**Stock Price Prediction using Machine Learning**

**STAT613 Final Project**

**Shiyuan Wang & Blake Actkinson**

**Part I - Survey**

**1.1 Background**

A company’s stock price is a strong indicator of its performance.  Owning the stock of a company implies owning often a small portion of it, and thus it’s important for investors to be cognizant of that company’s performance. Predicting a company’s stock price is paramount to smart investing, but is often challenging to do successfully in practice. From a financial perspective, the stock price changes because of  supply and demand which is driven by  company reputation, company revenue, product development, new contracts, changes in company management, and many other factors that could not possibly be listed here. Additionally, there could be a large number of unknown factors that also affect the stock price. Therefore, even though investors can get all the information they would want to know about a company, they could easily be limited by what they do not know. This makes machine learning models such as artificial neural nets attractive for this application. Neural nets have the capability to process multitudes of data and identify complex relationships which would not be obvious to the human eye. While neural nets suffer from problems regarding their interpretability and have a reputation for being a black box type model, this problem becomes less important in applications where the user cares more about generating successful predictions than understanding how those predictions were generated, such as predicting stock prices. There is a large number of neural net architectures that can make predictions on the future stock price, in which the historical stock price is used as training data to forecast the future stock price. In this project, a neural net architecture used frequently in time series applications, a Long Short Term Memory Recurrent Neural Network (LSTM-RNN), will be compared to more traditional time series forecasting techniques. Potential improvements to the LSTM-RNN will then be explored and assessed.

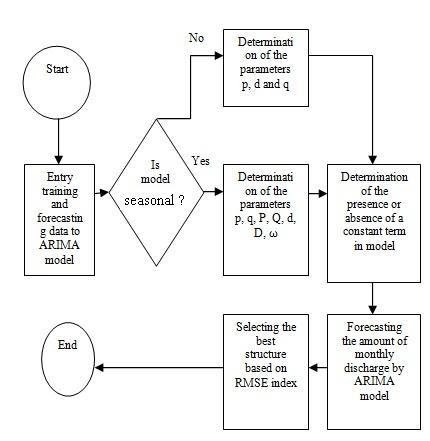
**1.2 Motivation**

Recently, stock market trading has been heavily featured in the news.  In January 2021, a short squeeze of the [stock](https://en.wikipedia.org/wiki/Stock) of the American video game retailer [GameStop](https://en.wikipedia.org/wiki/GameStop) (GME) shocked Wall Street and created an American media spectacle. The stock price of GME increased almost 30 times in 3 days because people from a Reddit forum named “wallstreetbets” coordinated trades publicly after several users pointed out that institutional investor Gabriel Plotkin had severely over leveraged his short position on the company and publicized it. Users, referring to themselves as simple primates, bought and held shares of the stock with their diamond hands in an effort to send the stock to the moon. According to this event, it is clear that a stock price is related to many seen and unforeseen factors, but it is not known how much those factors can influence the stock price. Therefore, in this project the objective is to develop a neural net architecture that can incorporate a wide variety of features to capture known and unknown relationships.

**1.3 Autoregressive Integrated Moving Average (ARIMA)**

ARIMA is a well-known model for time series forecasting. This model was originally introduced by G. E. P. Box and G. M. Jenkins in the 1970s. It is a generalized version of the autoregressive integrated moving average (ARMA) model. ARMA models can be used in forecasting, but its prediction is a linear combination of preceding values, making it problematic for non-stationary data. Under non-stationary conditions, the ARIMA model is more appropriate for forecasting. The ARIMA model is not only based on past realizations but also on past errors, and it does not require too many additional assumptions. An ARIMA model is usually expressed as ARIMA(p,d,q), where p refers to the order of the autoregressive model, d refers to the degree of differencing  and q refers to the order of moving average model. The autoregressive model part in ARIMA uses past data as input to the regression equation to forecast the next data point, the differencing part can be made to make non-stationary time series become stationary in the sense of mean, and the moving average part uses past forecast errors in a regression-like model to make the final prediction. The algorithm for an ARIMA model is shown in Figure 1. Normally, there are four general steps in constructing an ARIMA model:

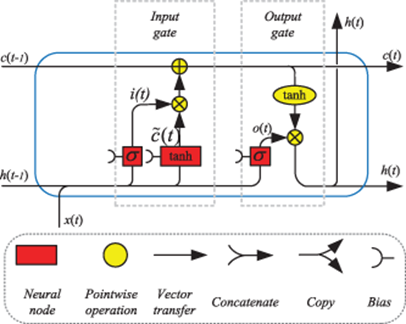
1. Identification: Given a time series, we try to incorporate relevant parameters (e.g. including parameters for seasonality).
2. Differentiation: Using different statistical tools, we select the appropriate model parameters.
3. Adjustment: We adjust model parameters to create an optimized model.
4. Prediction: We use our optimized model to create forecasts for points into the future.



**Figure 1. ARIMA model algorithm - (from Santosh Nanda)**

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

LSTM-RNNs are a popular neural net architecture widely used in the deep learning field to analyze and forecast time series data. It was created by Sepp Hochreiter and Jürgen Schmidhuber in 1995 to address problems with traditional recurrent neural networks (Hochreiter 1997). While traditional recurrent networks could make predictions on time series data that had important information in only the first few lags, it had trouble learning things based on larger lag periods. This is due to a vanishing errors problem, in which as the time lag between the current realization and past realization grew, the error would either vanish or blow up. To combat this problem, Hochreiter and Schmidhuber developed the Long Short Term Memory cell. The cell stores information from previous cells in a signal called the cell state, and has the ability to keep or toss components from the cell state through incorporating weights. The cell state signal is then combined with the current signal to produce a prediction at time t. A schematic of a standard LSTM cell, coined “vanilla LSTM,” is taken from Yu (Yu 2019) and depicted in Figure 2.



