Machine Learning Techniques and the S&P500

Matthew Hou, Martin Iglesias, Xiang Li, Robert Nash, Ruilin Zhang

Abstract

For many years, people have tried to predict future earnings in the stock market. The stock market has been considered unpredictable and due to its efficient nature, argued that future performance cannot be predicted based on past performance. This research used various machine learning techniques to predict future prices of the `S&P500` within a certain level of error. The four methods used were TensorFlow, Multi-Layer Perceptrons, Support Machine Vector and Long-Term Short Memory. Overall, all techniques used show to be not exactly accurate in being able to predict future stock prices. Long-term Short Memory was the most accurate but like all the techniques, not all information is considered within the calculation. Outliers such as socioeconomic events that can have a significant impact on the market, were not available to the algorithm. Although some trends can be induced by using machine learning algorithms, it is to our conclusion these machine learning techniques are not suitable for stock price prediction and that other methods should be explored, such as Bayesian Networks, that can include other outliers in the decision.

Introduction

According to legend, in 1815 while Napoleon's army was being defeated at the battle of Waterloo, a carrier pigeon of the news was heading to London. This pigeon belonged to investor Nathan Rothschild. With his new knowledge, he was able to make a fortune in the British market by buying government bonds before any other traders knew the outcome of the battle. Even in 1815, people were trying to figure out new ways to predict the future of the market. This can prove to be very lucrative for those who predict accurately(The Rothschild Archive 2020). One of the key issues with trying to predict the future of the stock market revolves around the efficient market hypothesis or EMH. Developed by Eugene Fama's Ph.D. dissertation in the 1960s, this hypothesis states that at any given time, stock prices fully reflect all available information. Past price and data have absolutely no bearing on the current or future price of the stock and therefore technical analysis cannot predict stocks. Burton Malkiel, author of *"A Random Walk Down Wall Street"*, was a huge advocate of EMH and argued that each day, stock prices are completely random and unpredictable(Downey 2020).

In this research, machine learning will be used to predict future prices of the `S&P500`, a security portfolio that reflects the value of the `500` largest publicly traded companies. Four different methods will be used and compared to see what method, if any, supply better results than the others and to see if the efficient market hypothesis holds true. The four methods used will include TensorFlow, multi-layer perceptron (MLP), support machine regression (SVR), and long-short-term memory recurrent neural network (LTSM). An identical data set reflecting the daily performance of the `S&P500` will be used across all four methods. The results will be analyzed, and a final comparison will be made based upon the error of the techniques.

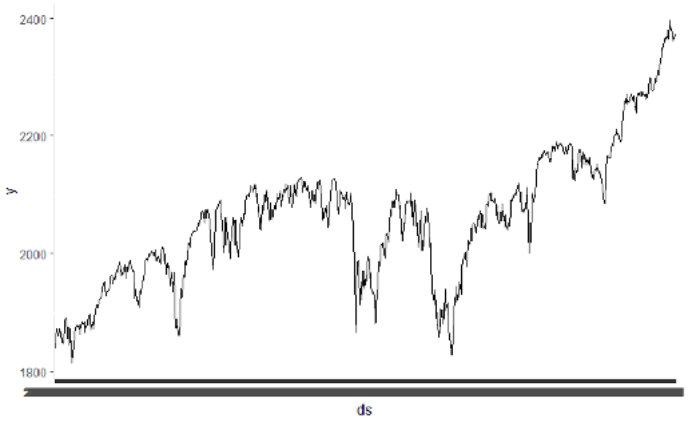
Methods

TensorFlow

The first technique we explored was TensorFlow using R. Developed by the Google Brain Team, TensorFlow is a machine learning library developed for numerical computation and large-scale machine learning (Kazemi 2020). We used the TensorFlow library and Keras library to help achieve our goal. After loading the

