Stock Market Predictions using HMM and LSTM-RNN

Malhar Mahant[2], Roydon Pereira[2]
Northeastern University, Boston, MA
[1] mahant.ma@northeastern.edu
[2] pereira.ro@northeastern.edu

Abstract

This paper discusses and compares different techniques of stock value predictions. Prediction of the stock market remains a difficult problem to solve mainly due to factors out of the historical data of the stock. In recent years many machine learning models have been proposed to make accurate and efficient stock market predictions. The objective of this paper is to evaluate two such models. The two models used for prediction are Hidden Markov Model and Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN). We try various features, window sizes, and configurations for the two models to improve the models individually. And then compare the performance of the two models against each other. The metric used for analyzing is Root Mean Squared Error using the actual values of the stock and the predicted values

Introduction

Forecasting of the stock market is a problem of great commercial interest and has huge financial implications. Accuracy in the prediction of values of the stock market is necessary to maximize profitability and avoid poor investment choices.

Financial institutions like banks and asset management companies use various tools based on traditional methods of predicting time series data. Methods such as auto-regressive integrated moving average (ARIMA) and generalized auto-regressive conditional heteroskedasticity (GARCH) models. These models work great for linear time series data, however, the stock market can be highly non-linear as stock prices can vary wildly due to external influences. Therefore, the stock market can be considered a non-stationary time series and it is another reason why traditional statistical methods perform poorly in predicting the values of the stock [3].

On the other hand, Machine Learning has been successful in various other fields like predictive text and classification problems. Machine Learning as a discipline is getting attention as a possible alternative for the problem of stock market prediction spurring several studies in the field. In several of the aforementioned studies, different machine learning models are compared to traditional statistical methods. In most studies, it is observed that the machine learning models perform marginally or greatly better than their traditional counterparts. However, this paper focuses on comparing two of these machine learning models with each other.

Our objective is to compare an implementation of a Hidden Markov Model (HMM) and a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN). The data that is used is daily data on the Google stock. Data from Dec 2014-17 is used as the training data and data from Dec 17-21 is the test dataset. Our metrics to analyze the performance will Rooted mean squared error. The two models will be implemented and tested out with different configurations and features and then we will compare the results of the two based on our chosen metric.

Background

This section provides some background of the methods used in this project.
Forecasting of the stock market is a problem of identifying the trends in data and variables in the historical data that result in the future outcome. Neural networks have a similar notion where the importance of a variable is defined by weights, which is essentially the identification of variables that impact the output the most. To explore this idea, we look at Neural Networks

Neural Networks

Neural Networks is a system inspired by the human brain, which consists of a set of algorithms that loosely mimic the human brain.
A neural network consists of three layers,

- the input layer,

- Single or multiple hidden layers
- and an output layer.

There could be multiple nodes or neurons in each layer. These layers and each node are connected to one another and have an associated weight and a threshold value. These weights and thresholds decide whether the output of a layer will be passed onto the next layer. In the hidden layer, the weighted sum of inputs is passed to an activation function to generate output values. The output layer generates an output from the predicted values from hidden layers choosing the values with minimum prediction error, which is the difference between the actual value and the predicted value calculated by a SoftMax function.
The mathematical formula can be given by:

$i_n = \sum^m w_n i.xi$

$o_n = f(i_n + b_n)$

where x1, . . ., xm are the input parameters; w_n1, . . ., w_nm are the weights for the node n; in is the weighted sum of inputs; b_n is the bias; f is the activation function; and o_n is the output of the neuron.
In the learning stage, these weights are adjusted by the neural network by comparing the predicted values with the actual values, the error in prediction is calculated. These prediction error values are backpropagated to the hidden layers and the weights for the neurons are adjusted accordingly. This process from the input layer to the output layer and then back-propagation of error and adjustment of weights is called one "epoch". When the prediction error is minimized, we consider the neural network to be trained.

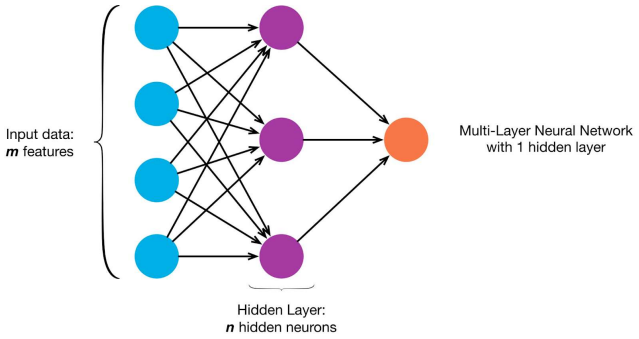

Figure 1: Structure of a Multi-Layer Neural Network [Kang, N 2017]

Recurrent Neural Networks (RNN)
In contrast to feed-forward Neural Networks, recurrent neural networks use the output of the previous timestep as part of the input to the next timestep. The observations from a previous time step are used as a "context" while predicting future values.