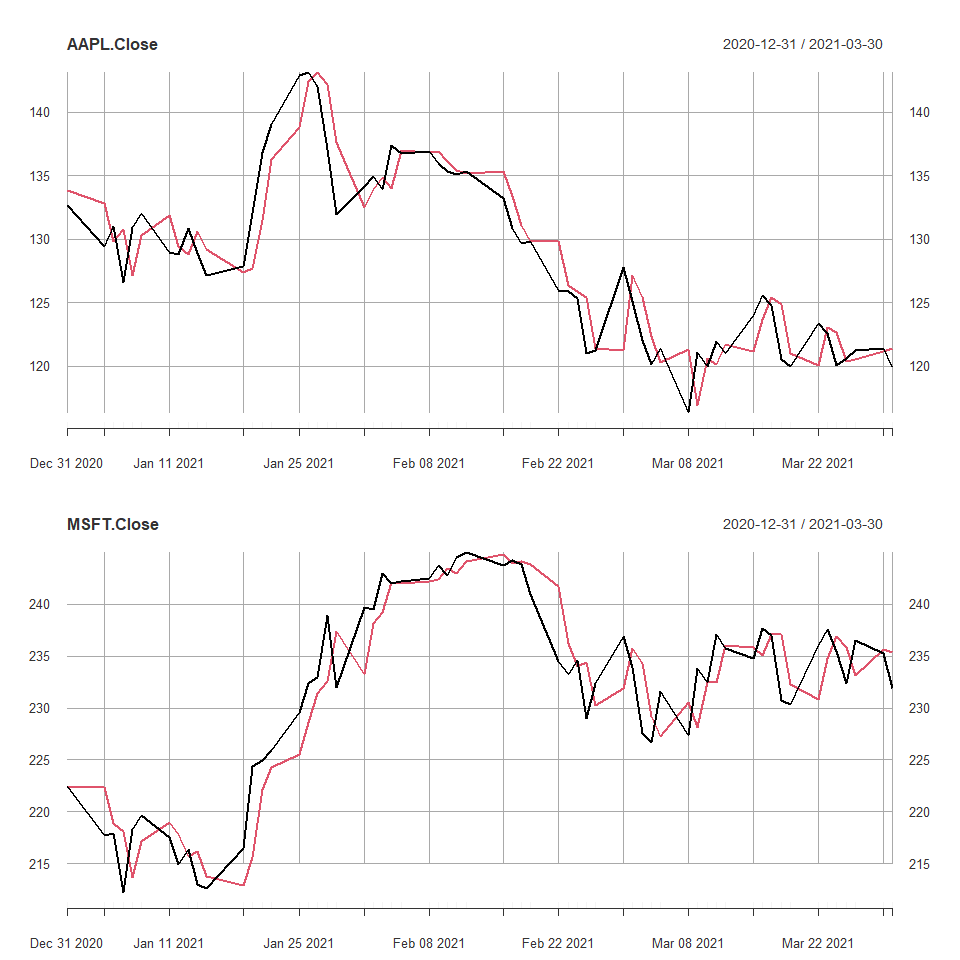
**Figure 2. LSTM-RNN algorithm**

Hochreiter, Schmidhuber, and others have made variations on the LSTM cell since its original publication. LSTM cells with a forget gate have been developed, which allows the LSTM cell to more explicitly toss features from the cell state (Gers 1999). LSTM cells that have more direct connections to the cell state via peephole connections have also been developed (Gers and Schmidhuber 2000). This list is not an exhaustive list of all the variations of LSTM cells that have been developed within the past 20 years. Suffice to say, no variation dominates any other in terms of performance (Greff 2017).

**Part II - Prediction on the Future Stock Price**

**2.1 Time Series Prediction Using ARIMA**

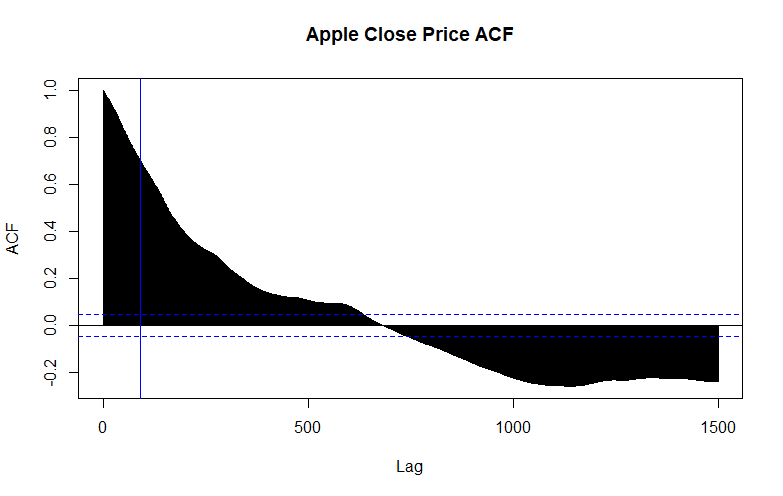
In this part, we use an ARIMA model to make predictions on the future stock price of Apple (AAPL) and Microsoft (MSFT) in a straightforward manner. Since we use historical data to forecast, we need to have a large amount of data to train the model properly. First, we split the whole dataset into 2 parts by a ratio of 29:1. We use the autocorrelation function (ACF) and partial autocorrelation function (PACF) to find the appropriate model parameters for each company (for convenience, we also use auto.arima function to get the best model parameters). Then, we use those model parameters to build an ARIMA model to forecast future stock prices. To evaluate model performance, we assess whether predictions from time t to t+1 go up or down with the true stock price. If the price prediction goes up when the true price goes up, or if the price prediction goes down when the true price goes down, we label it a True prediction. If the price prediction goes up while the true price goes down, or if the price prediction goes down while the true price goes up, we label it a False prediction. We divide the number of true predictions by the total number of predictions and multiply by 100 to obtain the trend accuracy percentage (%). We also evaluate the root mean squared error between the forecast and true stock price. Forecast results are shown in Figure 6, in which the red line represents the forecast, and the blue line represents the actual stock price.



