Stock Market Prediction using Tweet Sentiment Analysis

Rishabh Jaiswal
UFID: 2109-5276
Dept. of CISE, University of Florida
rishabh.jaiswal@ufl.edu

Swarnabha Roy
UFID: 3353-8585
Dept. of CISE, University of Florida
roys@ufl.edu

Abstract

Stock Price Prediction using machine learning helps discover the future value of company stock and other financial assets traded on an exchange. The advent of modern media services and social networking has changed the perception of investing. A bombarding news article or a tweet from a highly influential person may define how a particular stock performs, and it is a fact that investors purchase after public information is released. Although theoretically, an investor cannot benefit or outperform the market by trading on new information, several studies and research show that using a combination of deep learning, temporal data, and news articles can indeed outperform the market.

1 Introduction

Advancements in Artificial Intelligence led to the development of many automated algorithms that help investors in decision-making. But, they rarely used both numerical and textual data in the approach. Market volatility, acquisitions, mergers, global politics, and many other dependent and independent variables may influence the value of a stock in the market. These variables make it extremely difficult for any stock market expert to correctly anticipate the rise and fall of the market with great precision. As a result, more and more investors and organizations are using Machine Learning and robust prediction algorithms for analyzing and forecasting stock prices.

One of the variables highly influencing the stock market are the tweets. One tweet from a highly influential person can result in 3-fold rise or fall of a stock price. This phenomenon has recently attracted many ML enthusiasts to study the effect of tweets on stock prices and further attempt to predict the change due to them.

Akita et al. [1] proposed a novel approach to convert newspaper articles into their distributed representations using Paragraph Vector and model the effects of past events overtime on the opening prices of multiple company stocks with Long Short-Term Memory (LSTM). This paper tackles the drawbacks of existing methods by treating stock prices between companies within the same industry as correlational and not considering data from only one company for training. Moreover, the authors treated stock prices as time-series data and captured the changes using a recurrent network.

Pagolu V.S. et al. [2] collected and curated the tweet data of each company sorted and grouped by dates. They annotated the data manually and brought upon the idea that ”a strong correlation exists between twitter sentiments and the next day stock prices”. The authors also gave an approach to study this correlation effectively by proper data pre-processing steps.

In this project, we have made another attempt to extract some useful information from the tweets and use it to predict the stock prices using the *stockNet* [3] dataset. Features from tweets data is based on the Sentiment Analysis and different regression models - linear, decision tree, SVR, and LSTM - are then applied on the aggregated data to predict the prices.

2 Problem Statement

Our problem statement is as follows:

Given the tweets data sorted w.r.t. to company and date, and the stock data for those companies and dates. We want to predict the future price of stocks of a company taking into consideration, the tweets posted on the previous few days.

The performance of our model will be evaluated using Root Mean Sqaured Error (RMSE).

3 Algorithms

3.1 Linear Regression

Linear Regression (**Ordinary Least Squares**) models the linear relationships between a continuous response and explanatory variables. The relationship between input variables X = $(X_1, X_2, ...X_p)$ and output variable Y takes the form:

$$Y \approx \beta_0 + \beta_1 X_1 + ... + \beta_p X_p + \epsilon$$

where $\beta_0 ... \beta_p$ are the unknown coefficients (parameters) which we are trying to determine. We find $\hat{\beta}$ for $\hat{y} = \hat{\beta_0} + \hat{\beta} X + \epsilon$ by solving the closed-form equation for coefficient vector $\hat{\beta}$ = $(X^T X)^{-1} X^T Y$ which minimizes the SSE.
Evaluation Metrics

- Mean Squared Error (MSE) = $\frac{1}{n}\sum(y_i - \hat{y})^2$
- Sum of Squared Error (SSE) = $\sum(y_i - \hat{y})^2$
- Total Sum of Squares (SST) = $\sum(y_i - \bar{y})^2$
- $R^2 = 1 - \frac{SSE}{SST}$, the proportion of explained y-variability
- Root Mean Squared Error (RMSE) = $\sqrt{\frac{1}{n}\sum(y_i - \hat{y})^2}$

Ideally, we should have **low RMSE** and **high** R^2.
For using Linear Regression we assume that the dataset has linear relationships and independent observations, error tersms have constant variance, errors are uncorrelated and follow normal distributions, and we have low multicollinearity.
However, these models can become overly complex leading to overfitting, to tackle which we need to normalize the data and add regularization ie. adding a penalty λ for large coefficients to the cost function. This can be:

- **Subset** (L_0): $\lambda||\hat{\beta}||_0 = \lambda(number\ of\ non{-}zero\ variables)$
- **LASSO** (L_1): $\lambda||\hat{\beta}||_1 = \lambda\sum|\hat{\beta}|$
- **Ridge** (L_2): $\lambda||\hat{\beta}||_2 = \lambda\sum(\hat{\beta})^2$

We could also combine LASSO and Ridge to get **Elastic Net** regularization.