In mathematical terms it is shown by

$h_t = f(w_{hx}.x_t + w_{hh}.h_{t-1} + b_h)$

$y_t = g(w_{yh}..h_{t-1} + b_h)$

where x_t is the input to the network at time step t. For example, x1 could be a vector of values for stock at time t.
h_t represents a hidden state at time t and acts as the "memory" of the network. h_t is calculated based on the current input and the previous time step's hidden state: h_{t-1}. The function f is a transformation function such as tanh.
The RNN has input to hidden connections parameterized by a weight W_{hx}, hidden-to-hidden recurrent connections parameterized by a weight W_{hh}, and hidden-to-output connections parameterized by a weight W_{yh} and all these weights are shared across time.y_t is the output of the network at each time step.
The hidden layer acts as a memory for storing the information of previous timesteps. This information is used to predict future outcomes. However, the problem with RNN is that it becomes difficult to capture long-term dependencies in the time series due to a problem called vanishing gradient. In RNNs backpropagation occurs at every time step this is called backpropagation through time. During backpropagation, each weight of the neuron of the network is updated in proportion to the partial derivative of the prediction error with respect to the current weight for each iteration of training. In some cases, the gradient will become very small, effectively preventing the weight from changing its value. This restricts the training of the network as weights would never be updated significantly

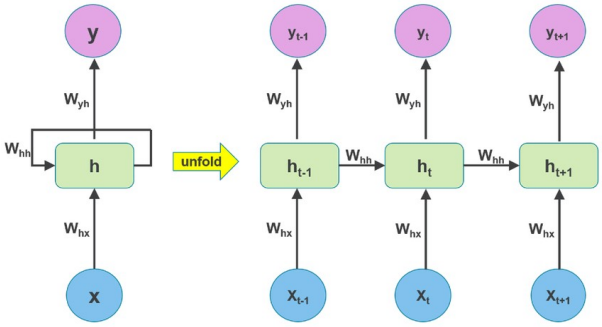

Figure 2: Structure of a Recurrent neural network [Venugopal Prakash, Vigneshwaran T (2019)]

Long Short-Term Memory (LSTM)
The Long Short-Term Memory (LSTM) is a special kind of RNN. The LSTM architecture overcomes the problem of vanishing gradient by maintaining cell states. These cells are maintained throughout the layers of the network in a sort of a continuous "conveyor belt" across the layers. These cells make LSTM capable of learning long-term dependencies in the data.

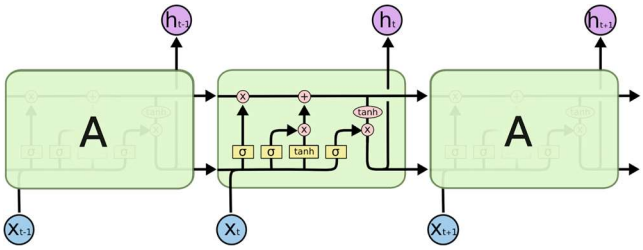

Figure 3: Internal structure of LSTM network (Colah's Blog 2015)

LSTM has a three-step process. First is the forget gate
It can be shown mathematically by the following function.

$ft = \sigma(Wf.[ht-1,xt] + bf)$

σ is a sigmoid function that takes in the new input xt and the information from the previous time step ht-1.
The output of this sigmoid function is in the range of 0 to 1 where a value near 0 means these values can be forgotten whereas a value 1 signals these values to be retained.

The second step is the update layer, we have the update gate which selectively updates cell state values

$i_t = \sigma(Wi.[ht-1,xt] + bi)$

i_t is the output of a sigmoid function that takes in the new input xt and the information from the previous timestep ht-1. The output is used to decide whether to update the values of the cell states or not.

$Ćt = \tanh(Wc .[ht-1,xt] + bc)$

Ćt is a candidate value that is calculated using a tanh function that takes in the new input xt at the current timestep and the information from time step t-1.

$Ct = ft. ct-1 + i_t. Ćt$

Finally, using the value of the forget gate ft and input gate i_t we decide whether or not to update the cell state with the old value Ct-1 and new candidate value Ćt. Value of 0 for ft or it means we drop the value that is being multiplied with it
The final part is the output gate.

$ot = \sigma(Wi.[ht-1,xt] + bo)$

ot is the output value of the sigmoid function, with values between 0 and 1.

$ht = ot. \tanh(ct)$

The value of ot is used to decide the final output ht which is obtained by multiplying ot with the output of running a tanh function the cell state ct

Thus the use of sigmoid functions in the LSTM allows the network to pass useful data through to the next layer while discarding irrelevant data.

Similar to the Neural Networks explained above, the markov process can also be used for analyzing and predicting randomly changing value systems. We explain more about this below.

Markov model

A markov model is a Stochastic method for randomly changing systems where it can be assumed that future states do not depend on the past states. Markov models are engineered to handle data which can be represented as sequence of observations over time. The markov models can represent all the possible outcomes from a state as well as the transitions and the probabilities of these transitions. This helps the model to recognize patterns, make predictions and to learn the statistics of sequential data.