**Figure 4. Price predictions using ARIMA.**

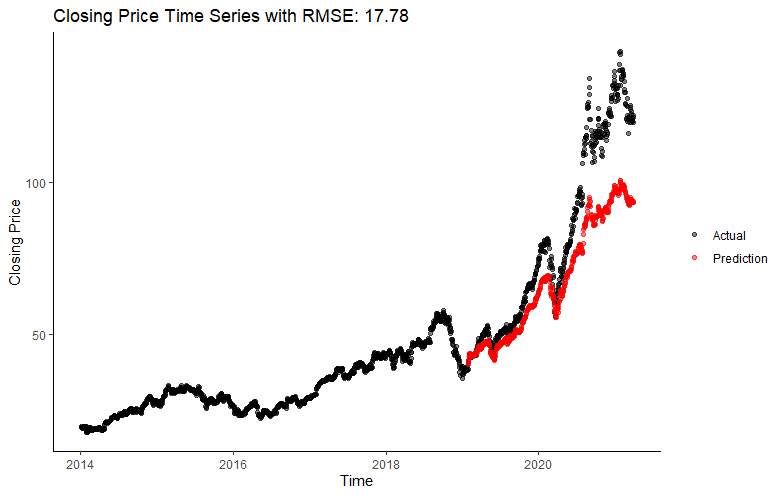
**2.2 Long Short Term Memory Recurrent Neural Network**

Like what was done in the ARIMA forecasting component of the report, an ACF plot was generated to determine what lags inform the current time point. From the ACF, we note heavy serial correlation at small lag values. For our LSTM RNN, we will select lags 1,2 and 3 to represent this heavy autocorrelation. Additionally, we also select a lag at 90 to allow the LSTM RNN to capture potential long term structure. The ACF value at lag 90 is depicted as a blue line in Figure 5.

****

**Figure 5. Apple Close Price ACF**

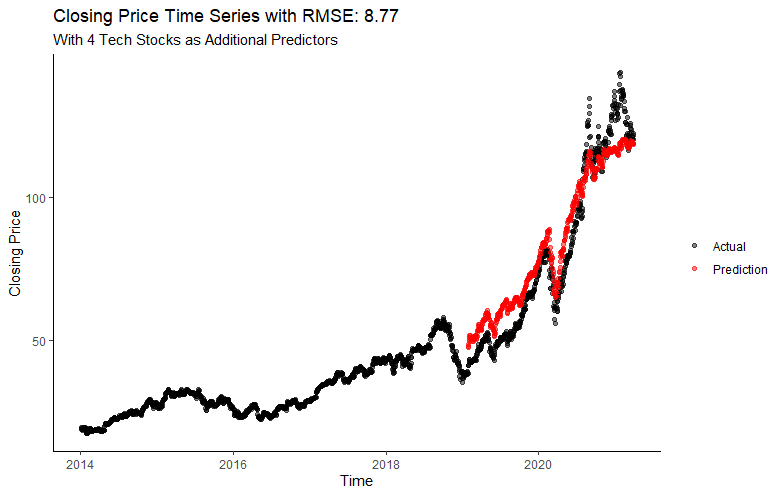
To create our LSTM RNN, we use the Keras library in R. We create an architecture that consists of two LSTM layers and one output layer. We dictate a dropout rate of 0.3 in the first layer to help prevent model overfitting. We first fit the AAPL stock price data just using AAPL stock prices alone.



**Figure 6. Closing Price Time Series with RMSE  17.78**

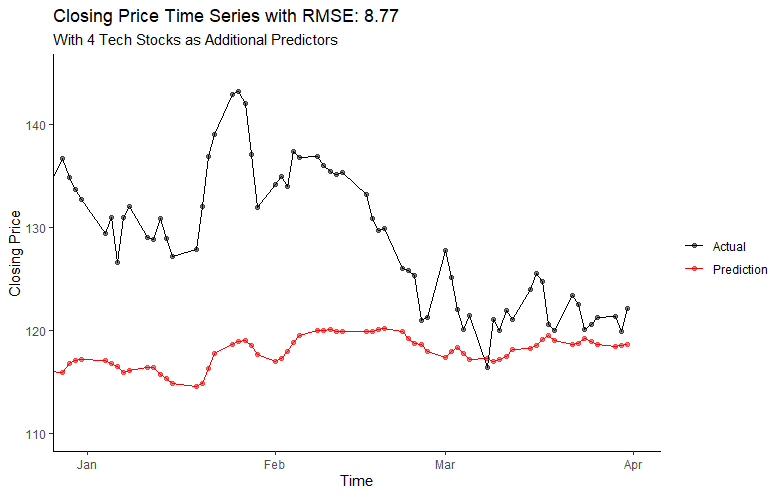
We use the model to predict stock data in the same time period that we used in ARIMA forecasting (12-31-2020 to 04-01-2021). The model decently tacks the stock price initially, but quickly diverges as the series progresses.

We hypothesized that including other stock prices in the feature space may help predict AAPL’s stock price. To test this, we incorporate 4 other tech stocks to predict AAPL’s stock price: MSFT, GOOG, FB, and AMZN. We lag each by 1, 2, 3 and 90 days and include each lagged matrix into our feature array. We evaluated the predictions using the same neural net architecture as before, ensuring everything else to be consistent aside from the changed feature space.