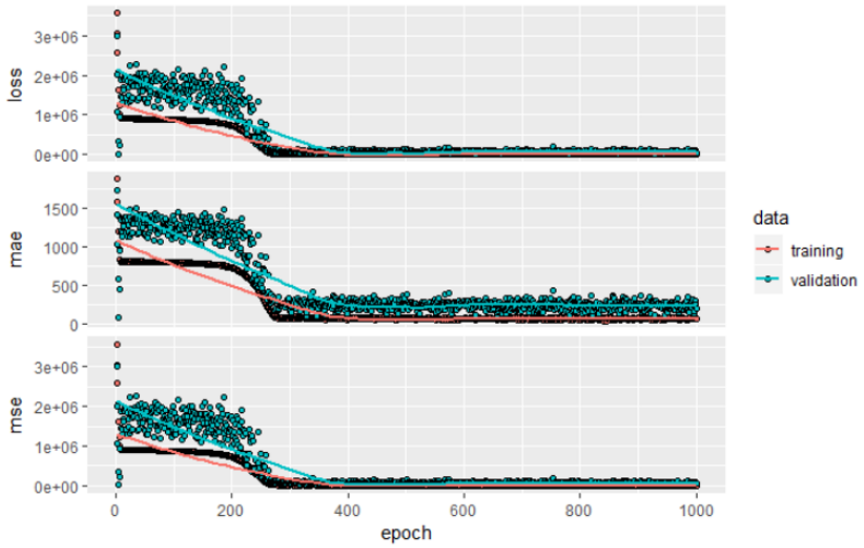
`SP500.csv` file ("S&P500," n.d.), we first cleaned the data by removing rows with `'N/A'` in the `'y'` column. These null values could have a negative impact on our results. Out of the `784` rows of data, only `27` rows had `null` values. This showed that removing the values will not have a significant impact on our predictions.

Below is a chart of the `S&P500` after removing dirty data. The x-axis represents dates and the y-axis represents the price of the stock in dollars. We can see that the general trend of the stock price is increasing, but the daily prices vary a lot and could not be shown clearly on the graph because of the size limitation.



Since we are not using other stocks' prices to predict our stock price, we separated the data into training sets and testing sets. We randomly gave `75%` of the data to training and `25%` of the data to testing. We used random sampling to minimize the impact of time. Since these data points are sorted by date, if we sampled the data sequentially the prediction might be influenced by time factor, such as depression, which will lead to an underestimate of our prediction. We then used the function `keras_model_sequential()` to initialize our TensorFlow model. Here, we added three layers, two of them were hidden layers and one of them was an output layer. We set the input shape to be the shape of our data. Now with our model initialized, we used `predict(model, data)` function to test if the model worked. After a successful run, we knew our model worked and applied the training data to train the model. We used `model%>%fit` function to train the data, store the value in history and set the epochs to `1000`. We did not set the early stop callback in our task(scikit-learn developers 2020a).

After training, we plotted the history.



Multi-layer Perceptron (MLP)

After TensorFlow, we continued to work with the S&P500 data set using our next method. The method used for forecasting is the multi-layer feed-forward network (MLP). MLPs are good for approximating solutions for complex problems. To succeed in stock price prediction, we do not need to know the exact price for each day(Turchenko et al. 2011). What we want is a trend whether the price will continue to increase or decrease. MLPs have the learning abilities to capture dynamic, nonlinear, and complicated trends of stock prices. We used built-in functions of the Scikit-Learn python package (SKL) to apply the MLP method. The model we used is the Multi-layer Perceptron regressor. This model optimized the squared loss using LBFGS or stochastic gradient descent(scikit-learn developers 2020b). The data consisted of a vector of dates and a vector of prices and we trained our model only on the date values. The date vector was input vector and the price vector was output vector. A couple rows held missing stock price values and those rows were omitted. To use SKL regressor functions, the date vector needed to be converted to a vector of integers. The first date value was 2014-3-13, and we made it a start point and assigned value 0 to it. We continued to stratify the data set into a training group and a test group. Our goal was to use early data to forecast future data. Thus, we sorted the data by date and chose the first ¾ as the training set and the rest as the test set. After the setup, we fitted the MLP regression model with the training set and then evaluated it with the test set using root mean square function (RMSE). the history.

Support Machine Regression (SVR)

Following the MLP method, we used support vector regression, or SVR. This model applied the same ideas from the Support Vector Machine (SVM) classification model but tried to predict values as opposed to a classification. This model was able to handle and recognize non-linearity in data sets such as our S&P500 data set. For this model we aimed to understand how well it can learn from our SP500 data set and compare its results to the other models.

