Comparing Machine Learning Algorithms on Time Series Data

Course Number: CS 4982

Name: Eric Foote

Department of Computer Science

University of New Brunswick

Dr. Mahanti

November 19 2021

Abstract

When comparing auto regressive integrated moving average (ARIMA); long-short term memory (LSTM) and convolutional neural networks (CNN) models applied to Black Berry stock data collected from February 3 1999 to September 27 2021; ARIMA had the smallest percent error and smallest compile time when compared to the other models by machine learning, artificial intelligence and time series analysis techniques. This was all tested using data collected from Yahoo Finance and Python 3 libraries such as tensor flow and keras to create and test these models.

Table of Contents

Table of Figures

Index of Tables

Introduction

"The techniques I developed for studying turbulence, like weather, also apply to the stock market".[31] The common techniques shared between meteorological and financial data is the field known as time series data. Which is defined as "a sequence of observations measured as successive times.'[1] In this project I set out to compare different machine learning algorithms for predicting data such as auto regressive integrated moving average (ARIMA); long-short term memory (LSTM) and convolutional neural networks (CNN) applied to Black Berry stock data collected from February 3 1999 to September 27 2021. The first thing I am going to do is outline some background information about all the different machine learning algorithms applied to the data and explain time series some more. Following that I am going to outline the methods applied and discuss the results. Finally, I am going to outline some future work that I would like to do on this project and draw some conclusions. However, I start with going through some background information.

Background Information

This section is several subsections to discuss the various parts of the theory behind the project. We start with a look at what exactly time series is and outline some ways to analyze the data.

Time Series

Time series data is "a sequence of observations measured as successive times.'[1] Basically its collecting data points over the same period for the same object of study. This time period is either “monthly, trimesteral, annual, weekly, daily, hourly, biennial, decennial.”[1] Some examples of time series data is: “weather records, economic indicators, patient health evolution, disk ops write and usage data, network log data, traces (a list of subroutine calls that an application performs during

execution”[33] are all examples of different time series data. Since this type of data covers so many fields of study, it is useful to want to analyze the data. Some of the models we can use to analyze the data are auto regressive integrated moving average (ARIMA); long-short term memory (LSTM) and convolutional neural networks (CNN). Time series analysis is “a statistical technique dealing in time series data, or trend analysis.”[2] In other words, it's using the earlier information to have insight on the future data. Time series data is broken down into three categories: time series data, cross-sectional data “data values of one or more variables, gathered at the same time point.”[2] and finally, pooled data which is “a combination of time series and cross-sectional data”[2]. An example of time series analysis is examining stock prices. We also can consider univariate and multivariate time series forecasting. Univariate time series forecasting is “if you use only the previous values of the time series to predict its future values.”[5] In other words its basically just the time series no other variables is in model creation. Multi-variariate time series forecasting is “if you use predictors other than the series to forecast.”[5] Basically, it's considering other factors in the model creation. We deal with forecasting new data using time-series analysis which “comprises the use of some significant model to forecast future conclusions by known past outcomes.” [5] Which means that we are applying machine learning algorithms to predict some new data point given the history of the data is known by the model. We are next going to look in detail on our first model ARIMA.

ARIMA

ARIMA is known as auto regressive integrated moving average and its “a class of models that ‘explains’ a given time series based on its own past values.”[5] One key trait that cannot be present in the dataset is the time series has to be “non-seasonal”. To break it down a little further “an ARIMA model is characterized by 3 terms: p,d,q where, p is the order of the AR term, q is the order of the MA term and d is the number of differencing required to make the time series stationary.”[5] In other words,

p is the order of the auto regressive part and q is the order of the moving average part and d is the number of differences to make the series stationary. Now we need to discuss what exactly the term auto regressive means it can be described as “a linear regression model that uses its own lags as predictors.”[5] Now that we have this understanding of what exactly auto regressive means we can put the two statements together to get that p “refers to the number of lags of Y to be used as predictors”[5] and q is “the number of lagged forecast errors that should go into the ARIMA model.”[5] We can state the mathematical definition of ARIMA model in words as: “Predicted Yt = Constant + Linear combination Lags of Y (upto p lags) + Linear Combination of Lagged forecast errors (upto q lags).”[5] This is as deep as I am going to go into the theory behind ARIMA models since any further exploration into this topic becomes a discussion of statistical theory and that is beyond the scope of this project. We are now going to look deeper at LSTM models.

Long Short Term Memories

Long-short term memory or LSTM in short is “an advanced version of recurrent neural network architecture that was designed to model chronological sequences and their long-range dependencies more precisely than conventional RNNs.”[16] With “the major higlights include the interior design of a basic LSTM cell.”[15] LSTM consists “of four layers that interact with one another in a way to produce the output of that cell along with the cell state. These two things are then passed onto the next hidden layer.”[15] “LSTMs comprises of three logistic sigmod gates and one hyperbolic tangent layer. Gates have been introduced to limit the information that is passed through the cell. They determine which part of the information will be needed by the next cell and which part to be discarded.”[15] In other words a LSTM network is a series of four neural network layers that is designed similarly to recurrent neural networks, each cell interacts with the four layers and are then passed on to the next layer. One of the layers applies the hyperbolic tangent function from mathematics and each cell uses the sigmoid

function three times to limit the information passed. We are now going to go to our last model type a CNN.

