Forecasting Time series Data

Greena Simon

#5024 5123

University at Buffalo

Abstract

Time series data requires a different set of tools and techniques for predictions compared to other datasets. It is the time dependency between the data points that makes a time series unique and more complex. This paper discusses about some of the major methodologies like ARIMA models, long-short term memory networks, support vector machines etc. and their application on several time series data using Python and R. The paper also includes a model comparison within the context of the data used.

Introduction

Time series is a series of data points indexed in the order of time. So, what is different about time series data? Unlike other sets of observations, the time component makes it difficult to handle. A basic assumption for linear regression that observations are independent of each other does not hold here because of the time dependency. In addition, along with a decreasing or increasing trend in values, there could be seasonal patterns as well. Monthly sales of a retail store could be high during holiday seasons.

A time series can have four different components embedded to it.

- **Level:** The baseline of values if it were to be a straight line.
- **Trend:** Linearly increasing or decreasing values over time.
- **Seasonality**: Cycles/ repeated patterns in the observations.
- **Noise**: Variance that cannot be explained by the model.

All time series have a level, most have noise, and the trend and seasonality are optional. Depending on the characteristics a series possesses, data is treated for predictions and modelling.

Stock market prediction is one of the classic time series problems due to some of its interesting properties like randomness, volatility, other complex dependencies, benefits associated etc. Hence to make things further interesting, I am using four different stock prices; Facebook, General Electric, Apple and Goldman Sachs. I am also including bitcoin USD prices so that we have a highly volatile set as well. The historic price data is downloaded directly from yahoo finance using python hence I would not be using any external datasets for my analysis.

First, I will explain different methods and its basics within the current context and then will move on to python/R implementations and model comparisons.

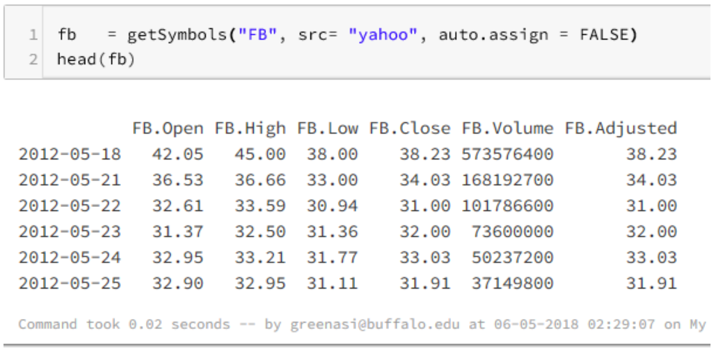
Implemented models for time series forecasting

ARIMA Models

A classic way of dealing with time series data is through ARIMA (Auto Regressive Integrated Moving Averages) models. A series has to be at least weakly stationary to apply ARIMA on it. A series is said to be stationary when its statistical properties as mean, variance and covariance are tend to be constant over a period. A majority of the time series models work under the assumption that the series is stationary. It is easier to work with this assumption because if a time series exhibits a particular behaviour over a period, it is highly likely that it will follow the same in the future. Most of the time series data we encounter are not stationary by default. They will need to be explicitly converted using methods like Differencing, De-trending, transformations etc.

ARIMA is easier to implement using R. Hence, R is used here for modelling.

Below is the glimpse of one of the datasets we have taken from yahoo finance.

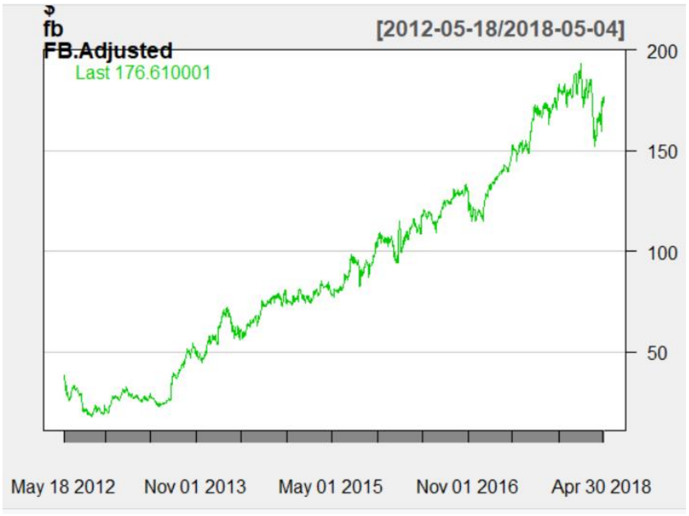


```
fb   = getSymbols("FB", src= "yahoo", auto.assign = FALSE)
head(fb)
```

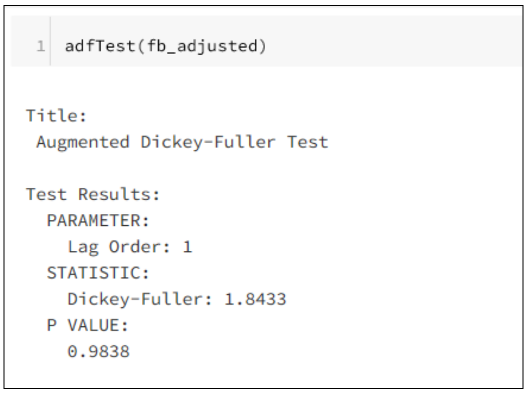
```
           FB.Open FB.High FB.Low FB.Close FB.Volume FB.Adjusted
2012-05-18   42.05   45.00  38.00    38.23 573576400       38.23
2012-05-21   36.53   36.66  33.00    34.03 168192700       34.03
2012-05-22   32.61   33.59  30.94    31.00 101786600       31.00
2012-05-23   31.37   32.50  31.36    32.00  73600000       32.00
2012-05-24   32.95   33.21  31.77    33.03  50237200       33.03
2012-05-25   32.90   32.95  31.11    31.91  37149800       31.91

Command took 0.02 seconds -- by greenasi@buffalo.edu at 06-05-2018 02:29:07 on My
```

If we plot the data with Adjusted close price, it is clear that the data is not stationary.



However, it is not always as easy as looking at the plot. One method to validate the stationarity of a series is through **Dickey-Fuller Test**. In this statistic-based test, the null hypotheses states that the series is non-stationary.