3.1.1 LASSO Regression

Least Absolute Selection and Shrinkage Operator (LASSO) Regression uses the (L_1)-regularization, and encourages sparsity in the weights ie. weights of the least significant features $\approx$ 0.

3.1.2 Ridge Regression

Ridge Regreassion, also called **Tikhonov Regularization** uses the (L_2)-regularization, and encourages minimization of the weights ie. weights will be close to 0.

3.2 Stochastic Gradient Descent

Gradient Descent is a generic iterative procedure to update the parameters based on the gradient, used to solve an optimization problem. It minimizes the average loss by moving iteratively in the

direction of steepest descent, controlled by the learning rate (size of steps we take towards the minimum). Given parameters/weights(θ), a learning rate (γ), we compute $\Delta_\theta L(\theta)$ which is the gradient of the loss function $L(\theta)$ with respect to the parameters. Now we take a step in the opposite direction which minimizes the loss. γ can be updated adaptively for better performance.
This can be written as an algorithm:

1. Initialize weights θ randomly with near-zero values
2. Loop until convergence:
    - Calculate the average network loss $L(\theta)$
    - **Backpropagation** - iterate backwards from the last layer, computing the gradient $\frac{\partial L(\theta)}{\partial \theta}$ and updating the weight $\theta \leftarrow \theta - \gamma \frac{\partial L(\theta)}{\partial \theta}$
3. Return the minimum loss weight matrix θ

To prevent overfitting, we can apply regularization by:

- Stopping training when validation performance drops
- Dropout - randomly drop some nodes during training to prevent over-reliance on a single node
- Embedding weight penalties into the objective function
- Batch Normalization - stabilizes learning by normalizing inputs to a layer

Now, **Stochastic Gradient Descent** only uses a single point to compute gradients instead of taking a step after sampling the *entire* training set. This leads to smoother convergence and faster compute speeds.

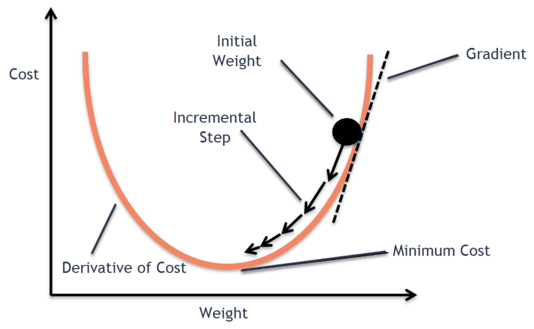


Figure 1: Illustration of Gradient Descent

3.3 Decision Trees

A decision tree is an acyclic graph used to make predictions. It is a non-parametric model suitable for classification or regression problems. Each node in the tree contains a simple feature comparison against some field. The result of each comparison is either true or false, which determines if we should proceed to the left child or right child of the present node. Decision Trees are also known as **Classification and Regression Trees (CART)**.
CART for regression minimizes SSE by splitting data into sub-regions and predicting the average value at leaf nodes. The complexity parameter c keeps only those splits that reduce loss by at least c.
CART for classification minimizes the sum of region impurity, where $\hat{p_i}$ is the probability of a sample being in category i. Possible measures, each with a max impurity of 0.5.

- Gini Impurity = $1 - \sum(\hat{p_i})^2$
- Cross Entropy = $-\sum(\hat{p_i})log_2(\hat{p_i})$

At each leaf node, CART predicts the most frequent category, assuming false negative and false positive costs are the same. The splitting process handles multicollinearity and outliers.
Since, decision trees make almost no assumptions about the data, they have low bias and high

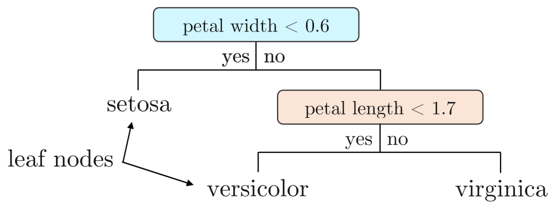


Figure 2: A sample decision tree on the IRIS dataset

variance, making them prone to overfitting if the model is too complex. A few ways to control the complexity are: **set maximum depth** to prevent tree from growing too deep, **set a minimum number of examples to split** to restrict splitting, and **pruning** to remove branches that do not reduce error significantly.
In practice though, we rarely use just one decision tree. Instead, we aggregate many decision trees using methods like ensembling, bagging, and boosting.