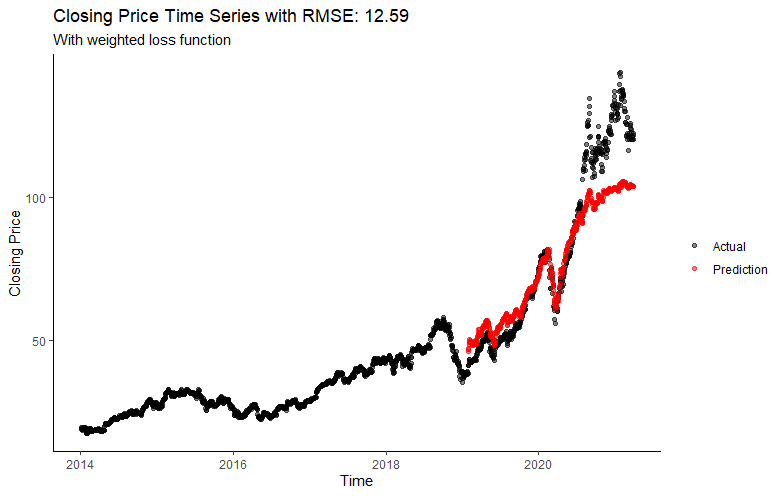
**Figure 7. Closing Price Time Series with RMSE 8.77**

Including multiple stocks seems to improve performance substantially, resulting in a 51% net reduction in RMSE. However, stock price prediction at the final points in the time series still suffers, as a zoomed-in Figure 8 nicely details.



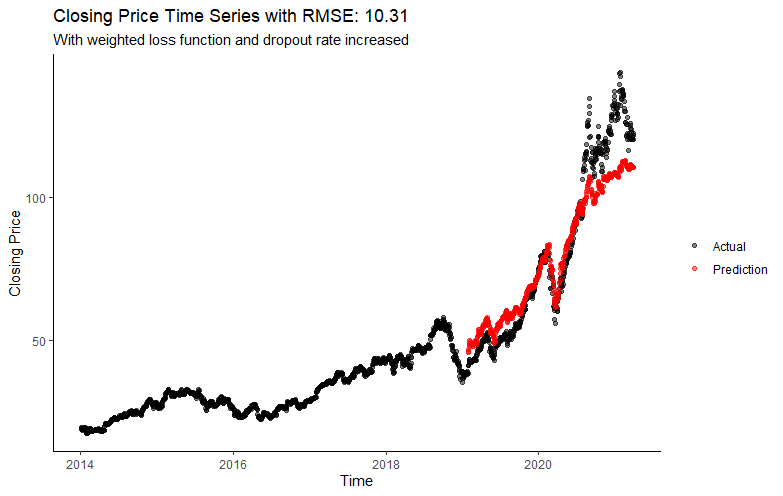
**Figure 8. Closing RMSE Time Series with RMSE 8.77 (prediction part)**

To improve performance in the latter portions of the time series, we decide to implement a loss function that weights realizations towards the end of the series more heavily than realizations towards the beginning of the series. We hypothesized that doing this weighting would enhance performance towards the tail end of the time series. Our weight vector (see code in appendix for more details - defined as K.weights) is simply the index within the batch series divided by the length of the batch series. As a concrete example, the 200th entry in a batch size of length 547 was weighted by 200/547. Results of this run are depicted in Figure 11.



**Figure 9. Closing Price Time Series with RMSE 12.59**

While the weighted model output results are variable, we note that the model with a weighted loss function generally leads to a decrease in performance. This decrease in performance suggests the model is overfit relative to its non-weighted counterpart, which is further supported by smaller returned values of the loss function. We attempt to further reduce overfitting by increasing the dropout rate of neurons in the first layer from 0.3 to 0.5 and plot the results in Figure 12.



**Figure 10. Closing Price Time Series with RMSE 10.31**

We note a slight uptick in performance. However, the weighted model performs worse than the non-weighted scheme. This suggests that employing weights to try to fix patterns in the prediction errors may exacerbate rather than ameliorate them. We summarize and compare the results of our model fitting in Table 1.

|  |  |  |
| --- | --- | --- |
| **Stock Ticker, Model** | **Accuracy Percentage(%)** | **RMSE** |
| AAPL, LSTM RNN | 45% | 17.78 |
| AAPL, ARIMA | 46% | 3.58 |
| MSFT, ARIMA | 44% | 4.87 |

**Table 1. Accuracy Percentage and RMSE**

**Conclusion**

In this report, we employ a time series modeling technique frequently used in forecasting, ARIMA, and compare it to forecasts output by a RNN-LSTM. The ARIMA model used tacks stock prices closely, but frequently forecasts stock price changes at times lagged from the actual change in price. The RNN-LSTM seems to do a better job of tracking changes in the stock price especially during the initial component of the series, but seems to suffer in performance the further out one forecasts. A weighted loss function was employed to try to fix these problems in forecasting farther out from the current time point. However, we discovered the weighted loss function did not adequately address the issue and instead exacerbated the issue. This work suggests that RNN-LSTM may be more suitable for short term forecasts, but caution should be applied for longer term forecasts.

**References**

1. Hochreiter, S. & Schmidhuber, J. Long Short-Term Memory. *Neural Computation* 9, 1735–1780 (1997).

1. Yu, Y., Si, X., Hu, C. & Zhang, J. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation* 31, 1235–1270 (2019).

1. Gers, F. A., Schmidhuber, J. & Cummins, F. Learning to forget: continual prediction with LSTM. 850–855 (1999) doi:[10.1049/cp:19991218](https://doi.org/10.1049/cp:19991218).

1. Gers, F. A. & Schmidhuber, J. Recurrent nets that time and count. in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* vol. 3 189–194 vol.3 (2000).

1. Nanda, Santosh & Tripathy, Debi & Nayak, Simanta & Mohapatra, Subhasis. (2013). Prediction of Rainfall in India using Artificial Neural Network (ANN) Models. International Journal of Intelligent Systems and Applications (IJISA). 5. 1. 10.5815/ijisa.2013.12.01.
2. Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R. & Schmidhuber, J. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems* **28**, 2222–2232 (2017).