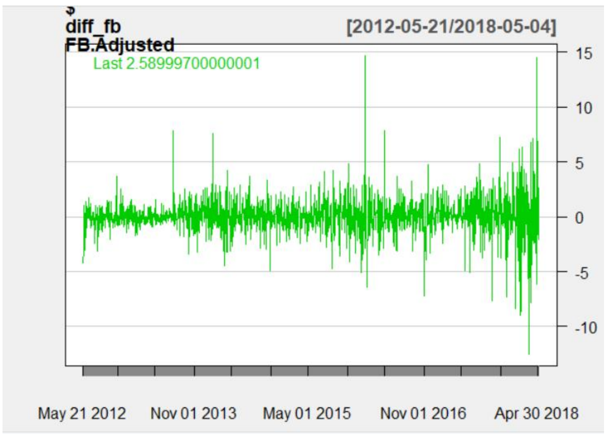


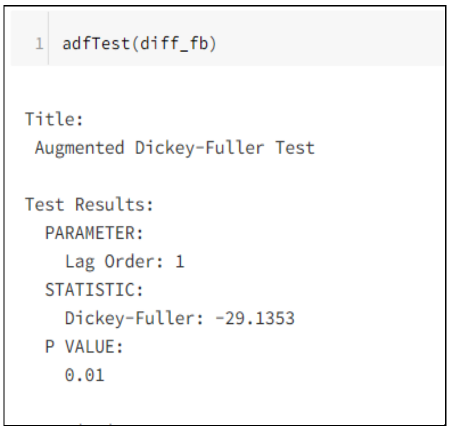
```
adfTest(fb_adjusted)
```

```
Title:
 Augmented Dickey-Fuller Test

Test Results:
  PARAMETER:
    Lag Order: 1
  STATISTIC:
    Dickey-Fuller: 1.8433
  P VALUE:
    0.9838
```

A very high p value indicates that we cannot reject the null hypotheses. As we discussed above, to achieve stationarity methods like transformation, moving average, differencing etc. are used. After first order differencing, the series looks like below.





```
adfTest(diff_fb)


Title:
 Augmented Dickey-Fuller Test

Test Results:
  PARAMETER:
    Lag Order: 1
  STATISTIC:
    Dickey-Fuller: -29.1353
  P VALUE:
    0.01
```

P value is low and hence with 95% confidence, we can reject the claim that series is non-stationary. Now that we know how to make a series stationary, we can use ARIMA to model the data.

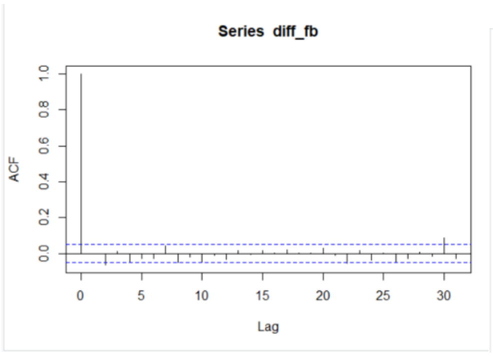
A non-seasonal ARIMA has 3 terms associated with it and represented as ARIMA(p, d, q) which includes,

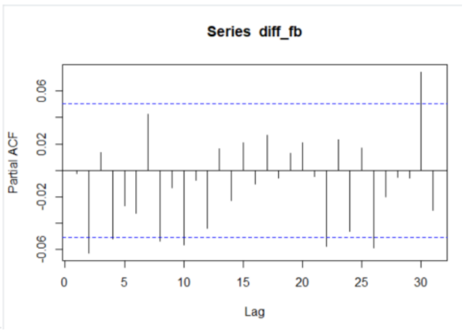
- **An auto regressive (AR) term p:** Shows the lags of the stationary series in the forecasting equation. Suppose if p = 3, then predictors of x_t will be x_(t-1), x_(t-2) and x_(t-3).
- **A moving average (MA) term q:** Shows the lags forecast errors in the equation. i.e. if q = 3 then predictors of x_t will be e_(t-1), e_(t-2) and e_(t-3) where e_i is the difference between moving average at i th instant and actual value.
- **Integrated(I) term d:** Indicates the number of non-seasonal differences needed for stationarity.

To calculate p and q, we need to know 2 other functions. ACF and PACF.

- **Auto correlation Function (ACF):** Gives a plot of total correlation between different lag functions. i.e correlation of x_t with x_(t-1), x_(t-2).. etc. So for a MA series of lag n, we will not see any significant correlation with x_(t-n-1). i.e there will be a cut off at the n th lag. This will be our q. For an AR series, correlation will gradually go down without any significant cut off.
- **Partial Auto Correlation Function(PACF):** Gives the partial correlation of time series with it's own lagged values, controlling for the values of time series at all shorter lags in contrast to ACF. So for lag 3 series, if we remove the effect of x_(t-1)..x_(t-3) , x_(t-4) will have no significant impact on x_t. There will be a sharp decline after 3 lags. This will be our p.

For the stationary FB series ACF and PACF plots are as per below.





Correlation plots similar to above indicates that the series is a random walk. A random walk is movements of an object or changes in a variable that follow no discernible pattern or trend and solely depend on its previous position or value. A knight's movements in a chessboard is a good example for a random walk. It can be modelled using Arima(0,1,0) or Arima(1,0,1). Adding a drift to the model embeds a trend to it otherwise, the prediction would be a flat line equal to its last observation.

Cmd 17

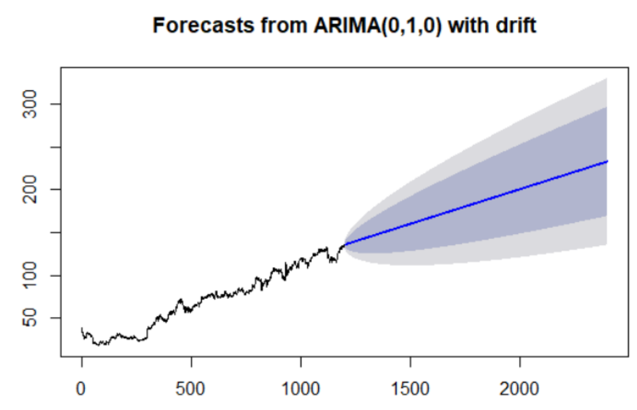
```
model_fb= Arima(fb_train$FB.Adjusted, c(0,1,0), include.drift = TRUE)
model_fb
```

