Assignment 4: Algorithmic Trading

Liam Devlin - bwb18180

An algorithmic trading (also known as automated trading) system is a computer program which automatically makes buying / selling orders, based on a predefined set of rules. The task for this assignment was to produce an algorithmic trading solution, and compare the profit made by the system with other approaches.

1. Background of the problem

1.1 Basic Overview

Algorithmic trading is the practice of employing computer programs to automate trading. Since its early adoption in the 1980's, it has become so commonplace that in one day of wall street trades, more than 90% of trades were performed by trading algorithms (Isidore, 2018). Approaches to algorithmic trading vary, with some systems aimed at Arbitrage (exploiting differences in stock markets), Trade execution, which optimises how many assets should be bought and when, and by far the most common - and the method being used for this assignment - Investment Analysis. In investment analysis, the prices of stocks are analysed over time with performance indicators such as Moving Average Covariance Divergence (MACD), Relative Strength Index (RSI) and Directional Movement Index (DMI), which when analysed and processed can be used as part of a system to make decisions about whether the asset should be bought, and the quantity which should be bought.

1.2 Examples of other systems

One system proposed by (Skabar & Cloete, 2002), looks at using a neural network to predict the open and close price of an asset, whilst using a GA to optimise the weights and biases within the network, with the intention of increasing the accuracy of predictions. This system was able to make a significant profit over a 4 year period (25-30%). Another system by (Worasucheep et. al, 2017) proposes the use of Particle Swarm Optimisation in combination with a plethora of indicators (MACD, DMI, RSI, STO, CCI) to influence the decision to buy / sell. This system performed extremely well and in the space of just a month, was able to pull together an average profit of 2.4%.

1.3 The Proposed System

Having reviewed the pre-existing systems which aim to solve the problem of algorithmic trading, the decision was made to develop a new system. This new approach will make use of a neural network to predict the open and close price of a single chosen asset. Using these predictions, the MACD and RSI will be calculated for the open price, and finally, the trading of stocks will take place at the close of trade

using the recorded close price for that asset. A GA will be employed to optimise parameters of the trading, such as the influence of RSI / MACD on the decision to buy, the minimum markup when selling, as well as the decision boundary for both selling and buying. The results of this system will be compared against a random solution, which randomly selects when to buy or sell, as well as another solution which simply uses the Moving Average to make decisions about buying / selling stocks.

1.4 Choice of Asset

For asset selection, the decision was made to use NVIDIA stock from the period of June 2018 to September 2019. This decision was made as NVIDIA is a relatively consistent asset, which has a few dips and spikes which will test the accuracy of the system. As can be seen from figure 1.1.1 the price does rise slightly over time, however, the early massive valley in price may confuse the system. A volatile dataset is a good way to evaluate the robustness of a system, and assets with more standard movement may even benefit from improved returns. Having said this, the overall rise in price is beneficial as even a poor system would be able to pull together some profit. This shifts the priority away from purely making profit into maximising profit.