Consider a set where Xt contains some state variables. We assume that the state sequence starts at time t=0. For such a scenario, the transition model is represented by the probability:

$P(Xt|X0:t-1)$

However, since the state X0:t-1 is unbounded in size as t increases, we can apply the markov chain rule here to represent the transition model such that the current state depends on only a finite fixed number of previous states. Hence, the same transition probability can be represented as:

$P(Xt|X0:t-1) = P(Xt|Xt-1)$

Thus, if we are given a state transition matrix for all the required states, the probability of any sequence of states can be calculated by applying Markov chains.

Hidden Markov Models (HMM)

A HMM is a special case used when we cannot observe the states themselves but only the result of some probability function(observation) of the states. HMM is a statistical Markov model in which the system being modeled is assumed to be a Markov process with hidden states.
An HMM is characterized by the following

- number of states in the model (N)
- number of observation variables (where O represents and observation sequence)
- state transition probabilities (represented by transition matrix, A)
- initial state distribution
- observation emission probability distribution that characterizes each state (B)
- prior probability (π),

Hence, we represent the HMM as, $\lambda = (A, B, \pi)$

To work with HMM, the following three questions need to be resolved -

1. How do we compute the probability, P(O| λ) of an observed sequence O = O1, O2, …..., Ot given the model λ= (A, B, π) ?

2. How do we choose the most likely state sequence to generate the observed sequence O and the model λ ?

3. How do we learn the values for the HMMs parameters A and B given the observation sequence O and a space of models found by varying the model parameters A, B and π?

In order to solve the above questions, there exists some well-established algorithms. The most common ones used are - forward-backward algorithm to compute the P(O| λ), Viterbi algorithm to resolve question 2, and an Expectation-Maximization algorithm like the Baum-Welch algorithm to find the values of A and B in the HMM. The GaussianHMM from the hmmlearn package that we use in our implementation resolves all the above questions using these algorithms.

Related Work

Stock market forecasting has received great interest from the scientific community. As such there have been several studies on effective and accurate forecasting of the stock market. Traditional techniques for time series prediction such as ARIMA and GARCH models fall short in predicting the stock market.(Sima Siami-Namini, Akbar Siami Namin 2018).

We also see methodologies that integrate traditional models with neural networks. Guresen et.al (2011) which analysed are multi-layer perceptron (MLP), dynamic artificial neural network (DAN2) and hybrid neural networks which use generalized autoregressive conditional heteroscedasticity (GARCH) in their study. The study came to conclusions that were contrasting to findings of Roh (2007). So these hybrid models are found to be unsatisfactory and inconclusive in their findings.

Project Description

The goal of our project is to compare the performance of a Hidden Markov Model (HMM) and a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) for prediction of "Close" values for a specific stock. The problem is approached in following steps:

- Implement each model individually
- How does changing the window size for Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) affect the performance?
- How does changing the features that we select for training affect the performance?
- How does changing the number of epochs for training the LSTM-RNN affect the performance?
- Compare the performance of the HMM and the LSTM-RNN. Which model gives more accurate predictions?

Dataset

The dataset that used for the experiments are financial time series data acquired from Yahoo Finance. The data is daily stock data on two stocks on NASDAQ, namely, GOOGL and TSLA. The daily data for a stock consists of the Open, High, Low, Close, Adj Close, Volume for the day. Open is the the opening price of the stock for the day.

Close is the last traded value of the stock for the day. High is the highest traded value of the stock for the day. Low is the lowest traded value of the stock for the day. Adj Close it the adjusted close value where amends are made to the close price based on corporate actions taken during the trading session. Volume is the volume of the stock traded throughout that day.

The GOOGL data for the period 12/05/2014 to 12/04/2017 and TSLA data for the period 12/12/2014 to 12/11/2017 are used as the training data. GOOGL timeseries for the period 12/06/2018 to 12/03/2021 and TSLA data for the period 12/13/2017 to 12/10/2021 is used as the test data.

Long Short-Term Memory Recurrent Neural Network (LSTM-RNN)

Financial time series can be non-linear in nature and sometimes the data can be seemingly random. That is the timeseries are non-stationary. Recurrent Neural Networks are capable of handling such non-stationary data, and effectively identify relationships in data to predict future data. RNNs have the ability to store certain information about the data that allows the RNN to analyze relationships between the variables of the stock data. However, the problem with RNN is that as the layers increase the gradient to update the weight becomes smaller and the updates to the weight become negligible, thus stopping the network from learning. LSTM architecture proposed in 1997 solves this problem of vanishing gradient (Sepp Hochreiter, Jürgen Schmidhuber 1997). LSTM are capable of learning long term dependencies in the data. Two variations of the LSTM algorithm are implemented.