In addition, forecast is given by,

Cmd 19

```
forecast(model_fb, test)
```

Forecast plot predicted by ARIMA.



```
           FB.Adjusted    Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2017-02-27      136.41          135.5211 133.6797 137.3625 132.7049 138.3373
2017-02-28      135.54          135.6022 132.9980 138.2063 131.6194 139.5849
2017-03-01      137.42          135.6832 132.4938 138.8727 130.8054 140.5610
2017-03-02      136.76          135.7643 132.0815 139.4471 130.1319 141.3967
2017-03-03      137.17          135.8454 131.7279 139.9629 129.5482 142.1426
2017-03-06      137.42          135.9265 131.4159 140.4370 129.0282 142.8247
2017-03-07      137.30          136.0075 131.1356 140.8795 128.5566 143.4585
2017-03-08      137.72          136.0886 130.8803 141.2969 128.1232 144.0540
2017-03-09      138.24          136.1697 130.6454 141.6939 127.7211 144.6183
2017-03-10      138.79          136.2508 130.4277 142.0738 127.3452 145.1564
2017-03-13      139.60          136.3318 130.2246 142.4391 126.9916 145.6721
2017-03-14      139.32          136.4129 130.0341 142.7918 126.6573 146.1685
2017-03-15      139.72          136.4940 129.8547 143.1333 126.3400 146.6479
2017-03-16      139.99          136.5751 129.6851 143.4650 126.0378 147.1123
2017-03-17      139.84          136.6561 129.5244 143.7879 125.7490 147.5632
2017-03-20      139.94          136.7372 129.3716 144.1029 125.4724 148.0020
2017-03-21      138.51          136.8183 129.2259 144.4106 125.2068 148.4298
2017-03-22      139.59          136.8994 129.0869 144.7118 124.9512 148.8475
```

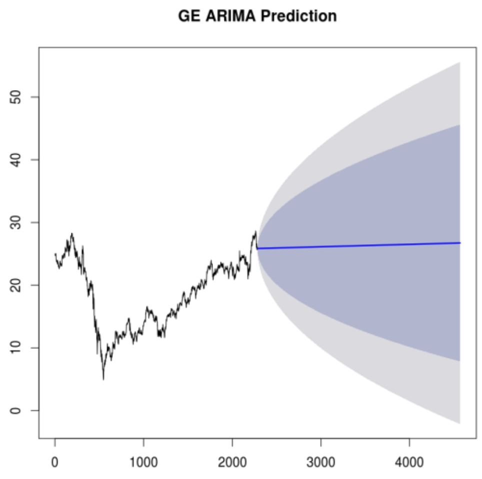
Accuracy of the model is as given below.

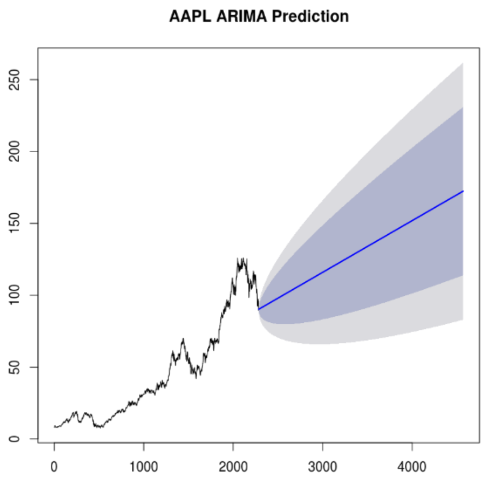
```
accuracy(forecast(model_fb, 100),fb_test$FB.Adjusted[1:100])
```

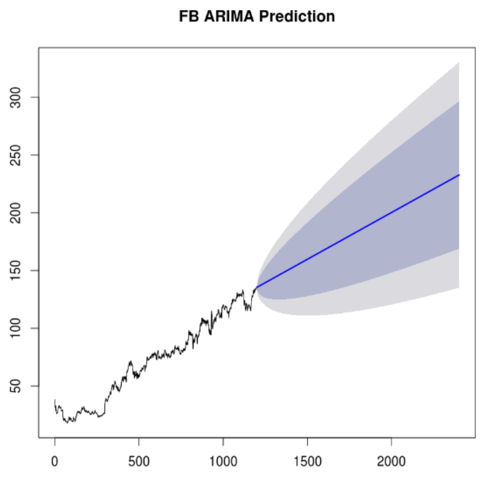
```
                        ME      RMSE       MAE         MPE     MAPE      MASE
Training set 3.179075e-05 1.435665 0.9856549 -0.08114989 1.623345 0.9980662
Test set     7.648065e+00 8.906098 7.6493085  5.07124522 5.072162 7.7456281
                   ACF1
Training set 0.03295019
Test set             NA
```

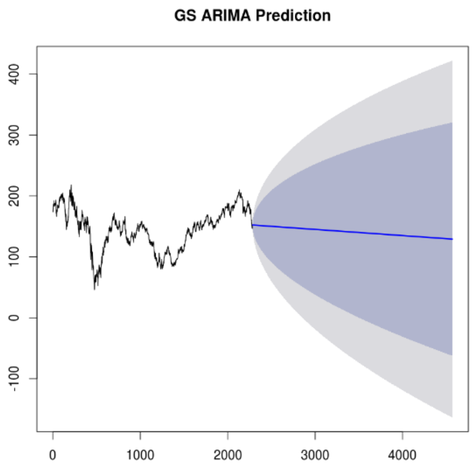
Similarly, applied Random walk Arima for other datasets as well and found mean square errors for predicting the next 100 data points and their forecast plots are as shown below.

#	Dataset	RMSE Training	RMSE Test
1	Facebook	1.43	8.90
2	Apple	0.95	7.24
3	General Electric	0.308	2.13
4	Goldman Sachs	3.12	5.78
5	Bitcoin USD	10.54	890.67









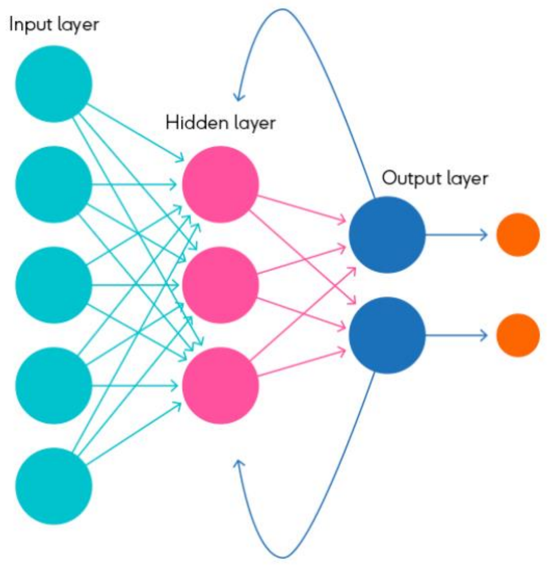
In addition, RMSE increases for all the datasets as the prediction dates get further away into the future.

R implementation: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/760519810048786/1587519275671800/1992077388330115/latest.html

Recurrent Neural Networks/LSTM

Neural networks or ANNs are a very rough model of how a human brain is structured. It comprises of an input layer, several hidden layers and an output layer. Patterns/ Inputs are given to the network through the input layer, processed in the hidden layers and then linked to the output layer to receive the results.

