Comparing The Effectiveness of Linear Regression and RNN Models in Predicting Tesla Stock Prices Using Tweets

by Zheng Lian (zhengl@stanford.edu) and Brian Zeng (bzeng313@stanford.edu)
Mentor: Benoit Zhou

Abstract

This project seeks to model the relationship between Tesla stock prices and Tesla-related tweets by using linear regression and a recurrent neural network and examines the reliability of these models when used to predict future Tesla stock prices and general Tesla stock trends[1]. Datasets and code for this project can be found at https://github.com/bzeng313/cs221-project.

Introduction

In recent years, Twitter has become not only a site where ordinary people can post short blurbs about their day, but also a place where influential public figures may voice controversial opinions. Consider recent Twitter activity by Elon Musk, the CEO and former chairman of Tesla, for example. On August 7, 2018, Musk tweeted that he had acquired investments to take Tesla private for $420 per share. In fact, however, Musk's claims of secured funding were false and had retracted from taking Tesla private a few weeks after this tweet. As a result, Musk was sued by the SEC for fraud, deposed as chairman, Tesla was fined $20 million, and stock prices for the company fell by 29% between the time of this tweet and late September.

We believe that tweets contain valuable information that help to predict stock market activities, as exemplified by the dramatic drop in share price as a consequence of Musk's fraudulent tweet. Motivated by this observation, we hope to make a predictor that intakes Tesla-related tweets in the last few days and outputs a prediction for the next day's Tesla stock price.

In this paper we explore the effectiveness of two models: linear regression and an recurrent neural network (RNN) with long short-term memory (LSTM) units. We hope that our results, with minor modifications, will also be applicable in estimating prices of other stocks.

Related Work

Our work takes inspiration from two papers. The first, by Pagolu, Reddy, Panda, and Majhi, ran sentiment analysis on Microsoft-related tweets to analyze the relationship between public sentiment over the company and the company's stock price behavior[2]. Specifically, they classified tweets over a three-day period as positive, negative, or neutral and predicted whether or not Microsoft's stock would increase or

---

[1] Our original idea of using solely Elon Musk's tweets proved to be unsuccessful: only a few tweets out of hundreds are related to Tesla, and even then, the tweets of a single person, although the chairman of Tesla at the time, is not an accurate reflection of the entire mood of the Tesla market. We ended up expanding our scope to include all tweets related to Tesla, regardless of who they were written by.

[2] Pagolu, Challa, Panda, and Majhi, "Sentiment Analysis of Twitter Data for Predicting Stock Market Movements", 1.

decrease on the fourth day[3]. They found that there is a strong relationship between public sentiment over a company and that company's stock values, achieving accuracies between 60% and 70% on various models[4].

We attempted some approaches outlined in the above paper, including Word2Vec and N-gram, with little success. While we expected N-gram to not behave too well due to its simplicity, Word2Vec might not meet our expectation due to the lack of labeled data. In Pagolu et al.'s case, they manually labeled close to 4,000 tweets and used those to train their Word2Vec model, an effort we had no time to replicate.

The second paper we drew inspiration from was Mittal and Goel's previous work on stock prediction using tweets. Their goal was to predict the stock market movement by using predicted moods (using filtered tweets) and preprocessed Dow Jones Industrial Average to generate a machine learning model. The resulting model obtained a remarkable 75.56% accuracy.

Many of our approaches were inspired by Mittal et al's work, including how we fill in missing stock prices and the usage of sentiments derived from past related tweets and past stock prices as features in the model. However, their approach is much more sophisticated than ours, including an implementation of their own sentiment analysis model to better analyze the tweet data, and a much better understanding of what neural network suits their need (we simply used the LSTM layer supplied by Keras).

Methodology

Dataset

We will use two datasets in this project.

The first dataset contains the stock price history of Tesla from January 1st, 2018 to November 27th, 2018, downloaded from Yahoo! Finance. Stock entries are organized in columns by date, opening price, highest price, lowest price, closing price, adjacent closing price, and volume.