**Appendix**

|  |  |
| --- | --- |
| rm(list=ls()) |  |
|  | #LIBRARIES |
|  | library(quantmod) |
|  | library(tidyquant) |
|  | library(tibble) |
|  | library(fs) |
|  | library(jsonlite) |
|  | library(httr) |
|  | library(XML) |
|  | library(rvest) |
|  | library(ggplot2) |
|  | library(dplyr) |
|  | library(keras) |
|  | library(rsample) |
|  | library(recipes) |
|  | library(Metrics) |
|  | library(glue) |
|  | library(tidyr) |
|  | library(tensorflow) |
|  | # FUNCTION DEFINITIONS |
|  | calc\_rmse <- function(finalized\_df){ |
|  | duplicate.locs <- duplicated(finalized\_df$date) | duplicated(finalized\_df$date,fromLast=TRUE) |
|  |  |
|  | duplicates <- finalized\_df[duplicate.locs,] |
|  |  |
|  | actuals <- duplicates %>% |
|  | group\_split(key) %>% |
|  | .[[1]] %>% |
|  | .['close'] %>% |
|  | unlist(use.names = FALSE) |
|  |  |
|  | predictions <- duplicates %>% |
|  | group\_split(key) %>% |
|  | .[[2]] %>% |
|  | .['close'] %>% |
|  | unlist(use.names = FALSE) |
|  |  |
|  | rmse\_val <- Metrics::rmse(actuals,predictions) |
|  |  |
|  | return(round(rmse\_val,2)) |
|  | } |
|  |  |
|  | create\_lstm\_tbl <- function(input.data,lag.vec,stock.name,key.type,...){ |
|  | lag.vec <- sort(lag.vec,decreasing = FALSE) |
|  |  |
|  | for (i in 1:length(lag.vec)){ |
|  | n <- lag.vec[i] |
|  | input.data <- input.data %>% |
|  | mutate("{stock.name}\_Lag\_{n}" := lag(close,lag.vec[i])) |
|  | } |
|  |  |
|  | lstm\_tbl <- input.data %>% |
|  | filter(!(is.na("Lag\_{n}"))) %>% |
|  | filter(key==key.type) %>% |
|  | {if(key.type=="Training") tail(.,...[[1]]) else .} |
|  | return(lstm\_tbl) |
|  | } |
|  |  |
|  | weighted\_mse <- function( y\_true, y\_pred ) { |
|  | K <- backend() |
|  | K$mean(K$square(K.weights\*(K$tanh(y\_true)-K$tanh(y\_pred)))) |
|  | } |
|  |  |
|  | fit\_LSTM\_to\_singlestock <- function(split,keras\_model,lag\_setting,batch\_size,train\_length,tsteps\_input, |
|  | epochs\_input){ |
|  |  |
|  | lag.setting <- lag\_setting # nrow(first.split.testing) |
|  | batch.size <- batch\_size |
|  | train.length <- train\_length |
|  | tsteps <- tsteps\_input |
|  | epochs <- epochs\_input |
|  |  |
|  | if(tsteps != length(lag.setting)){ |
|  | warning("Number of time steps not equal to number of input lags") |
|  | } |
|  |  |
|  | split.training <- split %>% |
|  | training() %>% |
|  | add\_column(key = "Training") |
|  |  |
|  | split.testing <- split %>% |
|  | testing() %>% |
|  | add\_column(key = "Testing") |
|  |  |
|  | overall.split <- bind\_rows(split.training,split.testing) |
|  |  |
|  | p1 <- ggplot(data=overall.split,aes(date,close,color=key)) + |
|  | geom\_point() + |
|  | labs(title = "Testing and Training Data First Split", |
|  | x = "Time", |
|  | y = "Closing Price (in US $)") + |
|  | theme\_classic() + |
|  | theme(legend.position = "bottom",legend.title = element\_blank()) |
|  |  |
|  | rec\_obj <- recipe(close ~ .,overall.split) %>% |
|  | step\_center(close) %>% |
|  | step\_scale(close) %>% |
|  | prep() |
|  |  |
|  | centered.data <- bake(rec\_obj,overall.split) |
|  |  |
|  | original.center <- rec\_obj$steps[[1]]$means["close"] |
|  | original.deviation <- rec\_obj$steps[[2]]$sds["close"] |
|  |  |
|  | lag\_train\_tbl <- create\_lstm\_tbl(centered.data,lag.setting,"AAPL","Training",train.length) |
|  |  |
|  | x\_train\_data <- lag\_train\_tbl %>% |
|  | select(contains("Lag")) %>% |
|  | as.matrix(nrow=nrow(x\_train\_data),ncol=ncol(x\_train\_data)) |
|  | x\_train\_arr <- array(data=x\_train\_data,dim=c(dim(x\_train\_data),1)) |
|  |  |
|  | y\_train\_vec <- lag\_train\_tbl$close |
|  | y\_train\_arr <- array(data = y\_train\_vec, dim=c(length(y\_train\_vec),1)) |
|  |  |
|  | lag\_test\_tbl <- create\_lstm\_tbl(centered.data,lag.setting,"AAPL","Testing") |
|  |  |
|  | x\_test\_data <- lag\_test\_tbl %>% |
|  | select(contains("Lag")) %>% |
|  | as.matrix(nrow=nrow(x\_test\_data),ncol=ncol(x\_test\_data)) |
|  | x\_test\_arr <- array(data=x\_test\_data,c(dim(x\_test\_data),1)) |
|  |  |
|  | y\_test\_vec <- lag\_test\_tbl$close |
|  | y\_test\_arr <- array(data=y\_test\_vec,dim=c(length(y\_test\_vec),1)) |
|  |  |
|  | for (i in 1:epochs) { |
|  | keras\_model %>% fit(x=x\_train\_arr, |
|  | y=y\_train\_arr, |
|  | batch\_size=batch.size, |
|  | epochs=1, |
|  | verbose=1, |
|  | shuffle=FALSE) |
|  |  |
|  | keras\_model %>% reset\_states() |