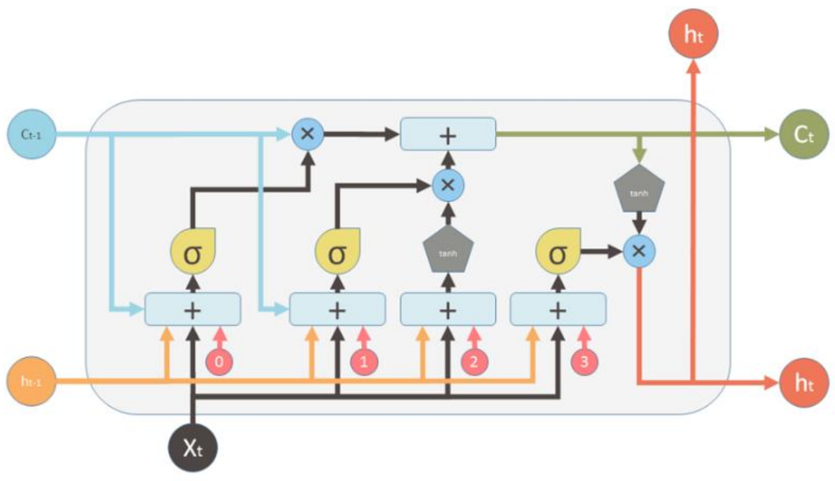
Even though ANNs are trained within the context of past data, predictions they make at each epoch are independent of the prior patterns/inputs. They don't possess the most important characteristic of time series - the notion of order in time, i.e., their previous decisions have no effect on the current decision. **Recurrent neural networks (RNNs)** were created to address this flaw in ANNs. RNNs not only consider the current input, but also look at what they perceived previously in time. The picture below can give you a basic idea.



The feedback loop from the output layer to a hidden layer is what differentiates RNNs from ANNs. As you can see, the decision that an RNN reaches at step t-1 will impact its decision at step t. RNNs use backpropagation through time(BPTT) algorithm for learning. BPTT is an extended version of the basic backpropagation algorithm and works by unrolling all input timesteps. Each timestep has one input timestep, one copy of the network, and one output. Errors are then calculated and accumulated for each timestep. The network is rolled back up and the weights are then updated for the network.

But RNNs are not perfect either. A network learns its parameters using gradient based methods by understanding how changes in value of these parameters affect the network output. When we backpropagate in the network and calculate gradients of loss w.r.t weight changes, the gradients tend to get smaller and smaller as we move backwards. This is majorly because many of the activation functions that we use in the network squash the input values to a very small range of output values. (between 0 and 1) and as the number of layers increase, output values get mapped to an extremely small range. If the effect is too small, network cannot learn the parameters effectively. This is called vanishing gradient problem. In the case of an RNN with too many hidden layers, earlier layers tend to learn very slowly compared to the later layers in the hierarchy. For that reason, training the network could take long and the accuracy of the prediction will suffer.

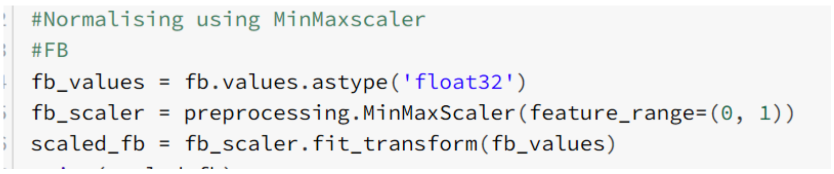
Long-short term memory network or LSTMs are another implementation of RNNS that try to resolve the vanishing gradient problem. LSTMs are capable of remembering information for longer periods of time. They create a distinction between relevant and irrelevant information whereas RNNs modify the whole information in hand without any such consideration.



A single LSTM network takes 3 inputs - Inputs of the current time step(X_t), Output of the previous timestep(h_t-1) and Memory of the previous unit(C_t-1). As outputs, it emits a new memory (C_t) and output of the current network (h_t).

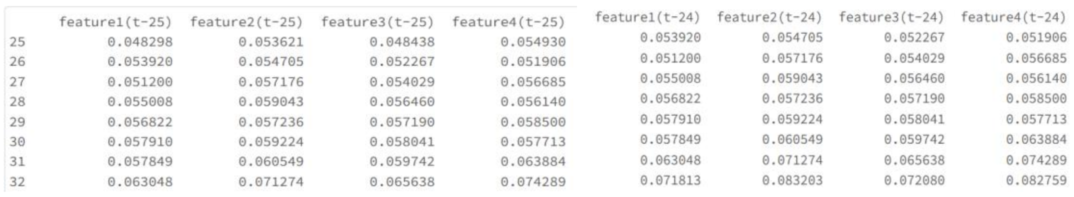
The flow of memory from the previous unit is handled by a forget gate. Output of the forget valve (will be between 0 and 1) is created by previous output, current input and a bias b_0 and is applied on the previous memory by multiplication. If the forget valve is completely shut (value 0), no memory will pass through and it is forgotten. If it is fully open, old memory will pass through completely and a value between 0 and 1 indicates some old memory will be allowed to pass through. Second one is the new memory gate or input gate which has the same input as forget valve and determines how much of the new memory should affect the old memory. Element wise multiplication with the output from tanh activation function implements the control here and this new memory will be added('+') to the old memory to form the new memory of the unit C_t. Finally, we generate the outout h_t which will be influenced by the new memory, current input, previous output and a bias vector(b_3). The output valve controls how much new memory should be outputted to the next LSTM unit. Gradient in LSTMs may also vanish. But it tends to happen very slowly compared to RNNs enabling them to catch long distant dependencies.

LSTMs can be easily implemented through python scikit library. Hence, python is used here to illustrate this choice of modelling. Data was pulled same way as before from yahoo finance. Scaled down using MinMaxScaler as neural networks are sensitive towards the data scales.