The second dataset contains 46,000+ tweets (no retweets included) containing the word "Tesla" from January 1st, 2018 to November 27th, 2018. Tweet entries include date and timestamp, content, replies, retweets, and favorites. We were able to obtain these tweets through repeated scrapping from Twitter using Web Scraper, a chrome extension..

Preprocessing

Currently, our stock price data is incomplete, as the stock market is not open on weekends and holidays. To fill the gap, we use the same methods by Pagolu et al. [2] and Mittal et al. [1]. If x is the stock price on one day and y is the next available stock price, the stock price after x is estimated by $\frac{x+y}{2}$. This method is used recursively until all missing stock data have been filled.

[3] Ibid, 1.
[4] Ibid, 3.

- Example: let's say 1/1/18 has the stock price of $200, 1/4/18 has the stock price of $300, and we don't have stock prices for 1/2/18 and 1/3/18 due to them being weekends, our model essentially fills in $250 for 1/2/18 as the average of 1/1/18 and 1/4/18, and and $275 1/3/18 as the average of 1/2/18 and 1/4/18

We also are not interested in the tweets themselves, but rather the sentiments elicited by the tweets and their subjectivities. So we ran sentiment analysis using TextBlob, a python library for processing textual data, on all the tweets at our disposal to first generate 46000+ sentiment scores from -1 to 1 and subjectivity scores from 0 to 1. Each score will then be amplified by the popularity associated with the tweet (the sum of retweets, replies, and favorites for that tweet for each tweet). In the end, the total sentiment score and subjectivity score of tweets from the same day will be calculated before we divide them by the total popularity to generate sentiment and subjectivity feature associated with each day.

- Examples: let's say a tweet from 1/1/18 has a sentiment score of 0.5, subjectivity score of 0.7, 5 replies, 10 retweets, and 15 favorites, we multiply each score by 5 + 10 + 15 to obtain 15 and 21; let's say the total sentiment score for 1/1/18 is 1000 using this method, while the total popularity is 2000, then the sentiment feature for that day is 1000/2000, or 0.5

Feature Extraction

After processing our dataset, we proposed these features to be used in our model:

1. Average tweet sentiment for a day
2. Average tweet subjectivity for a day.
3. Closing stock price for a day.
4. The product of the first and second features.
5. The product of the first and third features.
6. The product of the second and third features

Features 1-6 are then normalized to have mean 0 and standard deviation 1. Our last feature builds a time series from features 1-6.

7. The recordings of the previous 6 features, normalized to have mean 0 and standard deviation 1, in the past M days. So in total, we would have 6*M features.

To determine the relevance of features 1-6 (normalized), we first picked various values of M and computed the correlation of all 6*M features with the next day's stock price from our training set. What we found was that feature 3, for most of the M days, was highly correlated with the next day's stock price, and decreased in correlation as feature 3 went back farther in days. In particular, correlation between feature 3 and the next day's stock price dropped below 0.5 when examining feature 3 from 12 days back. For this reason, we deduced that having M be 11 would be best for our model. We also found that feature 2 had a correlation close to -0.1 and feature 6 had correlation close to 0.1 for the past 2 days. These were high in comparison to features 1, 4, and 5, which had correlations between -0.1 and 0.1 for all 15 days. For these reasons, we decided to keep features 2, 3, and 6 only.

Dataset Splitting and Validation

To test the generality of our models, we allocated 80% of the dataset as the training set and 20% of the dataset to the test set using Sklearn.

To validate our models, we conducted nested cross-validation on the training set with 5 folds. This method of validation trains on the first fold and tests on the second fold, then trains on the first and second fold and tests on the third fold, and so on. We then averaged all errors from the hold out sets to produce a validation error. These validation errors were used to tune hyperparameters in both our models. We chose this method of validation because our data includes time series, so we would like to avoid training on future data when conducting validation.