3.4 Support Vector Regression

Support Vector Regression (SVR) allows us to set how much error is acceptable in our model and finds a best-fit line or hyperplane in higher dimensions to fit the data.
The objective function for SVR is to minimize the coefficients, or, the L_2-norm of the coefficient vector. The error term is handled in the constraints by setting the absolute error less than or equal to a threshold, called the maximum error, ϵ. We can tweak epsilon to gain the desired accuracy of our model.
This can be formulated as:

$$\textbf{Minimize } \frac{1}{2}||w||^2$$
$$\textbf{Constraints } |y_i - w_i x_i| \leq \epsilon$$

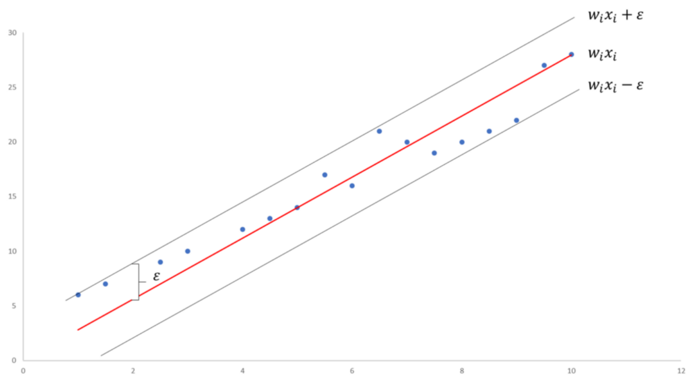


Figure 3: Support Vector Regression

3.5 LSTM

To understand LSTMs, we first need to understand **Recurrent Neural Networks** or RNNs. RNNs predict sequential data using a temporally connected system that captures both new inputs and previous outputs using hidden states
RNNs can model various input-output scenarios, such as many-to-one, one-to-many, and many-to-many. They rely on parameter (weight) sharing for efficiency. To avoid redundant calculations during backpropagation, downstream gradients are found by chaining previous gradients. However, repeatedly multiplying values greater than or less than 1 can lead to **Exploding gradients** where there is model instability and overflows or **Vanishing gradients** where we have loss of learning ability. We can mitigate this using:

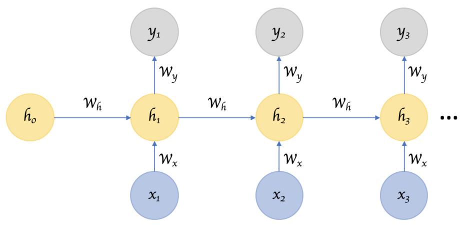


Figure 4: A sample RNN structure

- Gradient clipping - cap the maximum value of gradients
- ReLU - its derivative prevents gradient shrinkage for $x > 0$
- Gated cells - regulate the flow of information

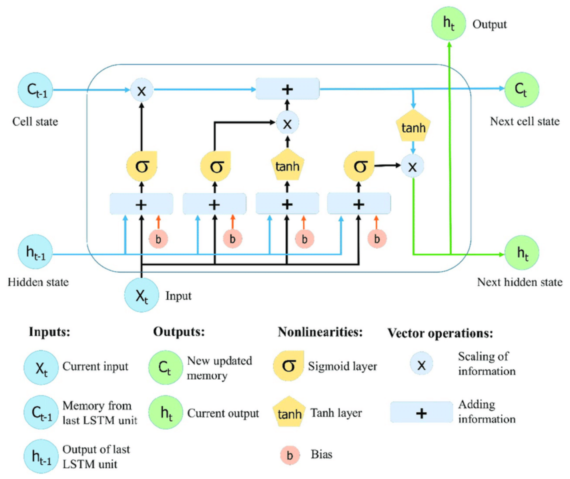
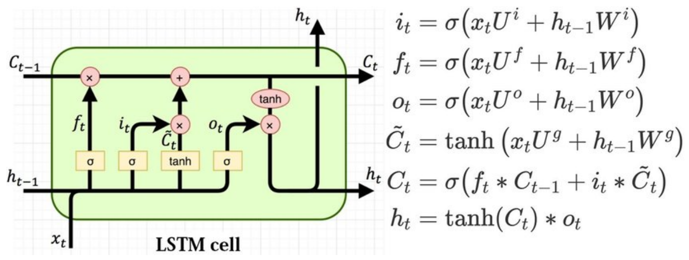