- First implementation is a multi-step univariate algorithm (Sima Siami-Namini, Akbar Siami Namin 2018).
- Second, implementation builds on the first algorithm to develop a new multi-variate algorithm.

The Keras library is used to implement the algorithms and the experiments were carried out on the Google Colab platform. For both of the algorithms a function export_report has been defined that takes in the actual and the predicted values and calculates the RMSE for the given dataset

Reporting procedure that is used at the end of both algorithms.

```
1.  # report performance
2.  Procedure export_report(actual,pre-
    dicted,filename):
3.    rmse = sqrt(mean_squared_error(actual,
    predicted))
4.    results = DataFrame()
5.    results.insert(0,"Actual values",actual)
6.    results.insert(1,"Predicted values",pre-
    dicted)
7.    results.insert(2,"Difference",actual-pre-
    dicted)
8.    results.insert(3,"RMSE",rmse)
9.    r2score = r2_score(actual,predicted)
10.   results.insert(4,"r2 score",r2score)
11.   results.to_csv(filename+'.csv')
```

The univariate algorithm

```
1.  Inputs: timeseries
2.  Output: predicted series, RMSE of predicted
    series
3.  #load dataset
4.  data ← pandas.read_csv(inputfile)
5.  #select Close column
6.  trainData ← data[“Close”]
7.  #Scale data to a feature range of 0 to 1
8.  trainData ← scaler.fit_transform(trainData)
9.
10. #split into multiple timesteps
11. for i in range
    (timesteps,length(trainData)):
12.     X_train.append(trainData[i-
    timesteps:i,0])
13.     y_train.append(trainData[i,0])
14.
15. #Reshape data to be used as input for LSTM
    in the form [samples, window size, fea-
    tures]
16. X_train = np.re-
    shape(X_train,(X_train.shape[0],X_train.sha
    pe[1],1))
17.
18. #Create 4 layer using keras library
19. #repeating the following line 4 times
20. model.add(LSTM(units=100, return_sequences
    = True, input_shape =(X_train.shape[1],1)))
21. #compile the model
22. model.compile(loss='mean_squared_error',
    optimizer='adam')
23. #train model
24. model.fit(X_train,y_train, epochs)
25.
26. #Prepare the test data in same way as the
    training data, then use the test data for
    prediction
27. y_pred = model.predict(X_test)
28. #The output is feature scaled to 0 to 1, so
    we inverse the transformation to get actual
    predicted values
29. predicted_price = sc.inverse_trans-
    form(y_pred)
30. #Call procedure export_report to calculate
    RMSE
31. export_report(actual,predicted,outputfile-
    name)
```

The univariate algorithm makes use of “Close” variable of the dataset as the only feature to be used in the LSTM model. The data is read from a CSV file and a series of the “Close” variable is extracted from the training dataset. The data is prepared by scaling the values to a feature scale of 0 to 1 using normalization and standardization by the MinMaxScaler function in Scikit and splitting the input into multiple timestep according to the given timestep value. Then the model is built using the Keras library as shown on line 20, the model takes in the number of neurons and the dimensionality of the input. The model makes use of mean square error as the loss function and ADAM as the optimizer algorithm to compile (Diederik P. Kingma, Jimmy Ba 2015). Then the model is trained by calling the fit function that takes in the training dataset and the number of epochs to be trained.

The multivariate algorithm

```
1.  Inputs: timeseries
2.  Output: predicted series, RMSE of predicted
    series
3.  #load dataset
4.  data ← pandas.read_csv(inputfile)
5.  #drop the Adj Close column
6.  data  ← data.drop(“Adj Close”)
7.  trainData ← data
8.  #Scale data to a feature range of 0 to 1
9.  trainData ← scaler.fit_transform(trainData)
10.
11. #split into multiple timesteps
12. for i in range
    (timesteps,length(trainData)):
13.     X_train.append(trainData[i-
    timesteps:i,0])
14.     y_train.append(trainData[i,0])
15.
16. #Create 4 layer using keras library
17. #repeating the following line 4 times
18. model.add(LSTM(units=100, return_sequences
    = True, input_shape
    =(X_train.shape[1],X_train.shape[2])))
19. #compile the model
20. model.compile(loss='mean_squared_error',
    optimizer='adam')
21. #train model
22. model.fit(X_train,y_train, epochs)
23.
24. #Prepare the test data in same way as the
    training data, then use the test data for
    prediction
25. y_pred = model.predict(X_test)
```

```
#The output is feature scaled to 0 to 1, so we inverse the transformation to get actual predicted values
predicted_price = sc.inverse_transform(y_pred)
#Select the predicted "Close" column
predicted  ← predicted_price["Close"]
#Call procedure export_report to calculate RMSE
export_report(actual,predicted,outputfilename)
```

The multivariate algorithm makes use of 5 features that are available. It takes in the Open, Close, High, Low and Volume variables of the dataset as the features to be used in the LSTM model. The data is read from a CSV file and the "Adj Close" variable is eliminated from the training dataset. The data is then preprocessed by scaling the values to a feature scale of 0 to 1 using normalization and standardization by the MinMaxScaler function in Scikit and splitting the input into multiple timestep according to the given timestep value. The model is built using the Keras library as shown on line 20, the model takes in the number of neurons and the shape of input. Here the input shape is samples, window size, no. of features. The model makes use of mean square error as the loss function and ADAM as the optimizer algorithm to compile (Diederik P. Kingma, Jimmy Ba 2015). Then the model is trained by calling the fit function that takes in the training dataset and the number of epochs to be trained.