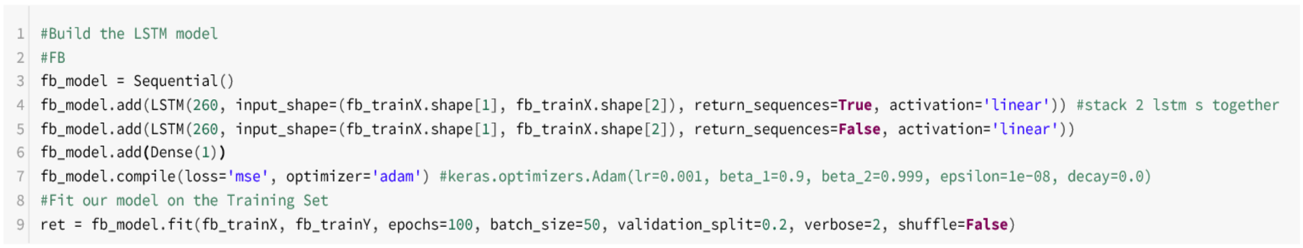
```
#Normalising using MinMaxscaler
#FB
fb_values = fb.values.astype('float32')
fb_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
scaled_fb = fb_scaler.fit_transform(fb_values)
```

Scaled values were combined with ‘n’ number of previous time step outputs to create an input sequence and fed into LSTM. Different values of input timesteps ‘n’ were used for each iteration of model training for better accuracy. The Input sequence after adding ‘n timesteps looked like below from t-n to t.



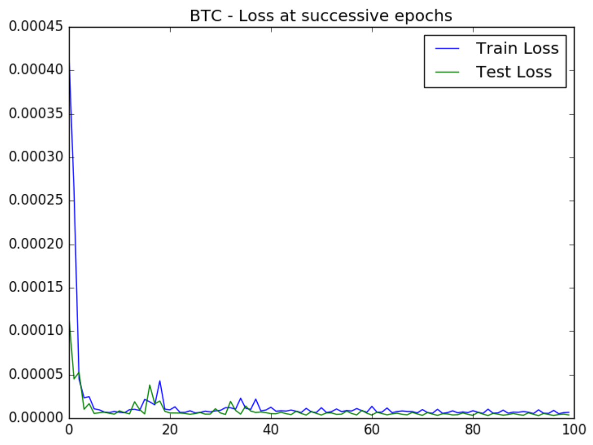
	feature1(t-25)	feature2(t-25)	feature3(t-25)	feature4(t-25)	feature1(t-24)	feature2(t-24)	feature3(t-24)	feature4(t-24)
25	0.048298	0.053621	0.048438	0.054930	0.053920	0.054705	0.052267	0.051906
26	0.053920	0.054705	0.052267	0.051906	0.051200	0.057176	0.054029	0.056685
27	0.051200	0.057176	0.054029	0.056685	0.055008	0.059043	0.056460	0.056140
28	0.055008	0.059043	0.056460	0.056140	0.056822	0.057236	0.057190	0.058500
29	0.056822	0.057236	0.057190	0.058500	0.057910	0.059224	0.058041	0.057713
30	0.057910	0.059224	0.058041	0.057713	0.057849	0.060549	0.059742	0.063884
31	0.057849	0.060549	0.059742	0.063884	0.063048	0.071274	0.065638	0.074289
32	0.063048	0.071274	0.065638	0.074289	0.071813	0.083203	0.072080	0.082759

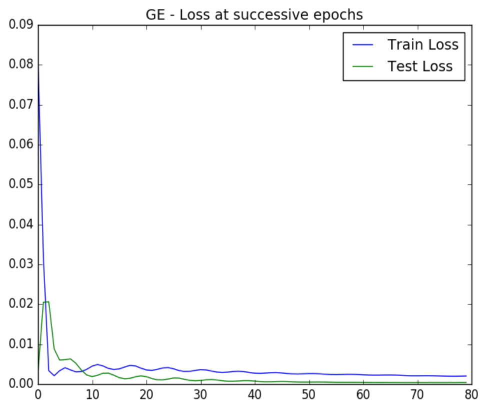
The models were configured by stacking multiple LSTMs after reshaping the input sequence to 3D array. Again, model training was repeated several times with different configurations for number of LSTMs, number of input layers, number of hidden layers, activation function, batch size, epochs etc.

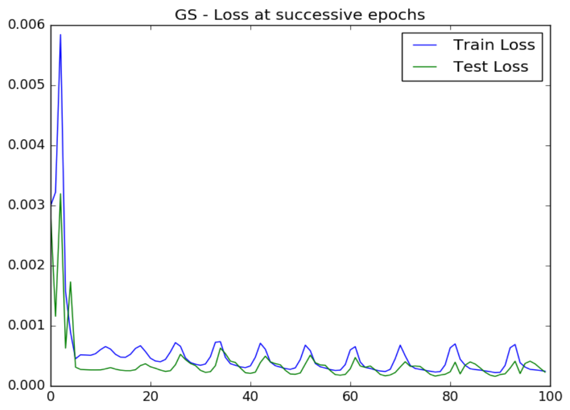


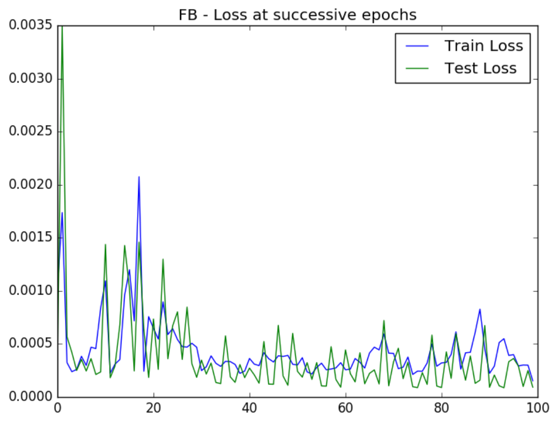
```
#Build the LSTM model
#FB
fb_model = Sequential()
fb_model.add(LSTM(260, input_shape=(fb_trainX.shape[1], fb_trainX.shape[2]), return_sequences=True, activation='linear')) #stack 2 lstm s together
fb_model.add(LSTM(260, input_shape=(fb_trainX.shape[1], fb_trainX.shape[2]), return_sequences=False, activation='linear'))
fb_model.add(Dense(1))
fb_model.compile(loss='mse', optimizer='adam') #keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
#Fit our model on the Training Set
ret = fb_model.fit(fb_trainX, fb_trainY, epochs=100, batch_size=50, validation_split=0.2, verbose=2, shuffle=False)
```

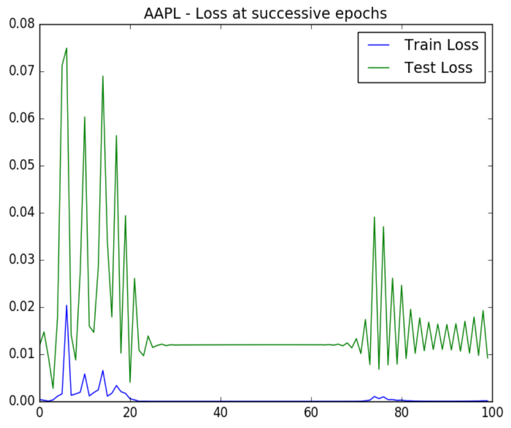
The value loss i.e., the residual sum of squares during training and validation for each dataset is shown below.