Before we could create and train our model, we needed to prepare the input data and output data. To do this, we first imported the CSV file into a Pandas data frame, then selected all the values except for the last value as the input and all the values except the first value as output. The output was shifted one day after the input. Once this data was prepared, we could split the input and output into the training and test set which could be used to monitor how well the model is performing. Once the data was properly split, we were able to create the model with sklearn. We choose to use a "rbf" kernel here as the data was nonlinear and a C value of 10^{-3} which allowed for a larger margin of separating hyper planes, as well as a gamma value of 0.1.

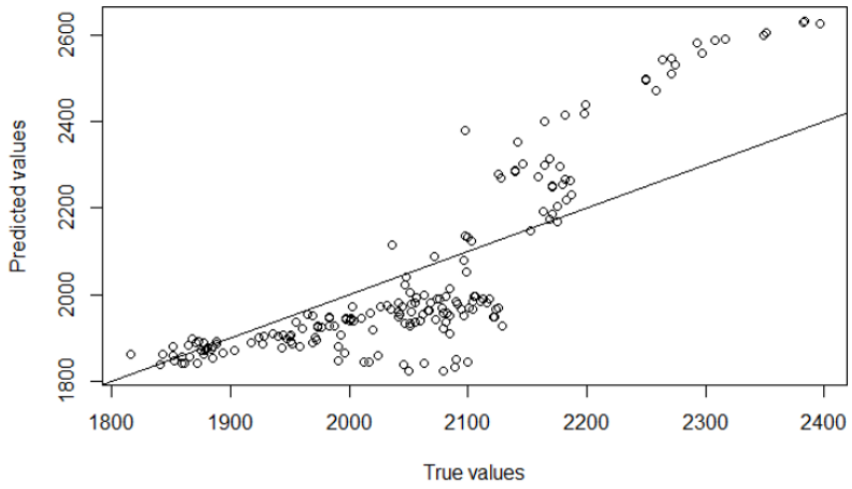
Long-Short-Term Memory Recurrent Neural Network (LTSM)

The method we used in this section is called Long-Short-Term Memory Recurrent Neural Network or more widely known as LSTM Network. This method belongs to the family of deep learning algorithms. In its architecture we can find feedback connections. The capability of this method to process the entire sequence of data gives it a big advantage over traditional neural networks. Its main architecture is built by a cell, an input gate, output gate and a forget gate. The cell remembers values over given time intervals and keeps track of the dependencies between the elements in the input. The other cells control the flow of the values should they enter the cell, remain in it, or be considered by the activation function. More in depth, this method works very well with time-series classification and prediction, that is why it is a great fit to our goal here in predicting the S&P500. What we want to do with our model is to train on some stock value data, so that it can predict the future stock values of other test data. To do so, we load in the stock value data, using the pandas library, from May 2014 to March 2017 of S&P500 and split this data into training data, 80% of it, and into test data, the remaining 20%. Afterwards, we scale our features in the range 0 to 1, so they are more convenient to work with. Then, we generate data sets using time steps of 5 days. Now, we have the data in the format that we can work with so we can start training our model. First, we generate a Keras Sequential model and we add to it a LSTM layer with 100 neurons. Then we add the output layer with 1 unit. Afterwards, we compile the model providing it our own root mean square error function and using the "rmsprop" optimizer.

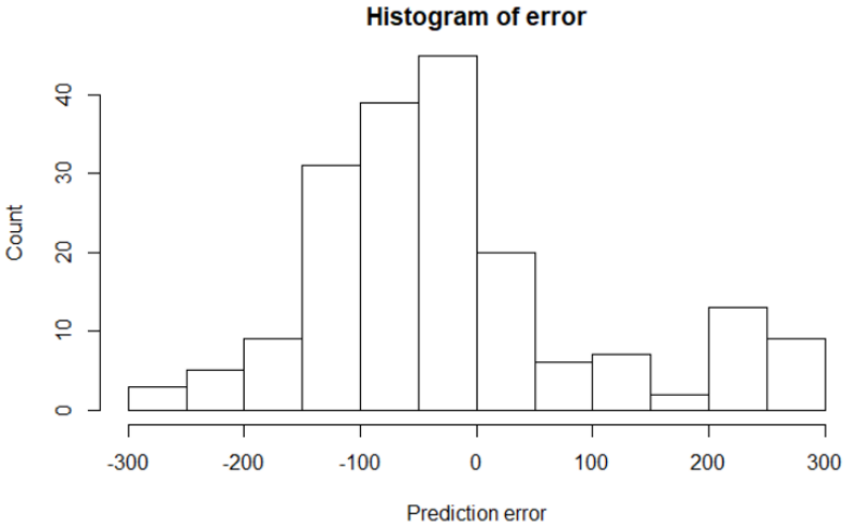
Results

TensorFlow

We can see that after `400` times of training, the mean square error and loss decreased to a suitable number which shows that our model works well on these training data. After 400 epochs, the three values stay constant. Then we use our model to predict the stock price. We applied the testing data set to the model, and used the predict(model, test) function. Plot the result of comparing the predicted values to the true values.