Convolutional Neural Network

Convolutional neural network or CNN for short is a “deep learning algorithm which can take an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.”[34] Basically, it is an algorithm that takes input learns traits about it and be able to distinguish between them. There are three layers to the CNN : the convolution layer, pooling layer, and fully connected layer. The convolutional layer “is the core building block of a CNN, and it is where a majority of computation occurs. It requires a few components, which are input data, a filter and a feature map.”[35] In other words, it is the central piece of it which has input data, some way to filter the data and a function between the layers. The pooling layer is “also known as downsampling, conducts dimensionality reduction, reducing the number of parameters in the input” [35] Basically, this layer removes samples and not necessary variables from the network. Finally, the fully-connected layer “performs the task of classification based on the features extracted through the previous layers and their different layers.”[35] It computes the classification based on the work done in convolutional layer. Now that we have some background knowledge of the theory behind the models. I am going to now go through my methods.

Methodology

The methods used in this project is to use the download historical data feature of yahoo finance [13] to get our dataset. This is an easy to use tool that allows us to get the historical data going back to the beginning of stock data collection, with the format of the file being csv which is good since its easy to load into any possible programming language that we wish to use. Furthermore, this tool gives us

nicely formatted data which gives us an easier time since we do not have to clean our dataset. The programming language that I choose for this project was Python 3, this choice was for a few reasons since it's a popular tool for data science, I figured that there would be packages and libraries readily available to make the model creation simpler. Furthermore, I also figured that there would be plenty of online resources and tutorials available to make the code easier to understand and to guide what specific packages and libraries would be necessary in order for everything to work. I used a Jupyter Notebook to cleanly divide my code up into sections such as library/package imports, rough data import, exploratory data analysis, data transformation, model creation, model forecasting and finally data analysis. This came in handy if I wanted to alter something in a previous section, it didn't require me to waste time executing other blocks of code. Refer to Appendix A for the entire code. I intentionally did not use any of the common shorthand for libraries such as "pyplot as plt" and "tensorflow as tf" to make the code easier to understand. I am going to start at the beginning of the code, the first thing was to import all possibly needed libraries and packages, this includes pandas to give us an easy way to import the data and a convenient data structure that is used across the models chosen; keras and tensorflow to give us neural networks and statsmodels to give us the statistical tests and ARIMA model. The next step was to import the raw data from the downloaded csv file obtained from yahoo finance. I then printed the dataset to look at how the data is structured, the column headers and what types of variables were recorded (See figure 2). Following this I decided to forecast the closing price of the data and first, I plotted the date vs closing price which gave us the following plot:

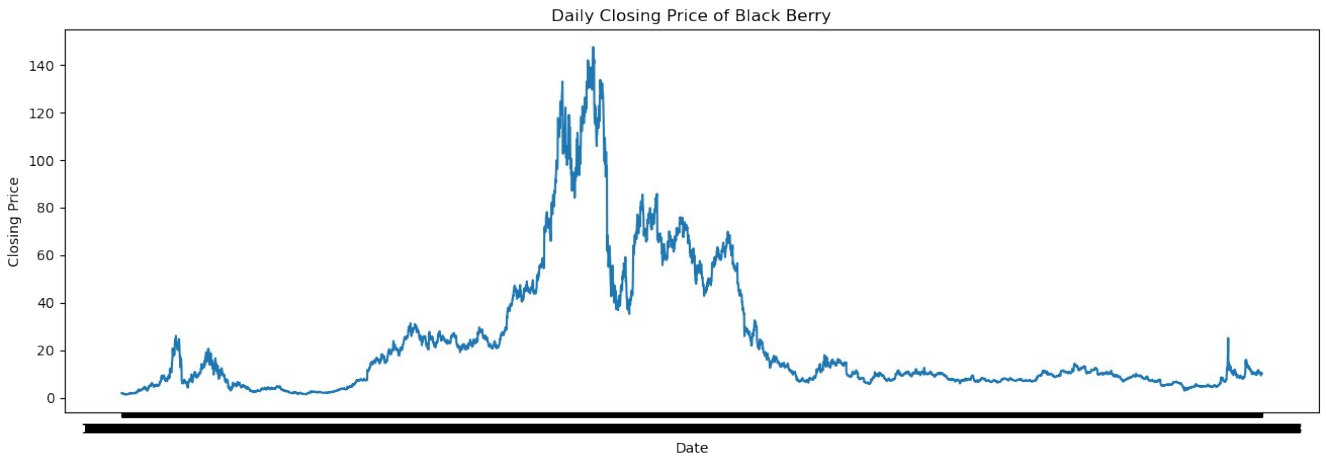

Figure 1: Daily Closing Price of BlackBerry

Then I got some basic statistics (i.e. mean and standard deviation) from the dataset which is seen below as:

Table 1: Descriptive Statistics

	Open	High	Low	Close	Adj Close	Volume
Count	5699.000000	5699.000000	5699.000000	5699.000000	5699.000000	5699.000000
Mean	23.095688	23.567537	22.573332	23.058380	23.058380	1.649957e+07
Std	27.719227	28.232380	27.102127	27.667360	27.667360	2.091388e+07
Min	1.291667	1.291667	1.140625	1.270833	1.270833	2.442000e+05
25%	7.115000	7.240000	6.990000	7.100000	7.100000	5.497250e+06
50%	10.360000	10.590000	10.180000	10.370000	10.370000	1.196540e+07
75%	25.523333	25.921666	25.073334	25.480000	25.480000	2.041865e+07
Max	146.479996	148.130005	143.889999	147.550003	147.550003	5.367394e+08

After that I tested for stationarity using ADF Test and KPSS Test. Since the p-value for the ADF test was less than 0.05 we reject the null hypothesis that the time series posses a unit root and is non-stationary. Since the p-value for the KPSS test was less than 0.05 we reject the null hypothesis that the time series is stationary around a deterministic trend. After that I de-trended the dataset by subtracting the line of best fit from the time series, which gave us a plot that looked like below:

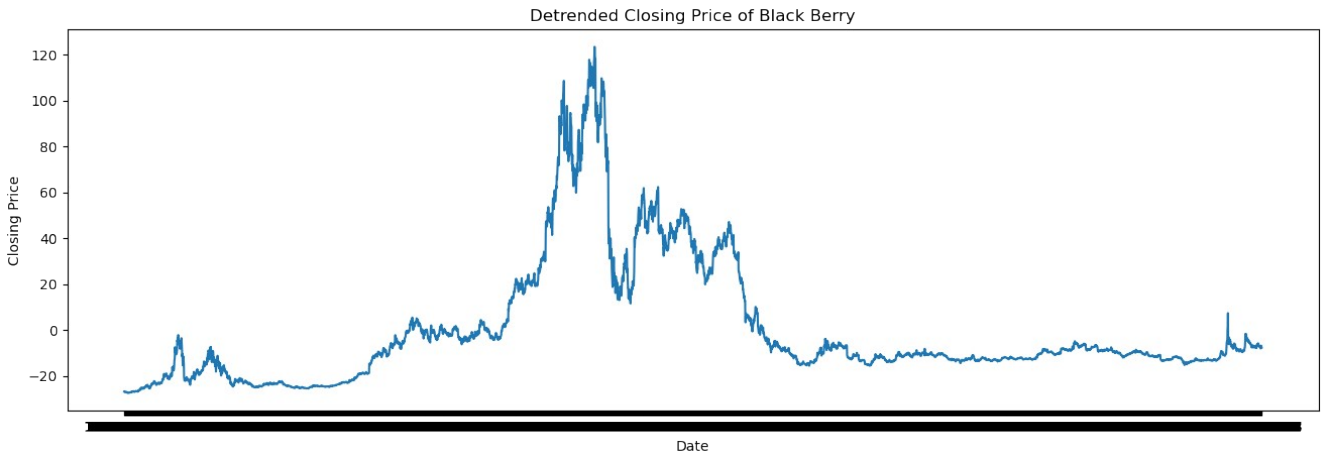

Figure 2: DeTrened BlackBerry Dataset

Following that I created the ARIMA model using the auto_arima function of the pmdarima package this is using a step wise search to minimize AIC to give the best possible model then I forecasted using the predict function of the pmdarima package to forecast the september 28 period. After that I repeated the previous step this time using the regular data set. After that I created the LSTM network by doing the following step taking the detrended closing data and making it our training set. Then I took the data set and scaled it to values between zero and one using the minimax scaler and then reshaped the dataset. Furthermore, I then created 2 arrays: x and y array that has scaled data points where x at each step has all the data points from the beginning of the for loop up to the time step which I set as 100 but could be changed. Also, y is just the singular data point at the I + time step position. Following that I then reshaped the training set to be of the form of a 3D vector to be put into the network. Then I created the LSTM network, it has 2 layers and the first layer has an input shape of a 2D vector. It was then compiled using a mean squared error loss and adam optimizer. I then fit the data to the model with 10 epochs. After that I then repeated the previous steps for the LSTM network but this time instead I used the non detrended dataset instead of the detrended one. The last model I created was the CNN. For data preperation, I followed the same steps as I did for the LSTM models. I then created almost the same network except I used Cov1D instead of LSTM. The same steps were then repeated for the non deterended dataset. Following that I finally did the the forecasting and data analysis for each model.

The first step in this was to import the new dataset which contained the September 28th datapoint. Since I worked with detrended data I also detrended this dataset to compare with the results of the detrended models. After all of that I extracted the September 28 datapoint from both the regular and the detreneded datasets and put them as seperate variables. The first model that I forecasted was the ARMIA model and it used the built in predict function with the number of periods set to 1. I then calculated the percentage error of the forecasting. To predict the LSTM model I had to transform the new dataset using minimax scaler and do a similar sequence of steps to put the test points in a vector and reshaping that vector and then sending that to the LSTM predict function. Following that I had to do the inverse minimax scaler transform to get the actual predicted values. Then I had to go to the end of the array of predictions and get the predicted value for September 28. I then calculated the Percent Differnce for the LSTM model. The same sequence of steps was applied to get the predicted September 28 for the CNN model and the percent difference. Finally I repeated all the above forecasting steps to collect the predicted values and the percent error for the non detrended dataset. Now that I have gone through step by step how I created the models and forecasted using them we can now talk about the results.