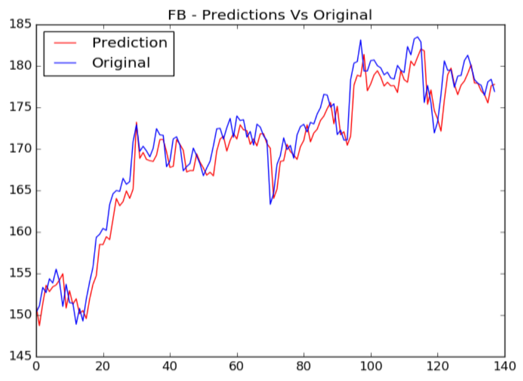


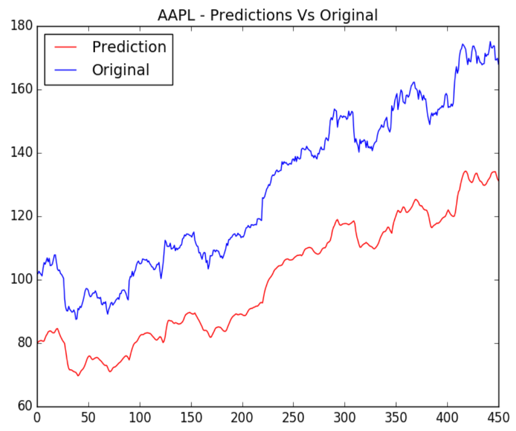


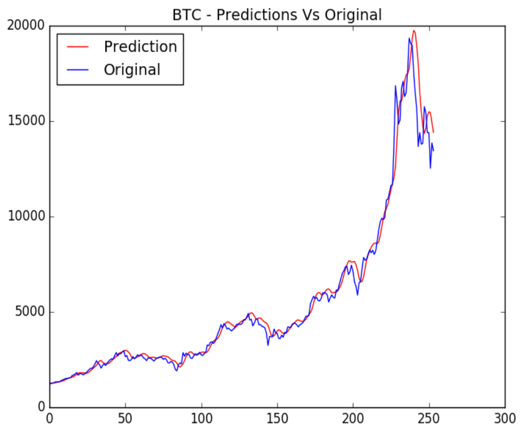
Mean square error (MSE) of each models prediction on test data and forecast plots are given below for the LSTM models. Also please note that the accuracy we have achieved here is based on a local minima we obtained during the training and unsure of a better model (Global Minima) exists or not since the MSE values depends on the model configurations we use.

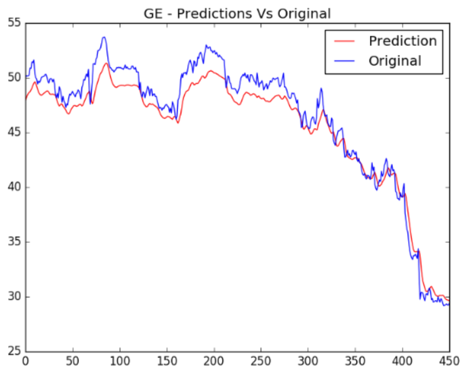
```
#Calculate RMSE
fb_rmse = sqrt(mean_squared_error(fb_original, fb_predictions))
print('FB - Test RMSE: %.3f' % fb_rmse)
```

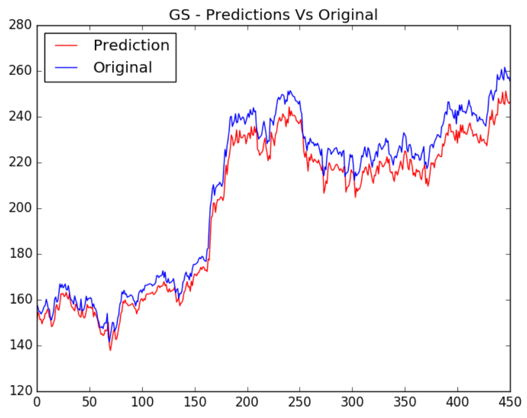
#	Dataset	MSE
1	Facebook Inc.	2.213
2	Apple Inc.	22.36
3	General Electric	1.4
4	Goldman Sachs Groups	7.54
5	Bitcoin USD	672.93







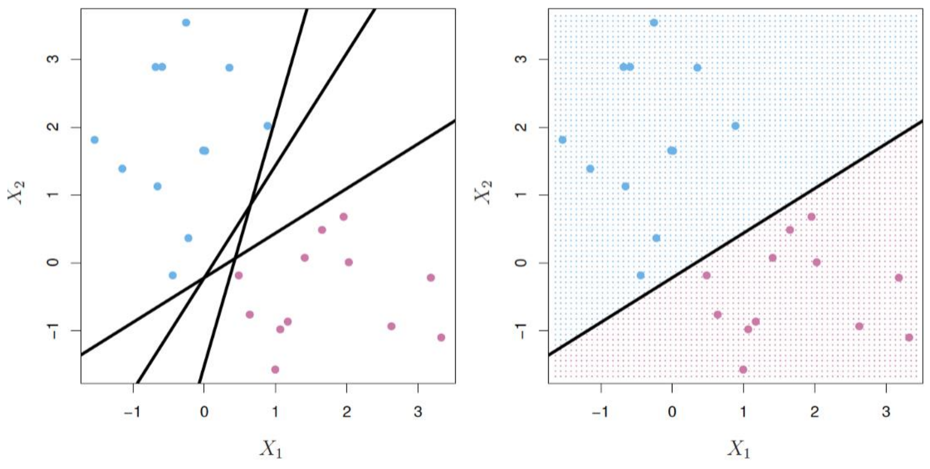




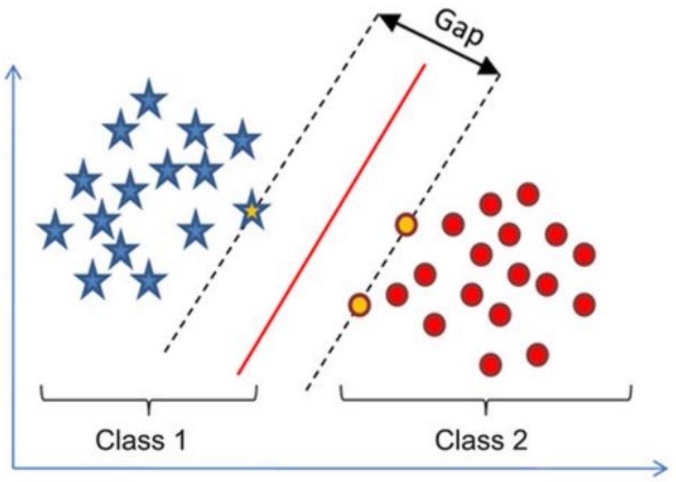
Python implementation: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/760519810048786/4109360145566906/1992077388330115/latest.html