Figure 5: A sample LSTM structure

Long Short-Term Memory or LSTM models learn long-term dependencies using gated cells and maintain a separate cell state from what is outputted.
Gates in LSTM perform the following functions:



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Figure 6: Detailed description of LSTM

1. Forget and filter out irrelevant info from previous layers
2. Store relevant info from current input
3. Update the current cell state
4. Output the hidden state, a filtered version of the cell state

We can stack LSTMs to improve performance. One widely used architecture is using CNN-LSTMs, where the data is first fed into an Convolutional Neural Network (CNN), and its output is passed as the input to a stacked LSTM to generate predictions.

3.6 VADER

Hutto and Gilbert [4] introduced VADER as a parsimonious rule-based model for sentiment analysis of social media text. VADER **(Valence Aware Dictionary for Sentiment Reasoning)** from the NLTK package in Python is a model used for analyzing text sentiment. It is sensitive to both polarity, that is, if the text shows a positive, negative or neutral sentiment, and intensity or strength of the emotion. VADER uses a dictionary that maps lexical features to emotion intensities called sentiment scores which is the sum of the intensities of each word in the text, and returns a sentiment score in the range -1 to 1, from most negative to most positive.
Five heuristics are used namely: **punctuation**, **capitalization**, **degree modifiers**, **shift in polarity due to "but"**, and **examining the tri-gram before a sentiment-laden lexical feature to catch polarity negation**.

4 Experiments

We start by fetching the related information from tweet data and perform sentiment analysis on it. We then use the sentiment analysis to generate a parameter which not only represents the gist of the data instantly, but also be useful in price prediction. Our experiment consists of three stages -

- Pre-processing of Tweet data and Stock data
- Sentiment analysis
- Merging of tweet and stock data
- Applying ML algorithms to predict future stock prices

4.1 Data pre-processing

The **StockNet** [3] dataset contains 119844 rows of tweet data aggregated company wise and date wise. Each row contains number of fields such as date of post, time of post, repost, followers count, tweet text, no. of shares, etc. We extracted *company*, *date*, *follower count*, and *tweet* from the raw data. For each date, there are a minimum of 4-5 tweets per company present in the dataset. The tweets were then cleaned using basic text processing methods - punctuation removal, removal of stop words, changing to lower case, and removing mentions and hashtags. This was necessary to so that sentiment analysis can be done in the later steps.

Similarly, StockNet dataset contains 108592 rows for stock price data. Each row represents a full day statistic for the stock price in the form of opening price, closing price, highest figure, lowest figure, adjusted closing price, volume, company name, and date. This data is processed to include 30 timesteps from the past. Our aim is to predict the closing price for a stock with the help of these timesteps and features from tweet data.

4.2 Sentiment analysis

To extract some meaningful feature from the tweets data, we used the *VADER Sentiment Analyser* which is especially designed for social media text. We obtained the *compound sentiment polarity* from each tweet using this analyser which tells the extent of the sentiment. A positive compound polarity means the sentence is positive, a negative compound polarity means the sentence is negative, and a zero polarity means the sentence is neutral. Then, we calculate our parameter called **Mean Follower-Sentiment Product** which is calculated as below:

$MeanFollower - SentimentProduct(MFSP)$ = SUM($Followercount * Sentimentpolarity$) / $Totalno.oftweetsforthatcompanyonthatday$

This parameter effectively tells about the impact of certain tweets made about a company on a certain day. If follower count of a person is low, his tweet will not influence the public as much as the person with high follower count. Furthermore, few no. of influential people will have more effect on public opinion than more no. of less influential people.

Linear Regression	Decision Tree	Support Vector	Long Short Term Memory (LSTM)
3.71803	44.1030	26920.0066	7893.7368

Table 1: Performance Evaluation (in terms of RMSE)
of Different ML Models for Stock Price Prediction

OLS	Ridge	Lasso	SGD
3.71803	5.70719	7.8157	9.9530

Table 2: Performance Evaluation (in terms of RMSE)
of Different Linear Models for Stock Price Prediction

4.3 Data aggregation

After the pre-processing, we find that the no. of rows in tweets data is less than those in stock data. In simple words, we do not have tweet data for every date for which we have the stock price data. We kept the MFSP for such rows to be zero. The two datasets were merged with [Comapany, Date] as the key, and the NaN values (for MFSP) were replaced with zero. Also, 30 timesteps for MFSP were also added to each row to incorporate the effect of old tweets in the prediction of today's price.