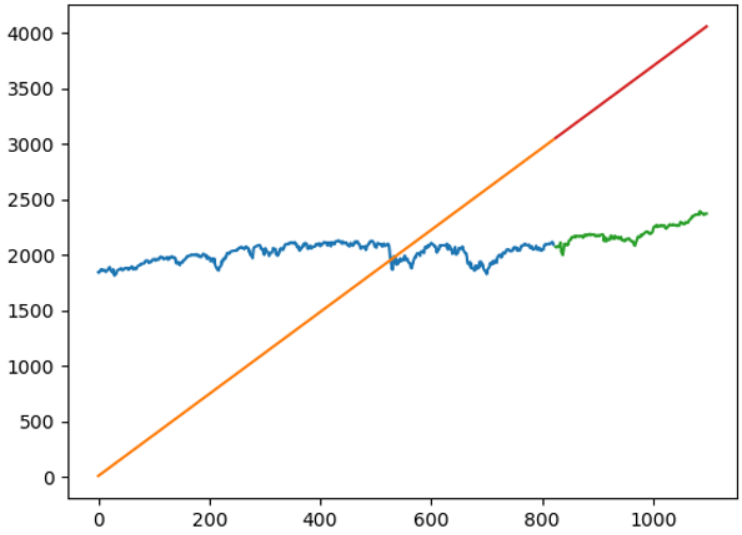
By drawing a line that points to the `point (x:2800, y:2800)`. We can see that the predicted values do not fit well on the true values. It underestimates the value when price is ≤ 2100, and overestimates the value when price is >2100. We set the `error = true_test_value - predicted_value` and use histogram to plot the error.



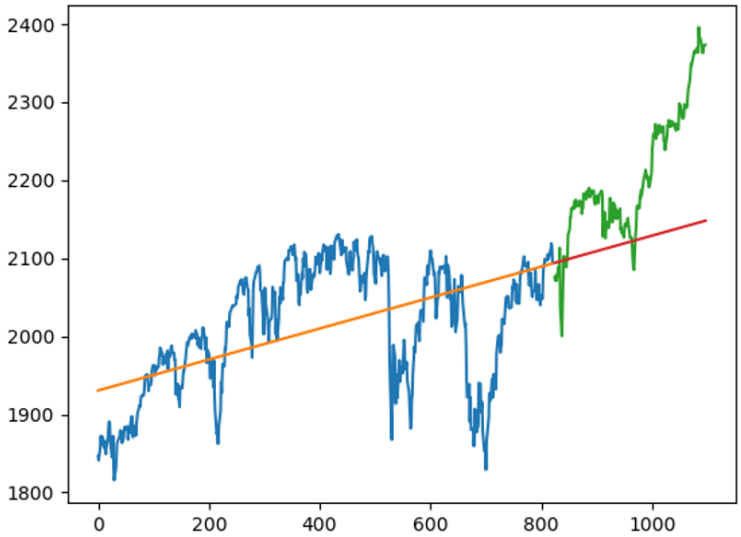
We can see that most of the prediction errors lie in the `range (-100, 100)`. But there are still many other larger errors. In conclusion, we can say that using TensorFlow to predict stock price is not perfectly accurate because of the large amount of errors happening. Based on `figure 1.3`, we can say that the model can predict a relative trend of stock price, but not the price, and `figure 1.4` also supplies evidence to support this conclusion.

Multi-layer Perceptron (MLP)

We started with a base MLP regressor. This regression model uses `relu` as its activation function. It has a single hidden layer of 100 nodes and a maximum of 200 iterations. The blue and green lines stand for the training and test target values. The orange and red lines stand for the training and test prediction values. This model has a large RMSE value for both training and test set: 976 and 1368.