Support Vector Machines

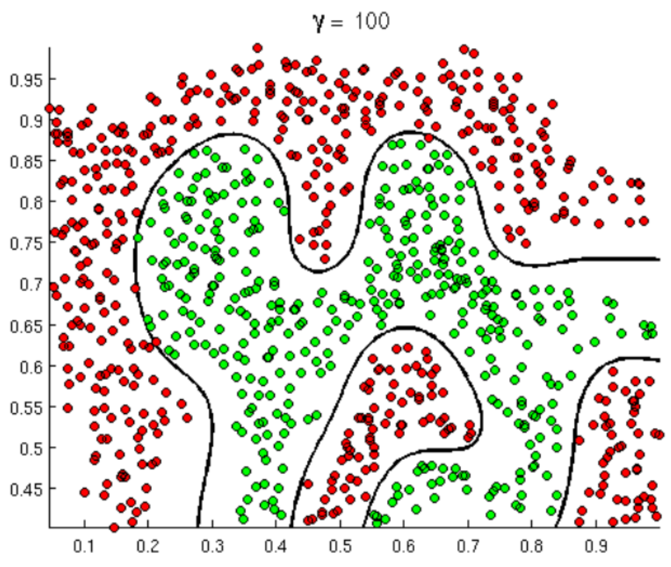
Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. Even though it is mostly used in classification problems, they are one of the popular models in forecasting time series data. Regression problems output a continuous value, which is simply a number that may or may not be bounded, but could take on an infinite number of values. In this algorithm, each data item is plotted as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the classes very well. This hyperplane is a linear separator for any dimension; it could be a line (2D), plane (3D), and hyperplane (4D+).



The best hyperplane is the one that maximizes the margin which is the decision boundary or maximum margin classifier. The margin is the distance between the hyperplane and a few close points. These close points are the support vectors because they control the hyperplane.



If we add new data, the Maximum Margin Classifier is the best hyperplane to correctly classify the new data. However, drawing a decision boundary is often not nearly as clear cut as the example above. It is common to see more complicated patterns that are not linear. But SVMs are capable of separating the non linear data as well as below.



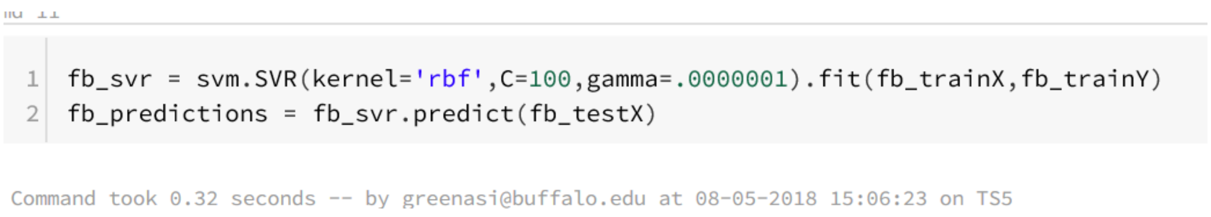
SVM does this through a technique called the kernel trick. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem and these functions are called kernels. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs defined. We have several kernel functions available for implementation like linear, rbf, poly etc. where poly and rbf are primarily for non linear separations. Linear kernel are used when there are more than thousand features and probability of separating them linearly is higher.

While separating non linearly, probability for overfitting could be high. Overfitting is controlled in SVMs by the parameter “C”. C tells us how much we want to tailor our model to the data points we observe, and thus, how curvy and spiky our decision boundary is. It also acts as measure trade-off between smooth decision boundary and classifying the training points correctly.

If the value of C is high, we might end up overfitting the data. A lower value of C will generalize the decision boundary and will create a less complex separator for the data. Similarly, there is another parameter gamma, which is the kernel coefficient for rbf or poly function we use. Again, a high value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.

SVM clearly works well with data that has a vivid margin of separation. Very effective in high dimensional data especially when number of dimensions are greater than the number of samples.

SVM here is implemented using scikit library in python. Similar to LSTM networks, input sequence has to re-shaped to 3-D array.



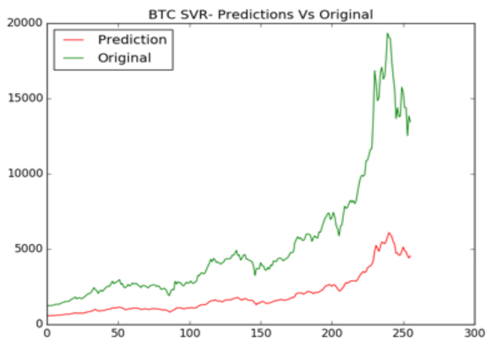
```
fb_svr = svm.SVR(kernel='rbf',C=100,gamma=.0000001).fit(fb_trainX,fb_trainY)
fb_predictions = fb_svr.predict(fb_testX)
```

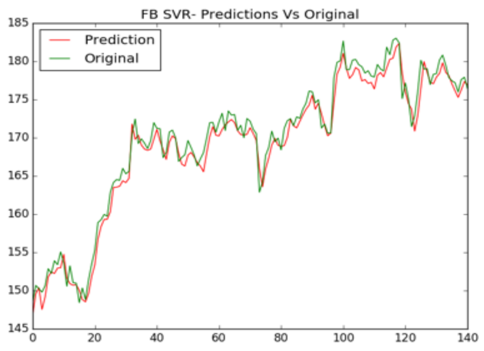
```
Command took 0.32 seconds -- by greenasi@buffalo.edu at 08-05-2018 15:06:23 on TS5
```

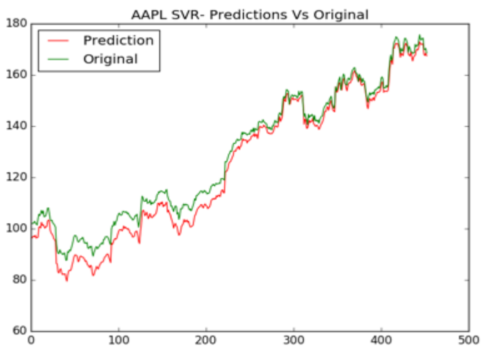
The model performance is described in the below table.