Concrete Example of Input and Output

Input: Assuming we are predicting the stock price for 1/12/18, our input would be a time series of the past 11 days, where each day contains features 2, 3, and 6. That is,

Feature\ Day	1/1/18	1/2/18	1/3/18	1/4/18	1/5/18	1/6/18	1/7/18	1/8/18	1/9/18	1/10/18	1/11/18
2	-1.16	-2.17	-2.46	-0.19	-0.09	-0.16	-0.10	-1.08	0.89	0.36	-0.70
3	0.19	0.23	0.01	0.11	0.08	0.50	0.65	0.81	0.71	0.75	0.87
6	1.09	-2.08	-2.38	-2.38	-0.06	-0.04	0.06	-0.91	1.10	0.56	-0.51

Table 1. Example of input. Note that the above features have been normalized.

Output: A prediction of Tesla stock price of the day (closing price).
$336.33

Implementation

Linear Regression Model (Baseline)
Our linear regression model treated each value in the time series as input, that is, features 2, 3, and 6 for each of the 11 days (totaling 33 features) were fed into the regression model. We then used stochastic gradient descent to minimize the mean squared loss. We found that having a learning rate of 0.08, 300 iterations, and a batch size of 15 worked well in practice.

Oracle
We have tried many different methods of oracles, including making the predictions ourselves after researching, or looking for online predictors. However, we are not very familiar with stock markets and made quite inaccurate predictions, and resources we found online suggest there are no known accurate predicting softwares, and human predictions are largely inaccurate or highly unspecific (comments on general trend).

RNN[5]

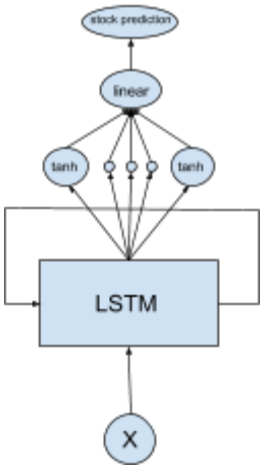

Figure 1. Depiction of our RNN.

The above is a depiction of our RNN with an LSTM layer. X represents our time series input of 11 days. The LSTM block therefore contains 11 memory cells. The last cell outputs N tanh nodes, each of which is fed a single day's features from the time series data. Lastly, each of the N tanh nodes are fed into the linear layer, which outputs a stock prediction. We used stochastic gradient descent to minimize the mean squared loss. We found that having a single LSTM layer with N being 11, a learning rate of 0.02, 250 iterations, and a batch size of 10 worked well in practice.

Evaluation Metrics

We used two evaluation metrics. The first was the average of the root mean squared difference between our model's prediction of the stock percentage change and the actual stock percentage change. This metric is used to determine how close we are in our predictions to the actual next day's stock price.

$$\sqrt{\frac{\sum_{i=1}^{n}(Prediction(x_i) - Actual_i)^2}{n}}$$

Our second metric was the percentage of times our model correctly predicted the stock trend, that is, whether or not our model predicted that the stock price would go up or down correctly. This metric is used to determine how well our model can predict stock trends.

$$\frac{\sum_{i=1}^{n} I[((Prediction(x_i) > Actual_{i-1}) \wedge (Actual_i > Actual_{i-1})) \vee ((Prediction(x_i) < Actual_{i-1}) \wedge (Actual_i < Actual_{i-1}))]}{n}$$

[5] We originally used a CNN, and was advised by our mentor to switch over to a RNN instead

Results

Datas and Graphs

Error\Model	Linear Regression	Recurrent Neural Network
Train	11.32	12.39
Test	17.40	17.20

Table 2. Root mean squared error from both models on the training and testing sets.

Trend Accuracy\Model	Linear Regression	Recurrent Neural Network
Train Accuracy	53.52%	54.30%
Test Accuracy	45.31%	56.25%

Table 3. Trend accuracy from both models on the training and testing sets.