|  | cat("Epoch: ",i) |
|  | } |
|  |  |
|  | ## Execute predictions on the test set |
|  | closing\_pred <- keras\_model %>% |
|  | predict(x\_test\_arr, batch\_size = batch.size) %>% |
|  | .[,1] %>% |
|  | as\_tibble() %>% |
|  | mutate(s1=value\*original.deviation,.keep="unused") %>% |
|  | mutate(close = s1+original.center,.keep = "unused") %>% |
|  | cbind("date"=lag\_test\_tbl$date) %>% |
|  | add\_column(key="Prediction") |
|  |  |
|  | ## Combine predictions with original split values to create overall set. |
|  | final.data.set <- overall.split %>% |
|  | select(date,close,key) %>% |
|  | mutate(key = "Actual",.keep = "unused") %>% |
|  | bind\_rows(closing\_pred) |
|  |  |
|  | rmse\_evaluation <- calc\_rmse(final.data.set) |
|  |  |
|  | ## Calcualte the RMSE between predictions, |
|  | ## Plot the results |
|  | p2 <- ggplot(data=final.data.set,aes(date,close,color=key)) + |
|  | geom\_point(alpha = 0.5) + |
|  | scale\_color\_manual(values = c("black","red")) + |
|  | labs(title = paste0("Closing Price Time Series with RMSE: ",rmse\_evaluation), |
|  | x = "Time", |
|  | y = "Closing Price") + |
|  | theme\_classic() + |
|  | theme(legend.title = element\_blank()) |
|  |  |
|  | p2 |
|  |  |
|  | return(list(rmse\_evaluation,p1,p2)) |
|  | } |
|  |  |
|  | fit\_LSTM\_to\_multistocks <- function(split,keras\_model,lag\_setting,batch\_size,train\_length, |
|  | num\_features,tsteps\_input,epochs\_input){ |
|  |  |
|  | lag.setting <- lag\_setting # nrow(first.split.testing) |
|  | batch.size <- batch\_size |
|  | train.length <- train\_length |
|  | num.features <- num\_features |
|  | tsteps <- tsteps\_input |
|  | epochs <- epochs\_input |
|  |  |
|  | if(tsteps != length(lag.setting)){ |
|  | warning("Number of time steps not equal to number of input lags") |
|  | } |
|  |  |
|  | split.training <- split %>% |
|  | training() %>% |
|  | add\_column(key = "Training") |
|  |  |
|  | split.testing <- split %>% |
|  | testing() %>% |
|  | add\_column(key = "Testing") |
|  |  |
|  | overall.split <- bind\_rows(split.training,split.testing) |
|  |  |
|  | rec\_obj <- recipe(key ~ .,data=overall.split) %>% |
|  | update\_role(date,new\_role = "id") %>% |
|  | step\_center(all\_predictors()) %>% |
|  | step\_scale(all\_predictors()) %>% |
|  | prep() |
|  |  |
|  | centered.data <- bake(rec\_obj,overall.split) |
|  |  |
|  | original.center <- rec\_obj$steps[[1]]$means[1] |
|  | original.deviation <- rec\_obj$steps[[2]]$sds[1] |
|  |  |
|  | col\_names <- names(centered.data)[2:(num.features+1)] |
|  |  |
|  | lag\_train\_tbl <- tibble(rep(1,train.length)) |
|  |  |
|  | for(i in 1:num.features){ |
|  | current.stock <- centered.data %>% |
|  | select(1,i+1,ncol(centered.data)) %>% |
|  | rename("close"=col\_names[i]) %>% |
|  | create\_lstm\_tbl(.,lag.setting,col\_names[i],"Training",train.length) |
|  |  |
|  | lag\_train\_tbl <- cbind(lag\_train\_tbl,current.stock) |
|  | } |
|  |  |
|  | x\_train\_data <- lag\_train\_tbl %>% |
|  | select(contains("Lag")) %>% |
|  | as.matrix(nrow=nrow(x\_train\_data),ncol=ncol(x\_train\_data)) |
|  | x\_train\_arr <- array(data=x\_train\_data,dim=c(train.length,tsteps,num.features)) |
|  |  |
|  | y\_train\_vec <- lag\_train\_tbl[,3] |
|  | y\_train\_arr <- array(data = y\_train\_vec, dim=c(length(y\_train\_vec),1)) |
|  |  |
|  | lag\_test\_tbl <- tibble(rep(1,nrow(split.testing))) |
|  |  |
|  | for(i in 1:num.features){ |
|  | current.stock <- centered.data %>% |
|  | select(1,i+1,ncol(centered.data)) %>% |
|  | rename("close"=col\_names[i]) %>% |
|  | create\_lstm\_tbl(.,lag.setting,col\_names[i],"Testing") |
|  |  |
|  | lag\_test\_tbl <- cbind(lag\_test\_tbl,current.stock) |
|  | } |
|  |  |
|  | x\_test\_data <- lag\_test\_tbl %>% |
|  | select(contains("Lag")) %>% |
|  | as.matrix(nrow=nrow(x\_test\_data),ncol=ncol(x\_test\_data)) |
|  | x\_test\_arr <- array(data=x\_test\_data,c(nrow(split.testing),tsteps,num.features)) |
|  |  |
|  | y\_test\_vec <- lag\_test\_tbl[,3] |
|  | y\_test\_arr <- array(data=y\_test\_vec,dim=c(length(y\_test\_vec),1)) |
|  |  |
|  | for (i in 1:epochs) { |
|  | keras\_model %>% fit(x=x\_train\_arr, |
|  | y=y\_train\_arr, |
|  | batch\_size=batch.size, |
|  | epochs=1, |
|  | verbose=1, |
|  | shuffle=FALSE) |
|  |  |
|  | keras\_model %>% reset\_states() |
|  | cat("Epoch: ",i) |
|  | } |
|  |  |
|  | ## Execute predictions on the test set |