In both the above algorithms certain parts from the pseudocode such as the Dense layer, Dropout Layer in model and other details of the implementation have been eliminated for the sake of simplicity. However, these eliminated details are important to the functioning of the algorithms.

Hidden Markov Model

The basic idea for using HMM for Stock Market Prediction calls for finding patterns from the previous datasets that match with current stock price behavior. Once these patterns are found, we compare these two datasets with neighboring price elements and predict the close price for the next day. However, in order to compare and recognize patterns in the behavior it is very important to define a proper feature that can be recognized by the algorithms. We do this by using the Open, Close, High, and Low prices and converting them into fractions based on how much the values have changed since the opening price of that day. We define 3 such fractions:

Fchange = (Close-Open) / Open

Fhigh = (High-Open) / Open

Flow = (Open-Low) / Open

A vector of these 3 features can be used to represent a single observation i.e., changes in stock prices in a single day. This single observation is:

Xt = <Fchange, Fhigh, Flow>

As the observations are a vector of continuous random variables, assuming that the emission probability distribution is continuous we can determine the parameters for the transition matrix, A and prior probabilities, π. This is done by fitting the training data to the 'GaussianHMM' class of the hmmlearn package. Once our model is trained, we can compute the Xt = <Fchange, Fhigh, Flow> observation vector given the observation data for t days, x1,…,xt, and the parameters of the HMM which are already computed during the training of the model.

Thus, the output of the HMM model should be such that it maximizes the posterior probability of:

$P(Xt+1|X1,\ldots,Xt,\theta)$ where $\theta = \{A, B, \pi\}$

The predicted Xt+1 represents the observation on the day for which Open price is known(from the test dataset). The Fchange fraction from this observation can be then used to compute the Close price for that day as:

Close = Open * (1 + Fchange)

Pseudocode:

```
Step 1: Initialize the datasets:
trainDS = pd.read_csv('GOOGL_Train.csv')
testDS = pd.read_csv('GOOGL_Test.csv')

Step 2: Initialize the HMM:
hmm = hmmlearn.hmm.GaussianHMM(n_components=hiddenStatesCount)

Step 3: Convert stocks values to fractions that will be used as features using training dataset
FChange = (close - open) / open
FHigh = (high - open) / open
FLow = (open - low) / open
FeatureVector = < FChange, FHigh, FLow >

Step 4: Train/fit the features to the hmm model
hmm.fit(FeatureVector)


Step 5: Define all possible next values for the feature vectors by dividing these fractional changes into some discrete values ranging between two finite variables
FChangeSteps = {-0.1, -0.075, -0.050, -0.025, 0, 0.025, 0.050, 0.075, 0.1}
FHighSteps = {0, 0.020, 0.040, 0.060, 0.080, 0.1}
FLowSteps = {0, 0.020, 0.040, 0.060, 0.080, 0.1}
possibleNextValues = Different permutations and combinations of <FChangeSteps, FHighSteps, FLowSteps>

Step 6: Predict the fraction change value for current day using previous day data as input to get score from the trained model
```

```
25. for outcome in possibleNextValues:
26. -previous10DayData from test dataset
27. -create vector of previous10Daydata and
    outcome values
28. -get score for the vector hmm.score(vector)
29. -add score to a list
30. Select the outcome with the max score
31.
32. Step 7: Calculate the close value for cur-
    rent day based on the open price and the
    predicted fraction change value
33. predictedFChange, predictedFHigh, predict-
    edFLow = outcome from Step 6
34. close = open * (1 + predictedFChange)
```

Analyzing the outcome.
Our chosen metric for assessing the performance of the model is Root Mean Square Error (RMSE).
It is given by the formula,

$$RMSE = \sqrt{\frac{\sum_{i=0}^{n}(yi - \tilde{y}i)^2}{n}}$$

Where n is the total number of observations, yi is the actual value at i in the series, $\tilde{y}i$ is the value at i predicted by the model.

The RMSE is used to measure the differences between values predicted by the model and the actual values. A lower RMSE value indicates that the model has greater accuracy and a better fit. The metric is used to compare the performance of different models on the same dataset, it cannot be used to compare the performance between datasets. Comparing values of RMSE for a model on two different datasets is an invalid metric.

Experiments and Results

Long Short-Term Memory Recurrent Neural Network (LSTM-RNN)
We base our experimentation on the steps we described in our methodology.