4.4 Stock price prediction

The steps involved in prediction process are as follows:

1. Scale the data using MinMax Scaler.
2. Split into train and test datasets.
3. Train the ML model on the train dataset.
4. Predict price for test data.
5. Evaluate the model using root mean sqaured error (RMSE).

The models used in the experiment are - Linear Regression, Decision Tree Regression, Support Vector Regression, Long Short Term Memory (LSTM) model.

The Linear regression model proved to be the best among all with $RMSE = 3.7180$.

4.5 Statistics

The performance of models decreased in the order Linear regression $>$ Decision tree regression $>$ SVR $>$ LSTM. The RMSE scores of the experiments are described in the table.

Given the efficiency of the linear models, we further carried experiments with different linear models such as Ordinary Least Squares, Ridge, Stochastic Gradient Decent, and Lasso Regressions. The RMSE scores for these models are also described here.

5 Conclusion

As we can see, Linear Model with OLS is performing the best out of all the models. For the given company, it the RMSE of 3.4427 and it's overall RMSE is 3.7180. Since the tweet data is not a time-series data, incorporating it in the prediction of stock prices has its own challenges. The models that usually perform well with time-series data (such as LSTM) are outperformed by linear models as linear models are able to take the sentiment parameter into the factor more precisely than other models. The sample predictions for a specific company are shown.

References

[1] Ryo Akita, Akira Yoshihara, Takashi Matsubara, and Kuniaki Uehara. Deep learning for stock prediction using numerical and textual information. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–6, 2016.

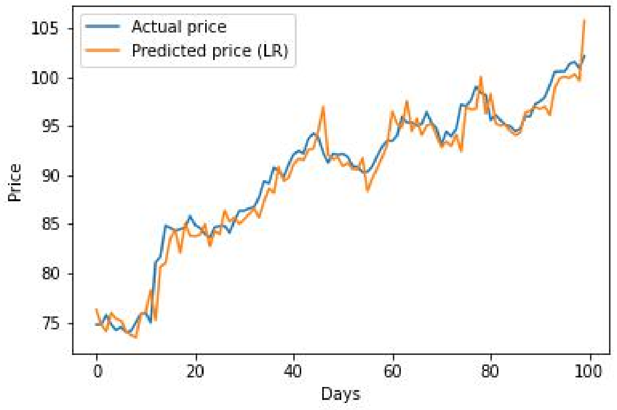


Figure 7: Prediction using OLS Linear Regression Model

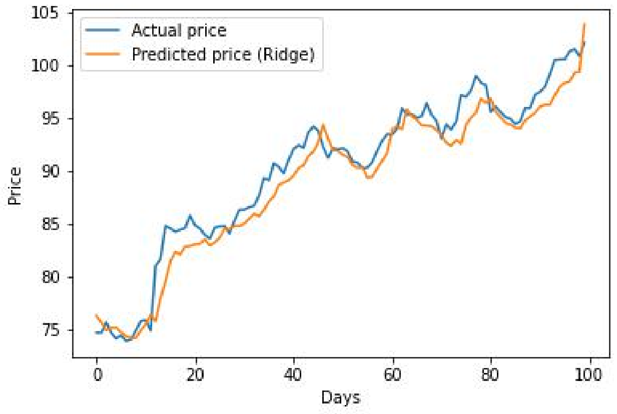


Figure 8: Prediction using Ridge Regression Model

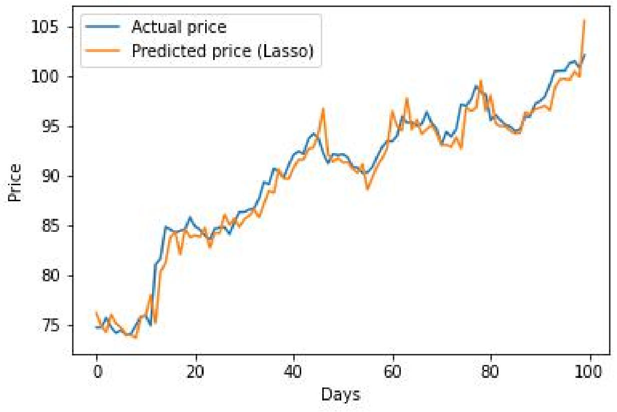


Figure 9: Prediction using Lasso Regression Model

[2] Venkata Sasank Pagolu, Kamal Nayan Reddy, Ganapati Panda, and Babita Majhi. Sentiment analysis of twitter data for predicting stock market movements. In *2016 international conference*