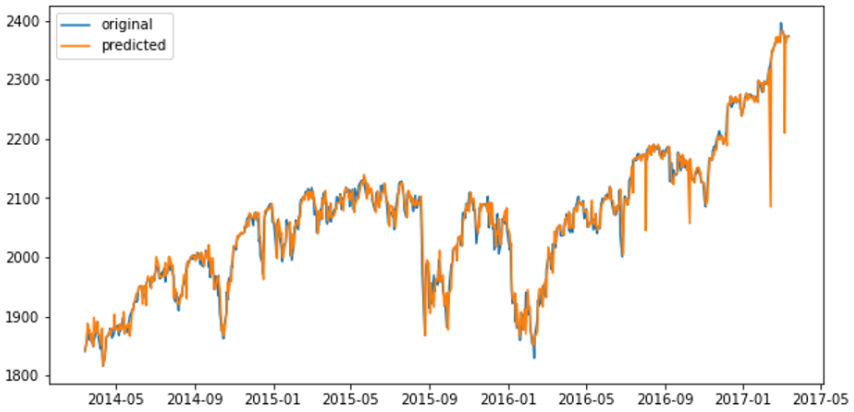


However, this model can be improved by changing the hidden layers, the activation function, and some other attributes. After several attempts, we created a new model with 2 hidden layers, 200 nodes each. We set identity as its activation function, and it has a maximum of 2000 iterations. Using this model, we can decrease `RMSE` values by more than `90%`. For the training set, the RMSE values are around 70. For the test set, the RMSE values are around 100. However, if we continue to increase the size of hidden layers, RMSE values tend to increase again, due to over fitting.

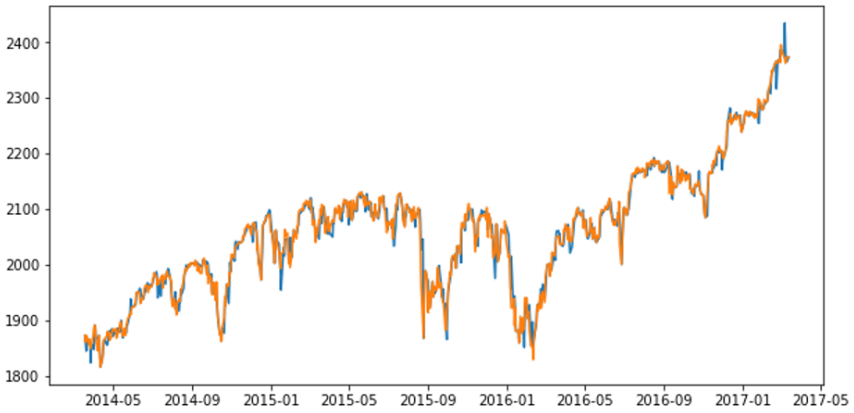


Support Machine Regression (SVR)

After fitting the model we see that it performs at `98.5%` for the training accuracy and `90.5%` for the test accuracy. Below, we can see the comparison between what the original `S&P500` data showed versus the predicted values that the model gives.