- How does changing the window size for Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) affect the performance?
- How does changing the features that we select for training affect the performance?

We begin with a simple univariate implementation of the LSTM-RNN. The GOOGL training data is used and our chosen feature is the "Close" price to predict future values. Initially we start with a single feature and configure the model for one-step forecast as implemented in Sima Siami-Namini, Akbar Siami Namin (2018). Subsequently, different window sizes are used for prediction. We use window sizes of 1, 10, 15, 30, and 60.

	Window size(days)				
Features	1	10	15	30	60
1	483.59	366.64	359.2	360	280

Table 1: Average RMSE values for GOOGL using single feature and different window sizes

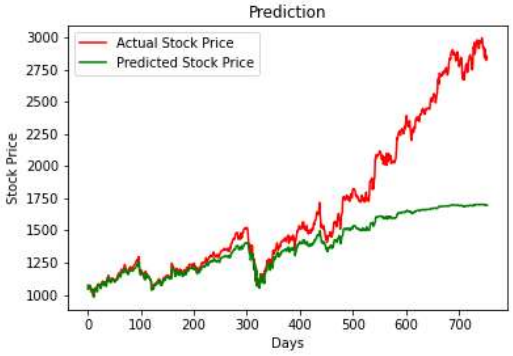

Figure 4: prediction for single feature with one window size

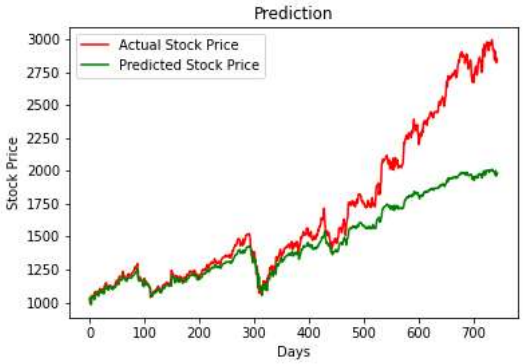

Figure 5: prediction for single feature with 10 window size

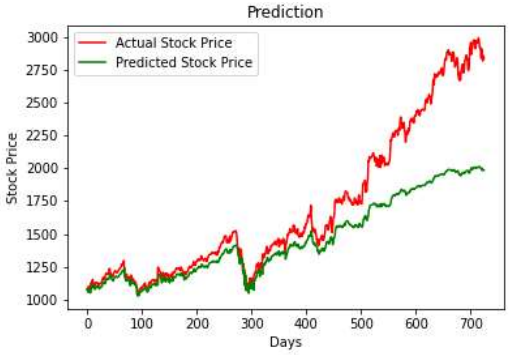

Figure 6: prediction for single feature with 30 window size

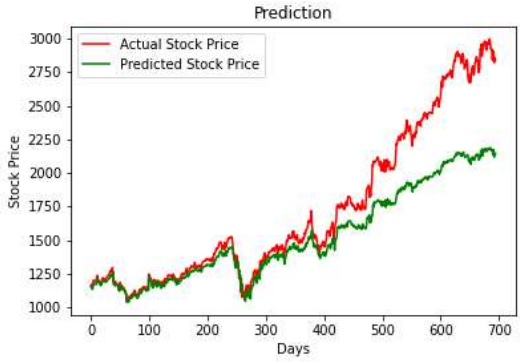

Figure 7: prediction for single feature with 60 window size

The predictions obtained by using a single feature indicates that the model can predict the trends of the stock market with varying accuracy for different window sizes. The table shows that the ability of the univariate algorithm to accurately predict the values is better for larger window sizes where LSTM has memory of longer periods of data.

Next configuration is to implement a multivariate algorithm that uses 5 features Open, Close, High, Low and Volume as the input for the model. This implementation is run for the same window sizes.

	Window size(days)				
Features	1	10	15	30	60
1	483.59	366.64	359.2	360.7	331.49
5	30.45	40.6	39.51	49.64	50.34

Table 2: Average RMSE values for GOOGL using 5 features and different window sizes