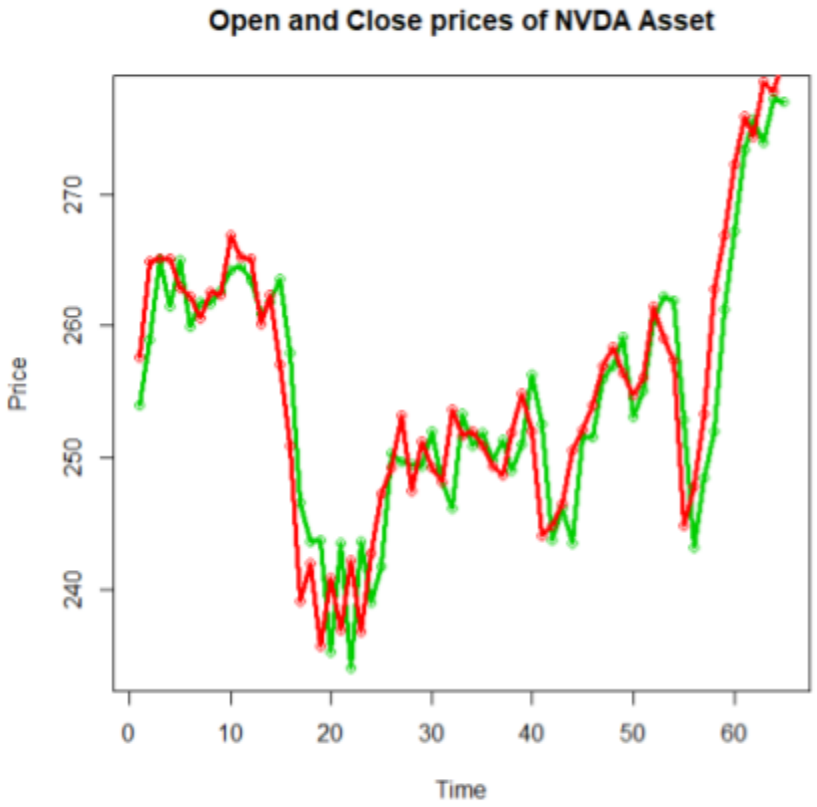

Figure 1.1.1 Open and Close prices for the NVDA stock over the period of March to June, 2018

2. Implementation

As the system is composed of multiple components, the first step is to create a main file to manage all these separate parts. This main file also controls the parameters for each component, e.g. the epochs during the neural network predictions, or the population size during the GA stage.

2.1 Neural Network for predictions

The first step in the system is to make predictions about the open and close price of the chosen asset (NVIDIA).

```
myStocks<-c("NVDA")
getSymbols(myStocks, src="yahoo", from="2018-03-01", to="2018-09-01")

# Get values we are interested in
nvda_close <- coredata(NVDA$NVDA.Close)[,1]
nvda_close_l1 <- Lag(nvda_close, lag_01)
nvda_close_l2 <- Lag(nvda_close, lag_02)
nvda_close_l3 <- Lag(nvda_close, lag_03)
nvda_close_l4 <- Lag(nvda_close, lag_04)
nvda_open <- coredata(NVDA$NVDA.Open)[,1]
nvda_open_l1 <- Lag(nvda_open, lag_01)
nvda_open_l2 <- Lag(nvda_open, lag_02)
nvda_open_l3 <- Lag(nvda_open, lag_03)
nvda_open_l4 <- Lag(nvda_open, lag_04)
```

Figure 2.1.1 Getting the stock values

As can be seen from Figure 2.1.1 , the stocks are collected over a period of 6 months in 2018. Once the stock object has been collected, we collect the open and close prices and create 4 lags for each, with the lag values defined in the main file. Lagging the data allows the neural network to better understand the way the value is moving. Once the lags have been created, we next combine them into a single data-frame object as can be seen in Figure 2.1.2. At this stage we also need to scale the data; scaling is crucial for neural networks as they are unable to produce meaningful results with large numbers due to the activation functions inside the neural networks usually being in the range -1 - 1. Once the data has been scaled, we extract the scaling parameters so that after the predictions have been made by the neural network, we can rescale them. Finally, we create a train / test split and convert the data frame to a time-series object. The next step is to tell the neural network what we are trying to predict. We do this by devising a formula, with the first part of the formula denoting the values we are trying to predict, and the second part the values that can influence the desired output.

For example, in Figure 2.1.3 we are trying to predict the *a* and *b* values (open and close) based on all the other data in the time-series frame. Renaming was performed to keep the formula simple and easy to read.
Once the formula has been created, we are ready to begin training the neural network; we do this in R using the neuralnet package, passing through the training data and the parameters held in the main file (epochs, learning rate). Finally, we collect the results and re-scale them for use in the trading algorithms.

```
# Create dataframe and add lags
nvda_lagged <- data.frame(nvda_close, nvda_open,
                          nvda_close_l1, nvda_close_l2, nvda_close_l3, nvda_close_l4,
                          nvda_open_l1, nvda_open_l2, nvda_open_l3, nvda_open_l4)

nvda_lagged <- nvda_lagged[complete.cases(nvda_lagged),]
scaled_nvda <- scale(nvda_lagged, scale=TRUE, center = TRUE)
scale_parameters <- attributes(scaled_nvda)[3:4]

# Reformat the data for training
colnames(scaled_nvda) <- paste(c('a','b','c','d','e','f','g','h','i','j'))
rownames(scaled_nvda) <- NULL

# Test Train Split
n_instances <- length(scaled_nvda[,1])
test_split_index <- as.integer((n_instances / 100) * train_test_percentage)
train_data <- as.ts(scaled_nvda[1:test_split_index-1,])
test_data <- as.ts(scaled_nvda[test_split_index:n_instances,])
```