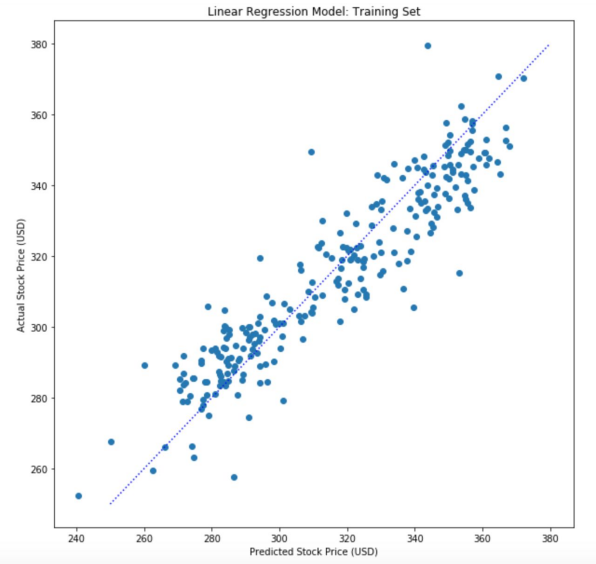

Figure 2. Predicted stock price from linear regression vs actual stock price on the training set.

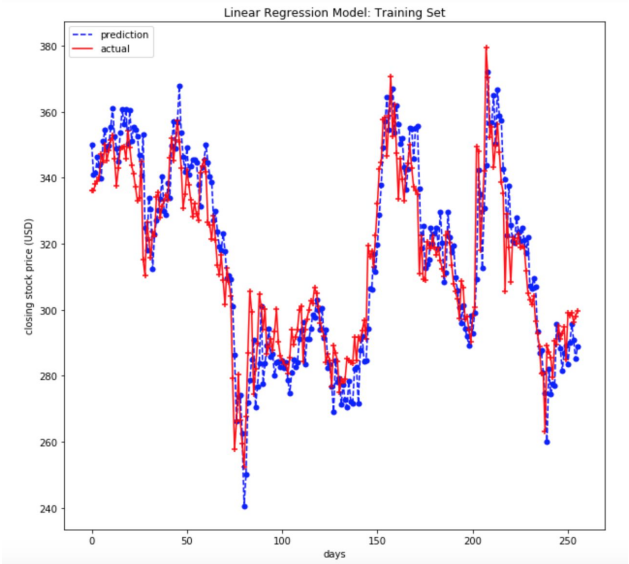

Figure 3. Days vs predicted stock price from linear regression and actual stock price on the training set.

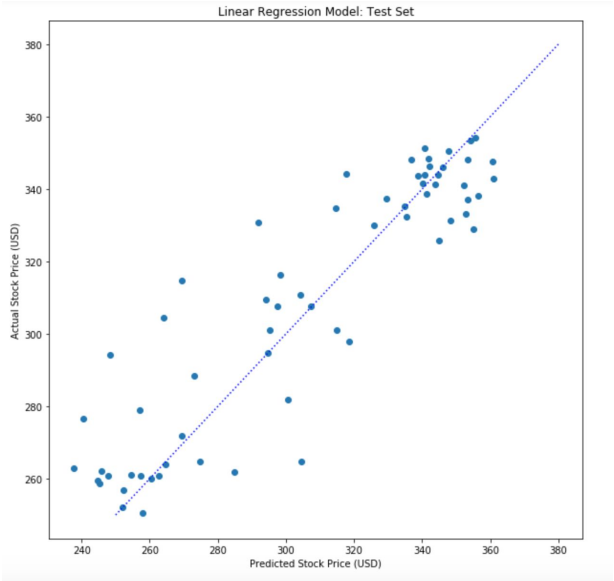

Figure 4. Predicted stock price from linear regression vs actual stock price on the testing set.

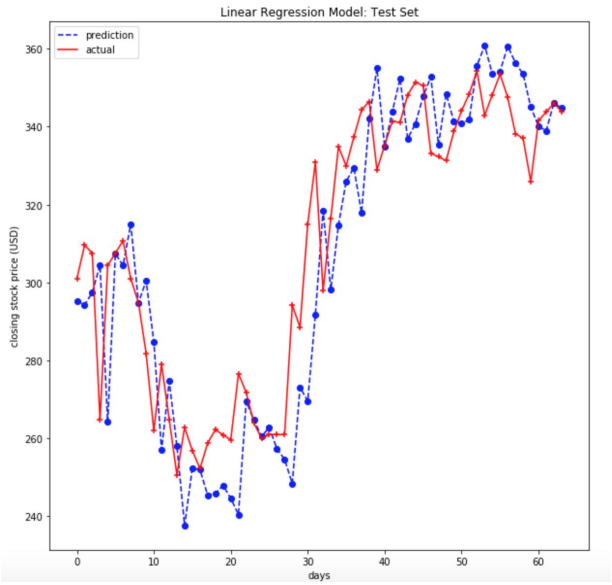

Figure 5. Days vs predicted stock price from linear regression and actual stock price on the testing set.