Results

The results of the forecasting using the models with a few more models with the change that this time around the data is not detrended to see if that created a difference in these 3 factors that I am testing. The results are below in tabular form:

Table 2: Results

Model	Predicted Value	% Error	Compile Time
ARIMA (detrended data)	7.25594278	4.50041861%	10.472 seconds
LSTM (detrended data)	21.898241	188.21518%	2864 seconds
CNN (detrended data)	6.1309347	19.307285%	1000 seconds
LSTM (not detrended)	37.14097	279.3766%	2975 seconds

CNN (not detrended)	11.373558	16.17526%	1000 seconds
ARIMA (not detrended)	10.13926028	3.56752077%	11.323 seconds

We see that the smallest % error is the ARIMA (not detrended) model. The fastest compile time was the ARIMA (detrended data) model. The slowest compile time was the LSTM (not detrended) model; which was also the one that was the farthest off. I am going to discuss some of these results in the discussion section.

Discussion

As we saw in the results section the model that was closest to the actual value was the ARIMA model whose input data was not detrended. This is interesting since when I ran a two statistical tests early on that their results indicated that the dataset should have been detrended. Furthermore, we see that out of the two artifical intelligence models that the CNN with detrended data had the smallest percent error and that both LSTM models are very far off from the actual value. An interesting point about the LSTM models are that they are in the range of the 75^{th} percentile to the maximum value of the entire dataset (as seen in table 1) which shows that these models would be more suitable for stock prices that stay similar. The CNN was similar to the ARIMA model where the non detrended data did better than the detrended data. I expand on this further in the recommendations section but I think that looking at different detrending methods and comparing them would possibly have some good insights. Next, we are going to draw some conclusions.

Conclusions

In conclusion, through creating 6 models: ARIMA with detrended data; ARIMA without detrended data; LSTM with detrended data; LSTM without detrended data; CNN with detrended data and CNN

without detrended data we see that the smallest % error is the ARIMA (not detrended) model. The fastest compile time was the ARIMA (detrended data) model. The slowest compile time was the LSTM (not detrended) model; which was also the one that was the farthest off. One possible result for this was the method chosen for the detrending process. The method chosen for the detrending process was subtracting the line of best fit from each data point. This was tested on just the Black Berry stock using libraries such as Keras, Tensor Flow and pmdarima. Finally, I am going to go over some recommendations for future work that I have after completing the project in its current form.

Recommendations

Some recommendations that I have for future work is to compare all of these models with bidirectional encoder representations from transformers (BERT) and generative adversarial network (GAN) models. Furthermore, to generalize the work beyond the Black Berry data set such that these models can be applied to other stock market entities. Another thing that I would do is when I removed the trend from the dataset I only did the method of subtracting the line of best fit; I could have done a few different methods such as subtracting the mean of the dataset or “Applying a filter like Baxter-King filter or the Hodrick-Prescott Filter to remove the moving average trend lines or the cyclical components.”[32] I would also take train the artifical intelligence models on smaller chunks of the data. Another thing that I would do is really look into the field of econometric to see if that particular field has some domain specific insights that would lead to different model choices, different choices for scaling data and possibly some different split percentages for training and testing sets. Finally, I would forecast beyond just the single day to see if there is some propagating error for later days.

References

1. Time Series. In: The Concise Encyclopedia of Statistics. New York, NY: Springer New York; 2008. p. 536–539.

2. Tyagi N. Introduction to Time Series Analysis in Machine learning. Analyticssteps.com. [accessed 2021 Nov 19]. https://www.analyticssteps.com/blogs/introduction-time-series-analysis-time-series-forecasting-machine-learning-methods-models

3. Ruiz P. ML approaches for time series - towards data science. Towards Data Science. 2019 May 19 [accessed 2021 Nov 19]. https://towardsdatascience.com/ml-approaches-for-time-series-4d44722e48fe

4. Cyrus. Time series forecasting with stacked machine learning models. Medium. 2019 Jul 27 [accessed 2021 Nov 19]. https://medium.com/@qs2178/time-series-forecasting-with-stacked-machine-learning-models-7250abdece0f

5. Prabhakaran S. ARIMA model - complete guide to time series forecasting in python. Machinelearningplus.com. 2021 Aug 22 [accessed 2021 Nov 19]. https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

6. Brownlee J. How to model volatility with ARCH and GARCH for time series forecasting in python. Machinelearningmastery.com. 2018 Aug 23 [accessed 2021 Nov 19]. https://machinelearningmastery.com/develop-arch-and-garch-models-for-time-series-forecasting-in-python/

7. Brownlee J. Multistep time series forecasting with LSTMs in python. Machinelearningmastery.com. 2017 May 9 [accessed 2021 Nov 19]. https://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-term-memory-networks-python/

8. Brownlee J. How to develop convolutional Neural Network models for time series forecasting. Machinelearningmastery.com. 2018 Nov 11 [accessed 2021 Nov 19]. https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/

9. Brownlee J. Multivariate time series forecasting with LSTMs in keras. Machinelearningmastery.com. 2017 Aug 13 [accessed 2021 Nov 19]. https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

