Predicting Stock Prices with Machine Learning

Riley Oest Antoine Sfeir Narek Bayramyan Jacob Maganizo Kapita

Introduction

Algorithmic trading is a niche in computer science that has increased in popularity since the emergence of machine learning and AI. Many of the top algorithmic traders use a large scope of algorithms to their benefit, including high-frequency trading, and sentimental analysis. The advantage of algorithmic trading is that it takes out the challenges of human psychology. Even with a sound strategy, traders can make mistakes due to psychological biases. Our goal in this project is to create a machine-learning model to predict the prices of stock.

Data

We chose to model the Dow Jones Industrial Average Index (DJIA). We were able to source the dates and prices of the index using Y!Finance API.

Methods

- We have built our application to be able to dynamically predict any stock, however for presentation, we chose the DJIA index. Input data consists of a selected number of input days.
- The algorithm used to predict the stock chart is a sliding window. (Using previous x days to predict the $(x+1)^{th}$ day).
- The sliding window algorithm takes the previous x_i days as input to predict the x_{i+1}^{th} day, where 'i' is optimized from running multiple models of different inputs. The window of input is then moved over to predict the x_{i+2}^{th} day, and so on until iterating through the entire dataset.
- For example, with an input size of 3, we have x_1, x_2, and x_3 as inputs and we predict the value of x_4. Since the algorithm iterates through the entire dataset by the sliding window technique, the next set of inputs in this example would then be x_2, x_3, and x_4 to predict x_5.
- Since the index's all-time price data ranges from \$1,000-\$30,000 (approximately) and neural nets don't work well with large ranges of inputs, we have used normalization to bring every price between a value of 0 and 1.

Dense Input	Input	(None, x_{i+1})
Input Layer	Output	(None, x_{i+1})

⇓

Dense Layer	Input	(None, x_{i+1})
Dense	Output	(None, 64)

⇓

Dense Layer	Input	(None, 64)
Dense	Output	(None, 64)

⇓

Dense Layer	Input	(None, 64)
Output Layer	Output	(None, 01)

Methods Continued.

- We don't know the number of input days that will be optimal for predicting any particular stock, and day-to-day that optimal number of inputs is likely to change. To accommodate this dynamically, we have constructed an algorithm to train multiple models via a specified range of input sizes. The results of each model are documented so that we can select the best one to do our predictions.
- After selecting the number of input days, each model generates training and validation data based on the inputs (close prices and dividends paid) and targets (close price). For each size of inputs, a time series is generated using Keras's TimeseriesGenerator to split the time series into training and validation sets. Each training set includes all prices except those recorded in the last 1000 days, which is left for validation.
- During training, the model's weights are adjusted according to the Adam optimization algorithm, which dynamically adapts the learning rate based on the gradient and slope of the mean squared error (loss function).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i)^2$$

Results

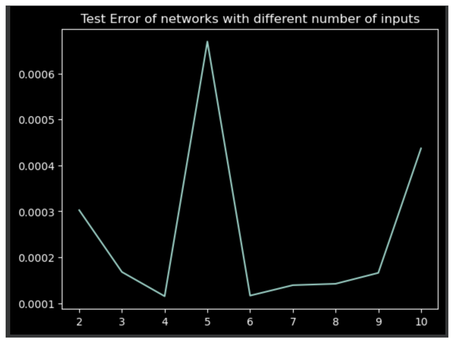

Figure 1. Graph of Error vs. Number of Inputs Trained For each Model.

```
True Error for best model (ie: 4 inputs) = 581.765208737297
Normalized Error for best model (ie: 4 inputs) = 0.00011499148968141526
```

Results of Model's Predictions

Since Min-Max Normalization was used on the input prices, we compute the error using the normalized values. However, to plot the results, we have denormalized the values to see the true prices of the index.

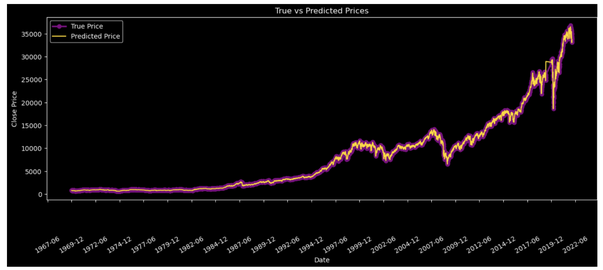

Figure 2. Graph of True vs Predicted Prices

Our best model shows that an input size of 4 days is optimal for predicting the "next days" of the stock. The true error is very high given the large range of prices, so the MSE adds up over a long time. However, the error computed on normalized data is 0.0001.

Conclusion

We have used the DJIA Index to show the validity of this stock-predicting tool. However, any stock can be used in this application. The chart of true and predicted prices looks very accurate, granted the domain (number of dates) is very large. Another note, is that our true error is not very attractive (580). Needless to say, this model cannot guarantee to be accurate in predicting future prices.

A future goal for this application is to implement automatic trading when a price is predicted to go up the next day. Another future goal is to use all the stocks comprised in an index to track the movement of the index itself. This model would have a lot more input data and thus, more information for a neural net to learn on.

References

- https://towardsdatascience.com/is-it-possible-to-predict-stock-prices-with-a-neural-network-d750af3de50b
- https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/TimeseriesGenerator
- https://finance.yahoo.com/quote/%5EDJI/
- https://www.tensorflow.org/api_docs/python/tf/keras/Sequential