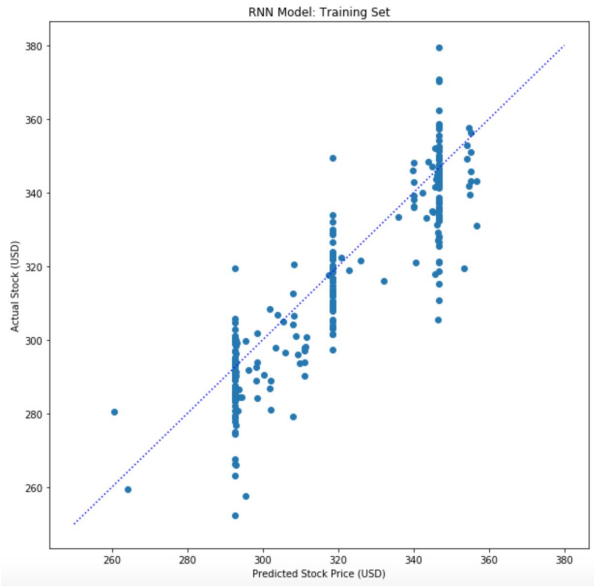

Figure 6. Predicted stock price from RNN vs actual stock price on the training set.

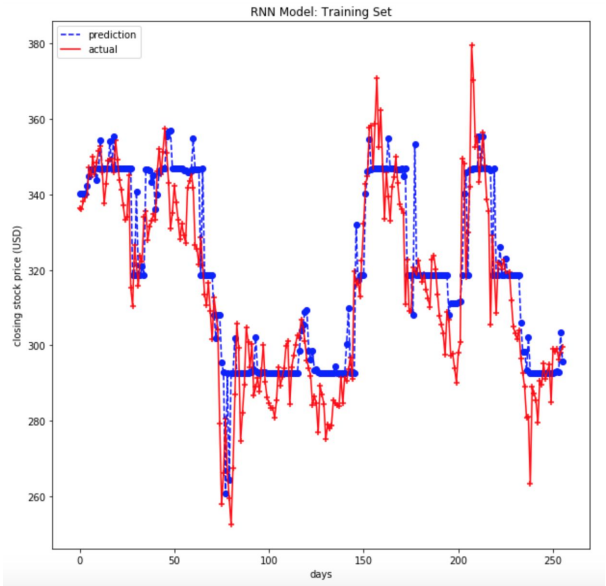

Figure 7. Days vs predicted stock price from RNN and actual stock price on the training set.

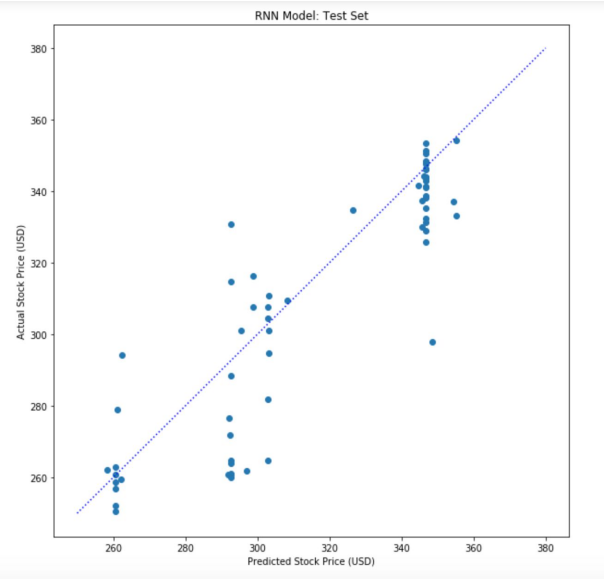

Figure 8. Predicted stock price from RNN vs actual stock price on the testing set.