10. Biswal A. An easy guide to stock price prediction using machine learning. Simplilearn.com. 2021 May 11 [accessed 2021 Nov 19]. https://www.simplilearn.com/tutorials/machine-learning-tutorial/stock-price-prediction-using-machine-learning

11. Dai Y, Zhang Y. Machine learning in stock price trend forecasting. Stanford.edu. [accessed 2021 Nov 19]. https://cs229.stanford.edu/proj2013/DaiZhang-MachineLearningInStockPriceTrendForecasting.pdf

12. Li K (yi). Predicting stock prices using machine learning. Neptune.ai. 2021 Jul 29 [accessed 2021 Nov 19]. https://neptune.ai/blog/predicting-stock-prices-using-machine-learning

13. BlackBerry Limited (BB). Yahoo.com. [accessed 2021 Nov 19]. https://finance.yahoo.com/quote/BB/history?period1=918086400&period2=1632787200&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true

14. Understanding LSTM Networks. Github.io. [accessed 2021 Nov 19]. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

15. Srivastava P. Long short term memory. Analyticsvidhya.com. 2017 Dec 10 [accessed 2021 Nov 19]. https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

16. Understanding of LSTM networks. Geeksforgeeks.org. 2020 May 10 [accessed 2021 Nov 19]. https://www.geeksforgeeks.org/understanding-of-lstm-networks/

17. Brownlee J. Time series prediction with LSTM recurrent neural networks in python with keras. Machinelearningmastery.com. 2016 Jul 20 [accessed 2021 Nov 19]. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

18. Clemente F. Synthetic time-series data: A GAN approach - towards data science. Towards Data Science. 2021 Jan 27 [accessed 2021 Nov 19]. https://towardsdatascience.com/synthetic-time-series-data-a-gan-approach-869a984f2239

19. Sonkiya P, Bajpai V, Bansal A. Stock price prediction using BERT and GAN. arXiv [q-fin.ST]. 2021. http://arxiv.org/abs/2107.09055. doi:10.1145/nnnnnnn.nnnnnnn

20. Sen J, Mehtab S. Design and analysis of robust deep learning models for stock price prediction. In: Artificial Intelligence. IntechOpen; 2021.

21. Bhardwaj K. Convolutional neural network in stock price movement prediction. Arxiv.org. [accessed 2021 Nov 19]. http://arxiv.org/abs/2106.01920v1

22. Zhong S, Hitchcock DB. S&P 500 stock price prediction using technical, fundamental and text data. arXiv [stat.ML]. 2021 [accessed 2021 Nov 19]. http://arxiv.org/abs/2108.10826

23. Time series forecasting. Tensorflow.org. [accessed 2021 Nov 19]. https://www.tensorflow.org/tutorials/structured_data/time_series

24. Forecasting: Principles and practice (2nd ed). Otexts.com. [accessed 2021 Nov 19]. https://otexts.com/fpp2/

25. Brownlee J. How to develop LSTM models for time series forecasting. Machinelearningmastery.com. 2018 Nov 13 [accessed 2021 Nov 19]. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

26. Time Series - LSTM Model. Tutorialspoint.com. [accessed 2021 Nov 19]. https://www.tutorialspoint.com/time_series/time_series_lstm_model.htm

27. Cohen I. Time series-introduction - towards data science. Towards Data Science. 2019 Jun 5 [accessed 2021 Nov 19]. https://towardsdatascience.com/time-series-introduction-7484bc25739a

28. Doron. How to import a CSV file into Python using Pandas. Datatofish.com. 2018 Feb 20 [accessed 2021 Nov 19]. https://datatofish.com/import-csv-file-python-using-pandas/

29. Pyplot tutorial — Matplotlib 3.5.0 documentation. Matplotlib.org. [accessed 2021 Nov 19]. https://matplotlib.org/stable/tutorials/introductory/pyplot.html

30. Pandas DataFrame iloc Property. W3schools.com. [accessed 2021 Nov 19]. https://www.w3schools.com/python/pandas/ref_df_iloc.asp

31. Benoit Mandelbrot - The techniques I developed for. Brainyquote.com. [accessed 2021 Nov 19]. https://www.brainyquote.com/quotes/benoit_mandelbrot_301434?src=t_stock_market

32. Prabhakaran S. Time series analysis in python - A comprehensive guide with examples - ML+. Machinelearningplus.com. 2019 Feb 13 [accessed 2021 Nov 19]. https://www.machinelearningplus.com/time-series/time-series-analysis-python/

33. What is Time Series Data? Influxdata.com. 2019 Dec 7 [accessed 2021 Nov 20]. https://www.influxdata.com/what-is-time-series-data/

34. Saha S. A comprehensive guide to convolutional neural networks — the ELI5 way. Towards Data Science. 2018 Dec 15 [accessed 2021 Nov 25]. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

35. IBM Cloud Education. What are Convolutional Neural Networks? Ibm.com. [accessed 2021 Nov 25]. https://www.ibm.com/cloud/learn/convolutional-neural-networks