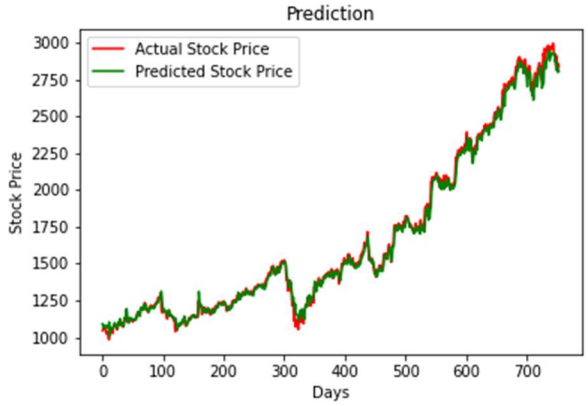

Figure 8: prediction for multiple features with 1 step window size

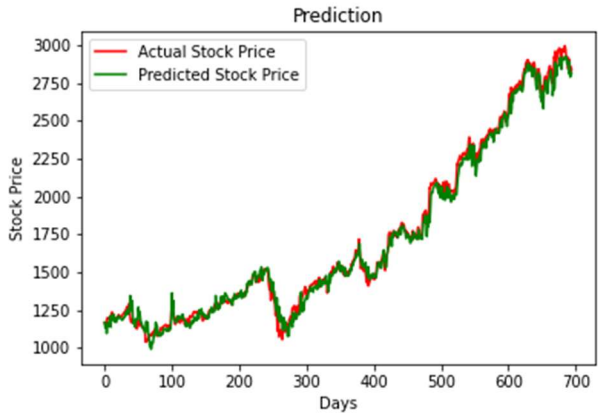

Figure 9: prediction for multiple features with 60 step window size

The predicted values using the 5 features of the stock as input are much more promising. In comparison to the univariate algorithm that uses Close as the single feature, the multivariate algorithm using 5 features of the stock data performs much more accurately. There are some contrasting findings as compared to the use of a single feature model. The accuracy for smaller time steps seems to be more accurate for the multi-feature model in contrast to the single feature.

But overall much greater accuracy in predicting both the trend as well as the stock values is observed. We see on average 88% increase in accuracy using the selected 5 features, and at best a 93% increase in the accuracy. Across all our trials we averaged a training loss around 0.0010 for use of 150 epochs for training.

The same model configuration are run on the TSLA dataset. We observe similar pattern accuracy using in the RMSE as we see for the GOOGL dataset. This means that the model works accurately for different datasets and is not overfit or underfit.

		Window size(days)				
Features	Stock	1	10	15	30	60
1	GOOGL	483.59	366.64	359.2	360	280
	TSLA	121.302	131.333	137.64	148.094	145.775
5	GOOGL	30.45	40.6	39.51	49.64	50.34
	TSLA	32.42	33.33	34.409	36.32	48.227

Figure 10: prediction for multiple features with different timesteps

Hidden Markov Model

The selection of the correct parameters for training and testing the HMM was done on the basis of the performance. The performance measuring metric used was Root Mean Squared Error (RMSE). As can be viewed from table 3, the

variation in Latency Days (number of previous day data used for Scoring Test data) does not affect the RMSE.

Configuration	Latency Days: 50 Fchange steps: 50 Fhigh steps: 10 Flow steps: 10	Latency Days: 10 Fchange steps: 50 Fhigh steps: 10 Flow steps: 10	Latency Days: 2 Fchange steps: 50 Fhigh steps: 10 Flow steps: 10
RMSE	31.21	31.21	31.21
Prediction Time	11 mins 50 secs	11 mins 54 secs	11 mins 56 secs

Table 4: Variation in RMSE based on Latency days

In contrast to that, table 4 shows the changes in RMSE values as the steps (number of discrete values between two variables) are changed. It can be observed that as the steps are reduced the RMSE also reduces until a point and then increases again. The prediction time also reduces significantly as the steps are reduced. The most optimal parameter assignment was found to be when $F_{ChangeSteps} = 3$, $F_{HighSteps} = 3$, $F_{LowSteps} = 3$

Configuration	Latency Days: 10 Fchange steps: 50 Fhigh steps: 10 Flow steps: 10	Latency Days: 10 Fchange steps: 25 Fhigh steps: 10 Flow steps: 10	Latency Days: 10 Fchange steps: 15 Fhigh steps: 10 Flow steps: 10
RMSE	31.21	29.42	27.04
Prediction Time	11 mins 54 secs	05 mins 59 secs	03 mins 38 secs
Configuration	Latency Days: 10 Fchange steps: 10 Fhigh steps: 10 Flow steps: 10	Latency Days: 10 Fchange steps: 5 Fhigh steps: 10 Flow steps: 10	Latency Days: 10 Fchange steps: 2 Fhigh steps: 10 Flow steps: 10
RMSE	24.13	18.35	134.11
Prediction Time	02 mins 28 secs	01 mins 16 secs	31 secs
Configuration	Latency Days: 10 Fchange steps: 3 Fhigh steps: 3 Flow steps: 3		
RMSE	18.35		
Prediction Time	45 secs		

Table 5: Variation in RMSE based on steps

The difference in the predicted values for Google Stock test dataset can be visualized from the figures 11, 12 and 13 below. Figure 11 represents the configuration where Latency Days = 10, $F_{ChangeSteps} = 50$, $F_{HighSteps} = 10$, $F_{LowSteps} = 10$. It shows the predicted values following the actual value line closely. However, Figure 12 shows the results where the predicted values and actual values seem to overlap most of the time. The configuration for this graph is the optimal configuration that was selected in table 4. Figure 13 shows the results for $F_{ChangeSteps} = 2$ scenario which provided the least suitable prediction for the test dataset.