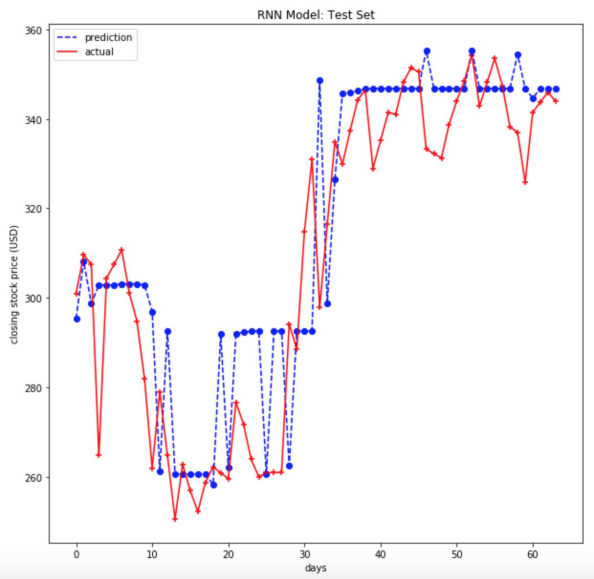

Figure 9. Days vs predicted stock price from RNN and actual stock price on the testing set.

Analysis

From our results we can see that both the linear regression and RNN models did not generalize well because the training errors and test errors for both models varied significantly, as the training and testing errors for the linear regression model were 11.32 and 17.40 respectively, while the training and testing errors for the RNN were 12.39 and 17.20 respectively. In fact, both models failed to accurately predict spikes in the market on the day the spike happened, as shown by figures 3, 5, 7, and 9.

In addition to the loss of generality, we can see that the linear regression model overfit the training data, while the RNN did not. Figure 5 shows the linear regression model predicting prices that are similar to the previous day's stock price. From the training data, we can see that, on average, stock movement from day to day is minimal. This resulted in the linear regression model having a narrow view of the market trend, learning to predict stock prices that were similar to the previous day's stock price only and not taking into account multiple days at a time. In figure 9, we can see that while testing error was high for the RNN, for the most part, the model learned to predict the same stock price for multiple days at a time, and in particular attempted to predict an average of multiple days' stock prices. .

Accuracy-wise, while the accuracy for our linear regression model is, unfortunately, basically equivalent to guessing (53.52% and 45.31% for training and testing set, respectively), our RNN model yielded an above-average accuracy (54.3% and 56.2% for training and testing set, respectively).

Conclusion and Next Steps

Improvement on Features

We believe our results to be inconclusive about whether or not tweet data can help in predicting stock movement, as the tweet information we extracted seemed to correlate very little with the stock price. This

is attributed to the fact that we used Textblob, a pretrained sentiment analyzer, to label the sentiment and subjectivity of our tweets. Textblob was most likely trained on text with sentiment labeled in terms of moods like happy, sad, or neutral. However, the sentiment of a sentence in terms of stock trends must be classified in terms of its effect on the market. For instance, the tweet “Tesla goes bankrupt” implies a neutral sentiment, as this is a non-opinionated statement. However, the sentiment in terms of Tesla’s stock trend should be negative. If more time and resources are available, we would like to label our own data to train a sentiment analysis model that better represents the sentiment and subjectivity elicited in the tweets in terms of stock market movement.

Improvement on Models

We were intrigued that we performed worse on our RNN than CNN even though RNN is the more suited model for this type of problem. A better understanding of different types of neural networks and the parameters they need can potentially answer our confusion and improve the model.

REFERENCES

[1] A. Mittal and A. Goel. Stock Prediction Using Twitter Sentiment Analysis. 2011.
[2] V. S. Pagolu, K. N. Reddy, C. G. Panda, and B. Maji. Sentiment Analysis of Twitter Data for Predicting Stock Market Movements. International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), 2016.
[3] https://www.promptcloud.com/blog/ scrape-twitter-data-using-python-r
[4] https://twitter.com/elonmusk
[5] https://finance.yahoo.com/quote/TSLA/