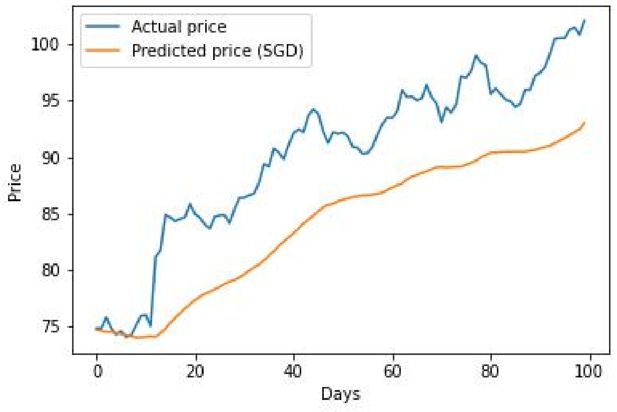


Figure 10: Prediction using SGD Regression Model

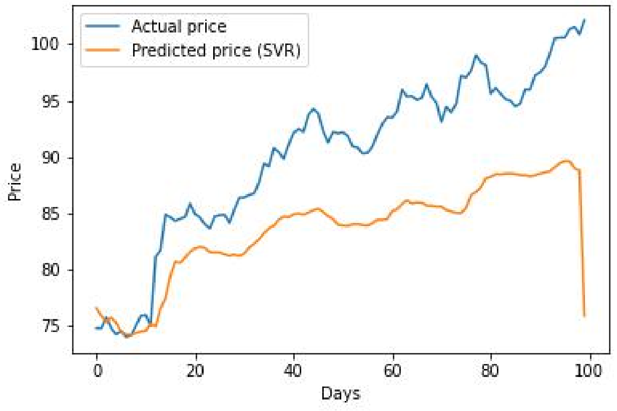


Figure 11: Prediction using SVR Model

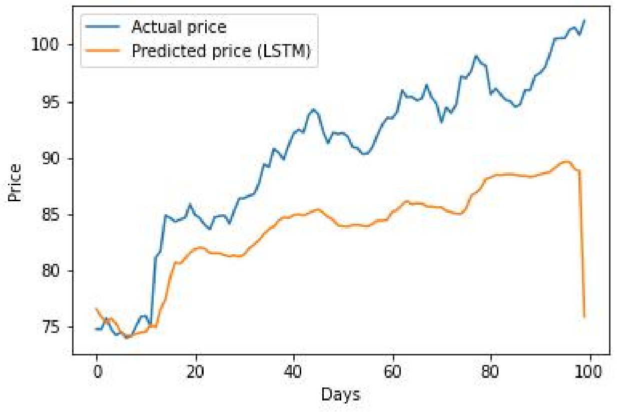


Figure 12: Prediction using LSTM Model

on signal processing, communication, power and embedded system (SCOPES), pages 1345–1350. IEEE, 2016.

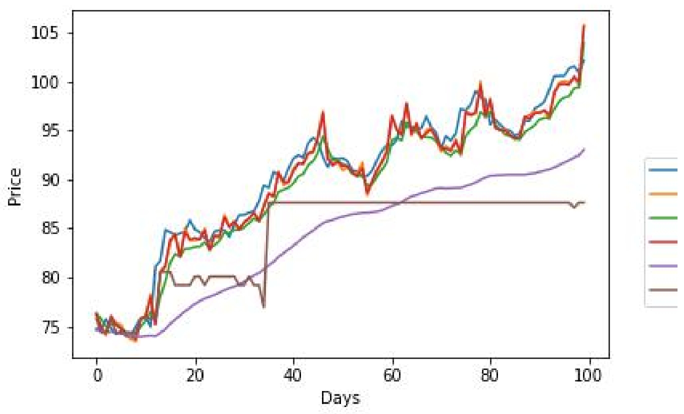


Figure 13: Comparison of Prediction from Different Linear Models

[3] Yumo Xu. Stocknet dataset github. 2018.

[4] C.J. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. 01 2015.