Appendix A: Code

```
# A list of possible needed imports
import os
import IPython
import IPython.display
import matplotlib as matplotlib
import matplotlib.pyplot as pyplot
import numpy as numpy
import pandas as pandas
import seaborn as seaborn
import tensorflow as tensorflow
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model
import pmdarima as pmdarima
from arch.__future__ import reindexing
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers import LSTM
from keras.layers import Dropout
from sklearn.preprocessing import MinMaxScaler
from statsmodels.tsa.stattools import kpss
from scipy import signal
# Import the Raw Data
raw_data = pandas.read_csv(r'/home/knighttwisted/2021fallcoursework/CS4980/BB.csv')
print(raw_data)
# Plot the Date vs. Closing Price
```

```
pyplot.figure(figsize=(16,5), dpi = 100)
pyplot.gca().set(title = "Daily Closing Price of Black Berry", xlabel = "Date", ylabel = "Closing Price")
pyplot.plot(raw_data["Date"], raw_data["Close"])
pyplot.show()
# Get some basic statistics from the dataset
raw_data.describe()
# Test for stationarity
# ADF Test
ADF_Test_Result = adfuller(raw_data["Close"], autolag = 'AIC')
# KPSS Test
KPSS_Test_Result = kpss(raw_data["Close"], regression = 'ct')
ADF_Test_Statistic = ADF_Test_Result[0]
ADF_Test_p_value = ADF_Test_Result[1]
KPSS_Test_Statistic = KPSS_Test_Result[0]
KPSS_Test_p_value = KPSS_Test_Result[1]
print("ADF Test Statistic " + str(ADF_Test_Statistic))
print("KPSS Test Statistic " + str(KPSS_Test_Statistic))
print("ADF p-value " + str(ADF_Test_p_value))
print("KPSS p-value " + str(KPSS_Test_p_value))
# Since the p-value for the ADF test was less than 0.05 we reject the null hypothesis that the time series posses a unit root and is non-stationary
# Since the p-value for the KPSS test was less than 0.05 we reject the null hypothesis that the time series is stationary around a deterministic trend
# We need to de-trend the dataset
# I am just going to subtract the line of best fit from the time series
raw_closing_data = raw_data["Close"]
detrended_closing_data = signal.detrend(raw_closing_data)
pyplot.plot(detrended_closing_data)
pyplot.title("Detrended Closing Data")
pyplot.show()
ARIMA_Model = pmdarima.auto_arima(detrended_closing_data, start_p =1, start_q=1,
                                  test='adf',
                                  max_p =5, max_q =5,
```

```
                    m=1,
                    d=None,
                    seasonal=False,
                    start_P=0,
                    D=0,
                    trace=True,
                    error_action='ignore',
                    suppress_warnings=True,
                    stepwise=True)
print(ARIMA_Model.summary())
ARIMA_Model_Not_Detrended = pmdarima.auto_arima(raw_closing_data, start_p =1, start_q=1,
                    test='adf',
                    max_p =5, max_q =5,
                    m=1,
                    d=None,
                    seasonal=False,
                    start_P=0,
                    D=0,
                    trace=True,
                    error_action='ignore',
                    suppress_warnings=True,
                    stepwise=True)
print(ARIMA_Model.summary())
#detrended_training_data = detrended_closing_data[0:3989]
#detrended_testing_data = detrended_closing_data[3990:5699]
detrended_training_data = detrended_closing_data
# We now need to feature scale
Scaler = MinMaxScaler(feature_range = (0,1))
detrended_training_data_scaled = Scaler.fit_transform(detrended_training_data.reshape(-1,1))
timeStep = 100
x_train = []
y_train = []
```

```
for i in range(len(detrended_training_data)-timeStep-1):
    point = detrended_training_data_scaled[i:(i+timeStep),0]
    x_train.append(point)
    y_train.append(detrended_training_data_scaled[i+80,0])
x_train = numpy.array(x_train)
y_train = numpy.array(y_train)
#Reshaping
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1],1)
#Create and Fit the Network
LSTM_Model = Sequential()
LSTM_Model.add(LSTM(50,return_sequences=True, input_shape = (x_train.shape[1],1)))
LSTM_Model.add(Dropout(0.2))
LSTM_Model.add(LSTM(50))
LSTM_Model.add(Dropout(0.2))
LSTM_Model.add(Dense(units=1))
LSTM_Model.compile(loss = 'mean_squared_error', optimizer='adam')
LSTM_Model_fit = LSTM_Model.fit(x_train,y_train,batch_size=1,epochs=10)
training_data = numpy.array(raw_closing_data)
# We now need to feature scale
Scaler = MinMaxScaler(feature_range = (0,1))
training_data_scaled = Scaler.fit_transform(training_data.reshape(-1,1))
timeStep = 100
x_train = []
y_train = []
for i in range(len(training_data)-timeStep-1):
    point = training_data_scaled[i:(i+timeStep),0]
    x_train.append(point)
    y_train.append(training_data_scaled[i+80,0])
x_train = numpy.array(x_train)
y_train = numpy.array(y_train)
```

```
#Reshaping
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1],1)
#Create and Fit the Network
LSTM_Model_2 = Sequential()
LSTM_Model_2.add(LSTM(50,return_sequences=True, input_shape = (x_train.shape[1],1)))
LSTM_Model_2.add(Dropout(0.2))
LSTM_Model_2.add(LSTM(50))
LSTM_Model_2.add(Dropout(0.2))
LSTM_Model_2.add(Dense(units=1))
LSTM_Model_2.compile(loss = 'mean_squared_error', optimizer='adam')
LSTM_Model_2_fit = LSTM_Model_2.fit(x_train,y_train,batch_size=1,epochs=10)
training_data = detrended_closing_data
# We now need to feature scale
Scaler = MinMaxScaler(feature_range = (0,1))
training_data_scaled = Scaler.fit_transform(training_data.reshape(-1,1))
timeStep = 100
x_train = []
y_train = []
for i in range(len(training_data)-timeStep-1):
    point = training_data_scaled[i:(i+timeStep),0]
    x_train.append(point)
    y_train.append(training_data_scaled[i+80,0])
x_train = numpy.array(x_train)
y_train = numpy.array(y_train)
#Reshaping
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1],1)
#Create and Fit the Network
CNN_Model = Sequential()
CNN_Model.add(Conv1D(filters=64, kernel_size=2, activation ='relu',
input_shape=(x_train.shape[1],1)))
CNN_Model.add(MaxPooling1D(pool_size=2))
CNN_Model.add(Flatten())