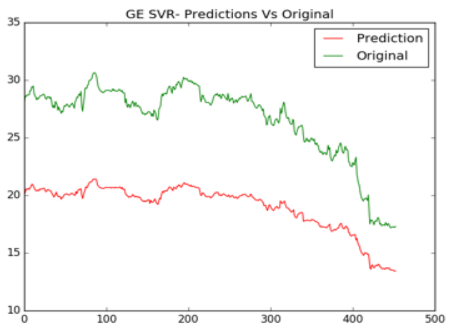
#	Dataset	MSE
1	Facebook Inc.	1.402
2	Apple Inc.	4.704
3	General Electric	7.664
4	Goldman Sachs Groups	12.142
5	Bitcoin USD	4447.452

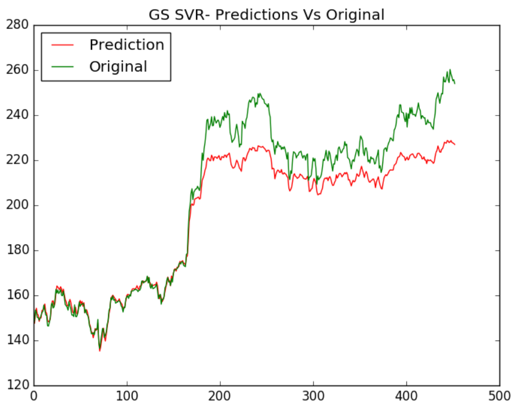
In addition, the forecast plots are given as,











Python Implementation: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/760519810048786/3420983527076539/1992077388330115/latest.html

Other Methods

There are a few other models, which started drawing more popularity recently. Basics of some of them are discussed here. Implementations of these models are out of scope of this paper.

Hidden Markov Models

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (i.e. hidden) states. Hidden Markov models are especially known for their application in reinforcement learning and temporal pattern recognition such as speech, handwriting, musical score following etc. and recently there researchers have started using them for exploring and forecasting time series data.

Markov chains are a common, and relatively simple, way to statistically model random processes. A Markov chain (model) describes a stochastic process where the assumed probability of future state(s) depends only on the current process state and not on any the states that preceded it. It has state space, transition matrix that includes probability distribution of all state transitions and initial probability of the states.

A Markov model becomes hidden when its true states are unknown to us and there are a set of observable behaviour available, which we assume, would represent the hidden states. They include one additional set of parameters called emission matrix, which indicates the probability for a hidden state given an observable behaviour. HMMs can be used to model three kinds of problems.

- Evaluation Problem: Given the HMM parameters and observed sequence, find the probability of an observed sequence. Forward Algorithm.
- Decoding: Given HMM parameters and observed sequence, find the most probable sequence of the hidden states. Forward-Backward Algorithm/Viterbi Algorithm.
- Learning: Given the HMM with unknown parameters and observed sequence, find parameters that will maximize the likelihood of the observed sequence. Baum-Welch Algorithm(EM)

We know that time series exhibit temporary periods where the expected means and variances are stable through time. These periods or regimes can be likened to hidden states. If that is the case, then all we need are observable variables whose behaviour allows us to infer the true hidden state(s). A combination of these algorithms could be used to explore and forecast the series effectively. Python has an hmmlearn package and R has an HMM package to support the modelling problems.

Prophet by Facebook

Prophet by Facebook is a recently developed model intending to resolve many of the usability issues of the existing forecasting methods. Major features include better handling of missing values and outliers and ability to consider historical trend changes while learning. Prophet’s default settings aims to produce forecasts that are often accurate as those produced by skilled forecasters, with much less effort. The forecast package includes many different forecasting techniques (ARIMA, exponential smoothing, etc), each with their own strengths, weaknesses, and tuning parameters. Forecasts are customizable in ways that are intuitive to non-experts. At its core, the Prophet procedure is an additive regression model with four main components:

- A piecewise linear or logistic growth curve trend. Prophet automatically detects changes in trends by selecting change points from the data.
- A yearly seasonal component modelled using Fourier series.
- A weekly seasonal component using dummy variables.
- A user-provided list of important holidays.

By default, Prophet will provide uncertainty intervals for the trend component by simulating future trend changes to your time series, which leads to judgement that is more accurate. Currently Facebook provides implementations of Prophet in both Python and R.

Summary

A table on the root mean square errors obtained by different models for all the datasets are given below.

#	**Dataset**	**ARIMA**	**LSTM**	**SVM**
1	Facebook Inc.	8.90	2.213	1.402
2	Apple Inc.	7.24	22.36	4.704
3	General electric	2.13	1.4	7.664
4	Goldman Sachs	5.78	7.54	12.142
5	Bitcoin USD	890.67	672.93	4447.452

As we can see, there is no best method available that fits all types and characteristics of the data and comes up with the best accuracy model for predictions. Occurrences of several local minima and inability of a model to recognize the global optimum makes the situation even worse. Model choices can vary based on time series properties. Hence, like any other datasets, time series analysis also should include exploring the data with various methods, understanding the strengths and weaknesses of each model choices and bring in a better model accordingly that can be actually useful in reality.

References

"Documentation of Scikit-Learn 0.19.1¶." *1.4. Support Vector Machines - Scikit-Learn 0.19.1 Documentation*, scikit-learn.org/stable/documentation.html.

Fuqua School of Business. "Introduction to ARIMA Models." *Seasonal Differencing in ARIMA Models*, people.duke.edu/~rnau/411arim.htm.

"Hidden Markov Model." *Wikipedia*, Wikimedia Foundation, 30 Apr. 2018, en.wikipedia.org/wiki/Hidden_Markov_model.

"Introduction to Hidden Markov Models with Python Networkx and Sklearn." *BLACKARBS LLC*, www.blackarbs.com/blog/introduction-hidden-markov-models-python-networkx-sklearn/2/9/2017.

Liberman, Neil. "Support Vector Machines – Neil Liberman – Medium." *Medium*, Augmenting Humanity, 2 Jan. 2017, medium.com/@neil.liberman/support-vector-machines-f79f50c9aa85.

Ray, Sunil, et al. "Understanding Support Vector Machine Algorithm from Examples (along with Code)." *Analytics Vidhya*, 14 Sept. 2017, www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

Srivastava, Tavish, et al. "A Complete Tutorial on Time Series Modeling in R." *Analytics Vidhya*, 20 Apr. 2018, www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/.

"Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras." *Machine Learning Mastery*, 21 Aug. 2017, machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/.

"Prophet: Forecasting at Scale." *Facebook Research*, research.fb.com/prophet-forecasting-at-scale/.