|  | closing\_pred <- keras\_model %>% |
|  | predict(x\_test\_arr, batch\_size = batch.size) %>% |
|  | .[,1] %>% |
|  | as\_tibble() %>% |
|  | mutate(s1=value\*original.deviation,.keep="unused") %>% |
|  | mutate(close = s1+original.center,.keep = "unused") %>% |
|  | cbind("date"=lag\_test\_tbl$date) %>% |
|  | add\_column(key="Prediction") |
|  |  |
|  | ## Combine predictions with original split values to create overall set. |
|  | final.data.set <- overall.split %>% |
|  | select(date,AAPL,key) %>% |
|  | rename(close=AAPL) %>% |
|  | mutate(key = "Actual",.keep = "unused") %>% |
|  | bind\_rows(closing\_pred) |
|  |  |
|  | rmse\_evaluation <- calc\_rmse(final.data.set) |
|  |  |
|  | ## Calcualte the RMSE between predictions, actual values. |
|  | ## Plot the results |
|  | p2 <- ggplot(data=final.data.set,aes(date,close,color=key)) + |
|  | geom\_point(alpha = 0.5) + |
|  | scale\_color\_manual(values = c("black","red")) + |
|  | labs(title = paste0("Closing Price Time Series with RMSE: ",rmse\_evaluation), |
|  | x = "Time", |
|  | y = "Closing Price") + |
|  | theme\_classic() + |
|  | theme(legend.title = element\_blank()) |
|  |  |
|  | return(list(rmse\_evaluation,p2)) |
|  | } |
|  | ts.predict <- function(data){ |
|  | # Compute the returns for the stock |
|  | stock = data |
|  | stock = stock[!is.na(stock)] |
|  | breakpoint = floor(nrow(stock)\*(2.9/3)) |
|  |  |
|  | # Initialzing a dataframe for the forecasted return series |
|  | model.parameter <- arimaorder(auto.arima(stock, lambda = "auto")) |
|  | p <- as.numeric(model.parameter[1]) |
|  | d <- as.numeric(model.parameter[2]) |
|  | q <- as.numeric(model.parameter[3]) |
|  | forecasted\_series <- c() |
|  | for (b in breakpoint:(nrow(stock)-1)) { |
|  | stock\_train = stock[1:b,] |
|  |  |
|  | # Summary of the ARIMA model using the determined (p,d,q) parameters |
|  | fit = arima(stock\_train, order = c(p, d, q),include.mean=TRUE, optim.control = list(maxit = 1000)) |
|  | arima.forecast = forecast(fit, h = 1,level=99) |
|  | # Creating a series of forecasted returns for the forecasted period |
|  | forecasted\_series = c(forecasted\_series,arima.forecast$mean[1]) |
|  | } |
|  | y <- stock[(floor(nrow(stock)\*(2.9/3))+1):nrow(stock),] |
|  | g <- cbind(y,forecasted\_series) |
|  | p1 <- plot(g, main = paste(names(data))) |
|  | for (i in 1:(nrow(g)-1)){ |
|  | g$difference.actual[i] <- as.numeric(g[,1][i+1])-as.numeric(g[,1][i]) |
|  | g$difference.predict[i] <- as.numeric(g[,2][i+1])-as.numeric(g[,2][i]) |
|  | } |
|  | tf <- c() |
|  | for (i in 1:nrow(g)){ |
|  | if (sign(g$difference.actual[i]) == sign(g$difference.predict[i])){ |
|  | tf[i] = 1 |
|  | } else{ |
|  | tf[i] = 0 |
|  | } |
|  | } |
|  | Accuracy.percentage <- sum(tf)/length(tf)\*100 |
|  | rmse.value <- rmse(g$difference.actual, g$difference.predict) |
|  | return(list(p1,Accuracy.percentage,rmse.value)) |
|  | } |
|  | #Final project |
|  | ##Extracting S&P500 data from the packages |
|  | stock\_list\_tbl <- tq\_index("SP500") %>% |
|  | select(symbol, company, weight) %>% |
|  | arrange(desc(weight)) |
|  |  |
|  | symbol <- stock\_list\_tbl$symbol[c(1:5)] |
|  |  |
|  | stock\_data <- tq\_get(symbol, get = "stock.prices", |
|  | from = "2014-01-01",to= "2021-04-01") |
|  |  |
|  | stock\_data <- stock\_data %>% |
|  | select(symbol, date, close) |
|  |  |
|  | ## Isolate the apple stock closing price |
|  | aapl.data <- stock\_data %>% |
|  | group\_split(symbol) %>% |
|  | .[[1]] |
|  |  |
|  | spread\_data <- stock\_data %>% |
|  | spread(.,symbol,close) |
|  |  |
|  | acf(aapl.data$close,lag.max=1500,main="Apple Close Price ACF") |
|  | abline(v=90,col="blue") |
|  |  |
|  | ## Fitting model to entirety of AAPL time series data |
|  |  |
|  | training.period <- round(0.70\*nrow(aapl.data),0) |
|  | testing.period <- nrow(aapl.data)-training.period |
|  |  |
|  | aapl.partitioning <- rolling\_origin( |
|  | aapl.data, |
|  | initial = training.period, |
|  | assess = testing.period, |
|  | ) |
|  |  |
|  | ## LSTM parameters |
|  | lag.setting <- c(1,2,3,90) # nrow(first.split.testing) |
|  | batch.size <- testing.period # both testing.period/batch.size and train.length/batch.size whole nums |
|  | train.length <- testing.period\*2 |
|  | tsteps <- 4 |
|  | epochs <- 200 |
|  |  |
|  | new.model <- keras\_model\_sequential() |
|  |  |
