tree, you can proceed to run every cell in order in "LSTM.ipynb", however since LSTM training does not take as long as Decision Tree, there is no need to skip over any blocks. Also unlike the decision tree algorithm, LSTM does incur some randomness in training. This means the results may not be identical to the ones presented, however they should remain relatively similar.

6.3 HMM

As with the Decision Tree and the LSTM, you can also run every cell in order in "HMM.ipynb", but closing_and_middle_price_in_range will take some time to run. Like the Decision Tree, you should be able to get the same results as there is no randomness in fitting the model. Thus, if you use the same training set, the data should have the same result.