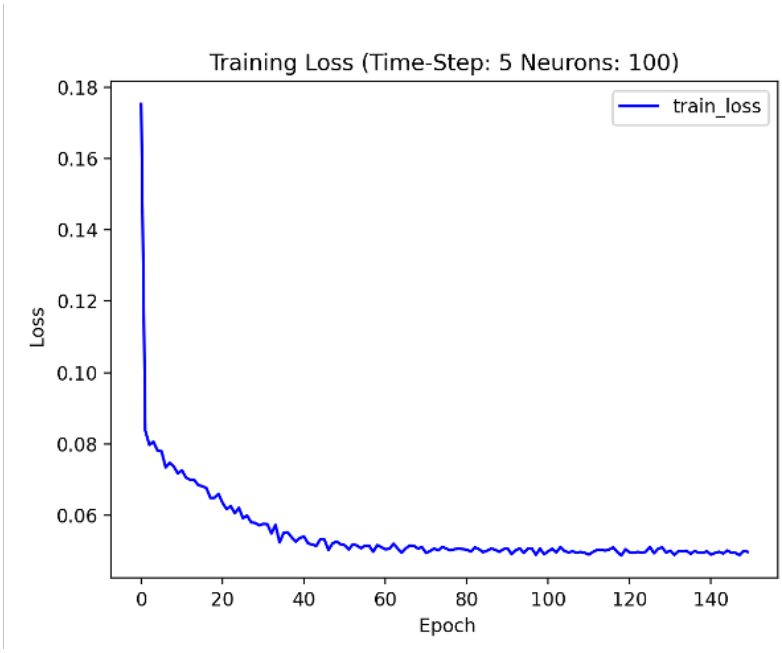


One improvement we can make to this model would be to have it look at the past 3 days instead of just the previous day's value. We can represent this change by having each element of the input being the past three days worth of values. With this change, we can improve the training accuracy to `99.0%` and the test accuracy to `95.3%`. As we can see from the difference in predictions, there are less wild spikes in the predictions as it follows slightly more closely to the original data.

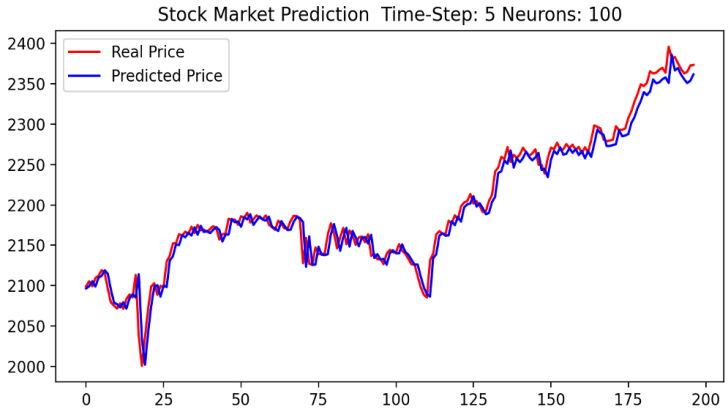


Long-Short-Term Memory Recurrent Neural Network (LTSM)

After fitting the model for 150 epochs we find that it ends up having a loss on the training data of roughly `0.02`. This makes us confident enough to go and try the model predict on our `20%` test data.



After predicting the values for out test data, we can observe a pretty decent job by our model having a loss value of `0.046`. The predicted values are very close to the original data.



Discussion

Using the same data set, we tried to use four different machine learning techniques to predict the future prices of the `S&P500`. We used TensorFlow, multi-layer perceptron (MLP), support machine regression (SVR), and long-short-term memory recurrent neural network (LTSM), each with varying degrees of success. While some techniques were able to predict trends, specific prices were not able to be predicted based on past historical data alone. Outliers such socioeconomic events that can have a significant impact on the market, were not available to the algorithm. Although some trends can be induced by using machine learning algorithms, it is to our conclusion these machine learning techniques are not suitable for stock price prediction and that other methods should be explored, such as Bayesian Networks, that can use statistical analysis of other events to help better predict stock prices.

References

Downey, Lucas. 2020. “Efficient Market Hypothesis (Emh).” 2020. https://www.investopedia.com/terms/e/efficientmarkethypothesis.asp.

Kazemi, Hamid. 2020. “TensorFlow.” University of Maryland; University Lecture.

“Keras Api Reference.” 2020. 2020. https://keras.io/api/.

“Keras Lstm Tutorial – How to Easily Build a Powerful Deep Learning Language Model.” 2020. 2020. https://adventuresinmachinelearning.com/keras-lstm-tutorial/.

Phi, Michael. 2018. “Illustrated Guide to Lstm’s and Gru’s: A Step by Step Explanation.” 2018. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

scikit-learn developers. 2020a. “Scikit-Learn Machine Learning in Python.” 2020. https://scikit-learn.org/stable/.

———. 2020b. 2020. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor.

“S&P500.” n.d. https://drive.google.com/file/d/0BxfedcgRnB_BcU1tVVhSeE1iMlk/view.

The Rothschild Archive. 2020. “Nathan Mayer Rothschild and ’Waterloo’.” 2020. https://www.rothschildarchive.org/contact/faqs/nathan_mayer_rothschild_and_waterloo.

The TensorFlow Authors and RStudio, PBC. 2020. “TensorFlow for R from Rstudio.” 2020. https://tensorflow.rstudio.com/tutorials/.

Turchenko, Volodymyr, Patrizia Beraldi, Francesco De Simone, and Lucio Grandinetti. 2011. “Short-Term Stock Price Prediction Using Mlp in Moving Simulation Mode.” In, 666–71. https://doi.org/10.1109/IDAACS.2011.6072853.