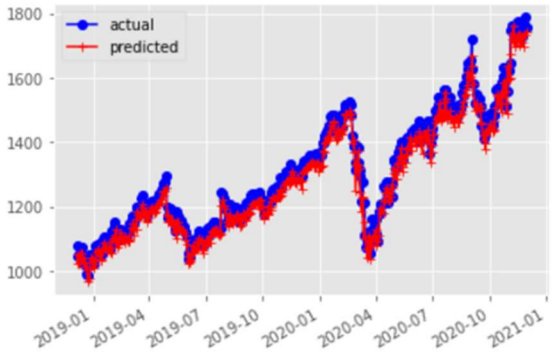

Figure 11: Prediction of values for 10 latency days and 50 FChangeSteps

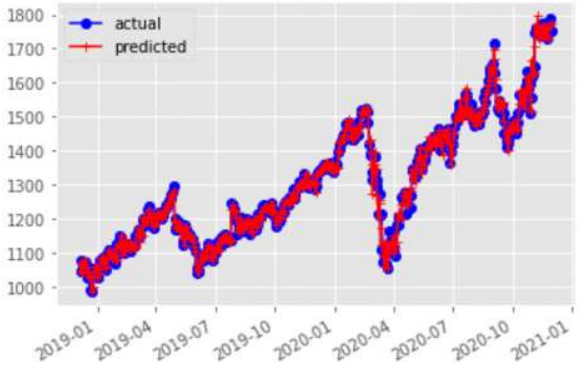

Figure 12: Prediction of values for 10 latency days and 3 FChangeSteps

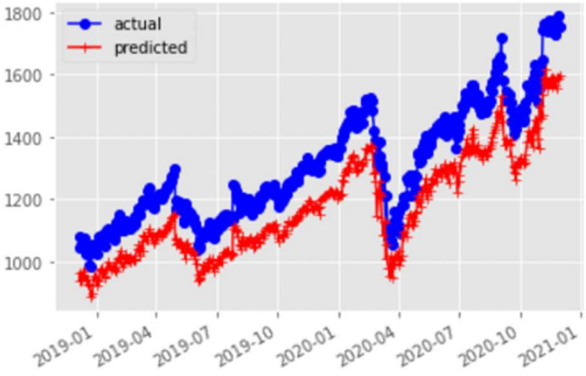

Figure 13: Prediction of values for 10 latency days and 2 FChangeSteps

We verify our findings by testing the model on the TSLA dataset and check if we achieve similar results

Conclusion

This paper analyzes two machine learning techniques HMM and LSTM-RNN. The authors optimize each model for

prediction of stock market value, then we compare the two models on the basis of accuracy.

In this paper, we see the different variations of LSTM in univariate and multivariate forecasting and trials on different timesteps. Several studies such as Sima Siami-Namini, Akbar Siami Namin (2018) show that machine learning model such as LSTM are superior to traditional statistical models such as ARIMA. It is reported that the LSTM based algorithm performed 85% better compared to ARIMA. It is observed that multivariate forecasting was superior to the univariate forecasting we see in several studies before. On an average there is an 88% improvement in accuracy over the single feature forecasting using the multiple features chosen in this study irrespective of the window sizes chosen.

In case of HMM, the performance is even better compared to LSTM-RNN as can be seen by comparing the RMSE values for both the models. We were successfully able to train HMM models to give results that were similar to the ones shown in studies by Md. Rafiul Hassan, Baikunth Nath (2005). This opens doors to developing models that could be a hybrid of Neural Networks and Markov processes in the future

References

Adil Moghar, Mhamed Hamiche (2020) Stock Market Prediction Using LSTM Recurrent Neural Network

Brownlee J, Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras

Christy Jackson, J; Prassanna, J; Abdul Quadir,Md; Sivakumar;V(2020) Stock market analysis and prediction using time series analysis

Colah's Blog 2015 Understanding LSTM Networks -- colah's blog

Diederik P. Kingma, Jimmy Ba 2015 Adam: A Method for Stochastic Optimization

Erkam Guresen, Gulgun Kayakutlu, Tugrul U. Daim (2011) Using artificial neural network models in stock market index prediction

Kang, N 2017 Multi-Layer Neural Networks with Sigmoid Function— Deep Learning for Rookies (2)

Md. Rafiul Hassan, Baikunth Nath (2005) Stock Market Forecasting Using Hidden Markov Model: A New Approach

Roh, T. H. (2007). Forecasting the volatility of stock price index.Expert Systems withApplications, 33, 916–922.

Sepp Hochreiter, Jürgen Schmidhuber (1997) Long Short-Term Memory

Sima Siami-Namini, Akbar Siami Namin (2018) Forecasting Economics and Financial Time Series: ARIMA vs. LSTM.

Venugopal Prakash, Vigneshwaran T (2019) State-of-Health Estimation of Li-ion Batteries in Electric Vehicle Using IndRNN under Variable Load Condition