CNN_Model.add(Dense(50, activation='relu'))
```

```
CNN_Model.add(Dense(1))
CNN_Model.compile(optimizer='adam', loss='mse')
CNN_Model.fit(x_train,y_train, epochs=1000)
training_data = numpy.array(raw_closing_data)
# We now need to feature scale
Scaler = MinMaxScaler(feature_range = (0,1))
detrended_training_data_scaled = Scaler.fit_transform(training_data.reshape(-1,1))
timeStep = 100
x_train = []
y_train = []
for i in range(len(training_data)-timeStep-1):
    point = training_data_scaled[i:(i+timeStep),0]
    x_train.append(point)
    y_train.append(training_data_scaled[i+80,0])
x_train = numpy.array(x_train)
y_train = numpy.array(y_train)
#Reshaping
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1],1)
#Create and Fit the Network
CNN_Model_2 = Sequential()
CNN_Model_2.add(Conv1D(filters=64, kernel_size=2, activation ='relu',
input_shape=(x_train.shape[1],1)))
CNN_Model_2.add(MaxPooling1D(pool_size=2))
CNN_Model_2.add(Flatten())
CNN_Model_2.add(Dense(50, activation='relu'))
CNN_Model_2.add(Dense(1))
CNN_Model_2.compile(optimizer='adam', loss='mse')
CNN_Model_2.fit(x_train,y_train, epochs=1000)
# First step is to import the data with the September 28 2021 data point included
data_including_Sept_28 =
pandas.read_csv(r'/home/knighttwisted/2021fallcoursework/CS4980/BB_Including_Sept_28.csv')
print(data_including_Sept_28)
# We need to de-trend the dataset
```

```
# I am just going to subtract the line of best fit from the time series
data_including_Sept_28_detrended = data_including_Sept_28["Close"]
data_including_Sept_28_detrended = signal.detrend(data_including_Sept_28["Close"])
pyplot.figure(figsize=(16,5), dpi = 100)
pyplot.gca().set(title = "Detrended Closing Price of Black Berry", xlabel = "Date", ylabel = "Closing Price")
pyplot.plot(data_including_Sept_28["Date"], data_including_Sept_28_detrended)
pyplot.show()
# Next up we need to extract Sept 28 from this to check the validatity of our forcasting
Sept_28 = data_including_Sept_28_detrended[5699]
Sept_28 = abs(Sept_28)
print(Sept_28)
September_28 = data_including_Sept_28["Close"][5699]
print(September_28)
# ARIMA MODEL FORECASTING AND % Error
ARMIA_Model_Sept28 = ARIMA_Model.predict(n_periods=1)
# % ERROR = (ACTUAL - THEORITICAL) / THEORITICAL * 100%
ARMIA_Model_Sept28 = abs(ARMIA_Model_Sept28)
print(ARMIA_Model_Sept28)
Percent_Difference_ARMIA_Model = (Sept_28 - ARMIA_Model_Sept28) / Sept_28 * 100
print(Percent_Difference_ARMIA_Model)
# LSTM MODEL FORECASTING AND % ERROR
#LSTM FORCASTING
detrended_testing_data_scaled = Scaler.fit_transform(data_including_Sept_28_detrended.reshape(-1,1))
x_test = []
for i in range(len(detrended_testing_data_scaled)-10-1):
    point = detrended_training_data_scaled[i:(i+10),0]
    x_test.append(point)
x_test = numpy.array(x_test)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],1)
LSTM_Model_Prediction = LSTM_Model.predict(x_test)
LSTM_Model_Prediction = Scaler.inverse_transform(LSTM_Model_Prediction)
```

```
#print(len(LSTM_Model_Prediction))
LSTM_Sept28 = LSTM_Model_Prediction[5688]
LSTM_Sept28 = abs(LSTM_Sept28)
print(LSTM_Sept28)
Percent_Difference_LSTM_Model = abs(((Sept_28 - LSTM_Sept28 )/Sept_28) * 100)
print(Percent_Difference_LSTM_Model)
#21.898241
# CNN MODEL FORECASTING AND % ERROR
# CNN FORCASTING
detrended_testing_data_scaled = Scaler.fit_transform(data_including_Sept_28_detrended.reshape(-1,1))
x_test = []
for i in range(len(detrended_testing_data_scaled)-100-1):
    point = detrended_training_data_scaled[i:(i+100),0]
    x_test.append(point)
x_test = numpy.array(x_test)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],1)
CNN_Model_Prediction = CNN_Model.predict(x_test)
CNN_Model_Prediction = Scaler.inverse_transform(CNN_Model_Prediction)
#print(len(CNN_Model_Prediction))
CNN_Sept28 = CNN_Model_Prediction[5598]
CNN_Sept28 = abs(CNN_Sept28)
print(CNN_Sept28)
Percent_Difference_CNN_Model = ((Sept_28 - CNN_Sept28 )/Sept_28) * 100
print(Percent_Difference_CNN_Model)
# LSTM_2 NOT DETRENDED MODEL FORECASTING AND % ERROR
#LSTM_2 FORCASTING
data_including_Sept_28 = numpy.array(data_including_Sept_28)
testing_data_scaled = Scaler.fit_transform(data_including_Sept_28.reshape(-1,1))
x_test = []
for i in range(len(testing_data_scaled)-10-1):
    point = training_data_scaled[i:(i+10),0]
    x_test.append(point)
```

```
x_test = numpy.array(x_test)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],1)
LSTM_2_Model_Prediction = LSTM_Model_2.predict(x_test)
LSTM_2_Model_Prediction = Scaler.inverse_transform(LSTM_2_Model_Prediction)
#print(len(LSTM_Model_Prediction))
LSTM_2_Sept28 = LSTM_2_Model_Prediction[5688]
LSTM_2_Sept28 = abs(LSTM_2_Sept28)
print(LSTM_2_Sept28)
Percent_Difference_LSTM_2_Model = abs(((September_28 - LSTM_2_Sept28 )/September_28) * 100)
print(Percent_Difference_LSTM_2_Model)
#21.898241
# CNN_2 MODEL FORECASTING AND % ERROR
# CNN_2 FORCASTING
testing_data_scaled = Scaler.fit_transform(data_including_Sept_28.reshape(-1,1))
x_test = []
for i in range(len(testing_data_scaled)-100-1):
    point = training_data_scaled[i:(i+100),0]
    x_test.append(point)
x_test = numpy.array(x_test)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],1)
CNN_Model_2_Prediction = CNN_Model_2.predict(x_test)
CNN_Model_2_Prediction = Scaler.inverse_transform(CNN_Model_2_Prediction)
#print(len(CNN_Model_Prediction))
CNN_2_Sept28 = CNN_Model_2_Prediction[5598]
CNN_2_Sept28 = abs(CNN_2_Sept28)
print(CNN_2_Sept28)
Percent_Difference_CNN_Model_2 = ((September_28 - CNN_2_Sept28 )/September_28) * 100
print(abs(Percent_Difference_CNN_Model_2))
# ARIMA MODEL FORECASTING AND % Error
ARMIA_Model_Not_Detrended_Sept28 = ARIMA_Model_Not_Detrended.predict(n_periods=1)
# % ERROR = (ACTUAL - THEORITICAL) / THEORITICAL * 100%
```

```
ARMIA_Model_Not_Detrended_Sept28 = abs(ARMIA_Model_Not_Detrended_Sept28)

print(ARMIA_Model_Not_Detrended_Sept28)

Percent_Difference_ARMIA_Model_Not_Detrended = (September_28 -
ARMIA_Model_Not_Detrended_Sept28) / September_28 * 100

print(abs(Percent_Difference_ARMIA_Model_Not_Detrended))
```