|  | new.model %>% |
|  | layer\_lstm(units = 50, |
|  | input\_shape = c(tsteps,1), |
|  | batch\_size = batch.size, |
|  | return\_sequences = TRUE, |
|  | stateful = TRUE) %>% |
|  | layer\_dropout(0.3) %>% |
|  | layer\_lstm(units = 50, |
|  | return\_sequences = FALSE, |
|  | stateful = TRUE) %>% |
|  | layer\_dense(units = 1) |
|  |  |
|  | new.model %>% |
|  | compile(loss = 'mse',optimizer = 'adam') |
|  |  |
|  | aapl.output <- fit\_LSTM\_to\_singlestock(aapl.partitioning$splits[[1]], |
|  | new.model, |
|  | lag.setting, |
|  | batch.size, |
|  | train.length, |
|  | tsteps, |
|  | epochs) |
|  |  |
|  | aapl.output[[3]] |
|  |  |
|  | ## Incorporating multiple features |
|  | all.splits <- rolling\_origin( |
|  | spread\_data, |
|  | initial = training.period, |
|  | assess = testing.period, |
|  | ) |
|  |  |
|  | num.features <- ncol(spread\_data)-1 |
|  |  |
|  | multi.model <- keras\_model\_sequential() |
|  |  |
|  | multi.model %>% |
|  | layer\_lstm(units = 50, |
|  | input\_shape = c(tsteps,num.features), |
|  | batch\_size = batch.size, |
|  | return\_sequences = TRUE, |
|  | stateful = TRUE) %>% |
|  | layer\_dropout(0.3) %>% |
|  | layer\_lstm(units = 50, |
|  | return\_sequences = FALSE, |
|  | stateful = TRUE) %>% |
|  | layer\_dense(units = 1) |
|  |  |
|  | multi.model %>% |
|  | compile(loss = 'mse',optimizer = 'adam') |
|  |  |
|  | multi.model.output <- fit\_LSTM\_to\_multistocks(all.splits$splits[[1]], |
|  | multi.model, |
|  | lag.setting, |
|  | batch.size, |
|  | train.length, |
|  | num.features, |
|  | tsteps, |
|  | epochs) |
|  |  |
|  | multi.model.output[[2]] + |
|  | labs(subtitle = "With 4 Tech Stocks as Additional Predictors") |
|  | # |
|  | multi.model.output[[2]] + |
|  | geom\_line() + |
|  | coord\_cartesian(xlim=c(date("2020-12-31"),date("2021-04-01")),ylim=c(110,145)) |
|  |  |
|  | weighted.model <- keras\_model\_sequential() |
|  |  |
|  | K.weights <- tf$Variable(c(1:547/547),tf$float32) |
|  |  |
|  | weighted.model %>% |
|  | layer\_lstm(units = 50, |
|  | input\_shape = c(tsteps,num.features), |
|  | batch\_size = batch.size, |
|  | activation = "tanh", |
|  | return\_sequences = TRUE, |
|  | stateful = TRUE) %>% |
|  | layer\_dropout(0.5) %>% |
|  | layer\_lstm(units = 50, |
|  | activation = "tanh", |
|  | return\_sequences = FALSE, |
|  | stateful = TRUE) %>% |
|  | layer\_dense(units = 1) |
|  |  |
|  | weighted.model %>% |
|  | compile(loss = weighted\_mse,optimizer = 'adam') |
|  |  |
|  | weighted.output <- fit\_LSTM\_to\_multistocks(all.splits$splits[[1]], |
|  | weighted.model, |
|  | lag.setting, |
|  | batch.size, |
|  | train.length, |
|  | num.features, |
|  | tsteps, |
|  | epochs) |
|  |  |
|  | weighted.output[[2]] + |
|  | labs(subtitle = "With weighted loss function and dropout rate increased") |
|  |  |
|  | ### calculate Accuracy Percentage |
|  | # a <- output[[4]] |
|  | # b <- a[a$key == "Actual",] |
|  | # c <- a[a$key == "Prediction",] |
|  | # colnames(c)[2] <- "Prediction.close" |
|  | # d <- merge(x= b,y = c,by = "date", all.x = TRUE) |
|  | # e <- d[!is.na(d$Prediction.close),] |
|  | # g <- cbind(e$close,e$Prediction.close) |
|  | # g <- as.data.frame(g) |
|  | # for (i in 1:(nrow(g)-1)){ |
|  | #     g$difference.actual[i] <- as.numeric(g[,1][i+1])-as.numeric(g[,1][i]) |
|  | #     g$difference.predict[i] <- as.numeric(g[,2][i+1])-as.numeric(g[,2][i]) |
|  | # } |
|  | # tf <- c() |
|  | # for (i in 1:nrow(g)){ |
|  | #     if (sign(g$difference.actual[i]) == sign(g$difference.predict[i])){ |
|  | #       tf[i] = 1 |
|  | #     } else{ |
|  | #       tf[i] = 0 |
|  | #     } |
|  | #   } |
|  | # Accuracy.percentage <- sum(tf)/length(tf)\*100 |
|  |  |
|  |  |
|  | ###Time series prediction |
|  | getSymbols('AAPL', from = "2014-01-01",to= "2021-03-31") |
|  | getSymbols('MSFT', from = "2014-01-01",to= "2021-03-31") |
|  | getSymbols('AMZN', from = "2014-01-01",to= "2021-03-31") |
|  | getSymbols('FB', from = "2014-01-01",to= "2021-03-31") |
|  | getSymbols('GOOG', from = "2014-01-01",to= "2021-03-31") |
|  | aapl <- AAPL[,4] |
|  | msft <- MSFT[,4] |
|  | amzn <- AMZN[,4] |
|  | fb <- FB[,4] |
|  | goog <- GOOG[,4] |
|  |  |
|  | a <- ts.predict(aapl) |
|  | b <- ts.predict(msft) |
|  | c <- ts.predict(amzn) |
|  | d <- ts.predict(fb) |
|  | e <- ts.predict(goog) |
|  |  |
|  | par(mfrow=c(2,1)) |
|  | a[[1]] |
|  | b[[1]] |