Appendix B: Data

```
In [3]: # Import the Raw Data

        raw_data = pandas.read_csv(r'/home/knighttwisted/2021fallcoursework/CS4980/BB.csv')
        print(raw_data)
```

```
            Date       Open       High        Low      Close  Adj Close  \
0     1999-02-04   2.145833   2.166667   1.895833   1.924479   1.924479
1     1999-02-05   1.929688   1.947917   1.822917   1.833333   1.833333
2     1999-02-08   1.854167   1.927083   1.783854   1.812500   1.812500
3     1999-02-09   1.822917   1.833333   1.656250   1.666667   1.666667
4     1999-02-10   1.708333   1.708333   1.604167   1.677083   1.677083
...          ...        ...        ...        ...        ...        ...
5694  2021-09-21   9.540000   9.600000   9.260000   9.370000   9.370000
5695  2021-09-22   9.500000   9.790000   9.410000   9.560000   9.560000
5696  2021-09-23  10.200000  11.050000   9.960000  10.600000  10.600000
5697  2021-09-24  10.460000  10.530000  10.140000  10.380000  10.380000
5698  2021-09-27  10.320000  10.350000   9.960000  10.140000  10.140000

        Volume
0     16788600
1      3053400
2      1548000
3      3501600
4      1597200
...        ...
5694   8769800
5695  14260500
5696  40686100
5697  10564500
5698   8993200

[5699 rows x 7 columns]
```

Figure 3: Raw Data