Figure 2.1.2 Creating the lagged data-set

```
# Neural net for open / close prices
f <- as.formula("a + b ~ c + d + e + f + g + h + i + j")
current_nn <- neuralnet(f, data=train_data,  hidden=nn_layers, threshold = threshold, stepmax=stepmax)

# current nn train results
nn_train_predictions <- compute(current_nn, train_data)
nn_train_results = nn_train_predictions$net.result

# current nn test results
nn_test_predictions <- compute(current_nn, test_data)
nn_test_results = nn_test_predictions$net.result
```

Figure 2.1.3 Creating the neural network and making predictions

2.2 GA Optimisation and Trading

Next, we will detail the development of the GA trading solution. This algorithm uses the MACD and RSI of the open price to influence the decision to either buy or sell an asset. Firstly, we collect the recorded price at close of the asset for the test period, as this is the simulated time at which trading takes place.

The decision was made to trade on only the test data, thus giving a more accurate representation of how the system would perform in real world conditions. The MACD is calculated using the recorded open values for the training period, after which the neural network predictions are used.
The same method is applied to the calculation of the RSI.

```
getGaResults <- function(x){

recorded_close <- final_return[[6]]
# MACD with recorded values for training segment and predicted values for test
# If above 0 sell signal, if less than 0, buy signal
macd_open <- MACD(append(final_return[[1]], final_return[[7]]), nFast = 3, nSlow = 7, nSig = 3)[test_split_index:n_instances]
# Again, used the recorded values for training segment of the data and predictions for test
# Rsi, greater than 70 is a sell signal, less than 30 is a buy signal
rsi_open <- RSI(append(final_return[[1]], final_return[[7]]), n = 3)[test_split_index:n_instances]
```

Figure 2.2.1 GA initialisation

The individual for the GA takes the form of a vector of 9 real weights; these control various parameters of the GA such as the influence of RSI vs MACD on the decision to buy, as well as at what point we should buy, and how much.

As can be seen from Figure 2.2.1, we also create a temporary budget variable, number of stocks held variable and a variable which holds on to the price at which we last bought an asset. Next, we define the fitness function for the system, which will iterate through our RSI and MACD vectors (simulating passing days) and make decisions at the end of each simulated day as to whether the asset will be bought, sold or held.

```
predicted_fitness <- function(x){
  weights <- normalize(x[1:2])
  budget<-init_budget
  stocks_held <- 0
  last_stock_buy_price <- 0

  macd_open_weight <- weights[1]
  rsi_open_weight <- weights[2]


  markup <- x[3]
  buyCutoff <- x[4]
  sellCutoff <- x[5]
  rsiBuyCutoff <- x[6] * 100
  rsiSellCutoff <- x[7] * 100
  macdBuyCutoff <- x[8]
  macdSellCutoff <- x[9]
```

Figure 2.2.2 Fitness Function initialisation

From figure 2.2.3, we first assess whether the RSI and MACD values fall into the ranges for buying and selling identified by the GA. The decision threshold is rather self explanatory storing the current decision value, which will always fall in the range -1 to 1. If the decision threshold is over 0.25, we will buy some amount of the asset, and if it is less than -0.25, we will sell some amount of the asset.

For each metric, we multiply the decision threshold by how far our current MACD/RSI value goes over the identified threshold.

For example, given the RSI, a value over 70 indicates a buy signal; the system will then look at how far the current RSI value is over 70, normalize this to the range 0-1 with 0 being RSI == 70 and 1 being RSI == 100. We then use this number as the increment to the decision threshold (which is also multiplied by the GA weights).

```
# For loop simulates each day of trading
for(i in 1:length(macd_open)){
  # if greater than 0.5 buy, if less than -0.5 sell
  decision_threshold <- 0
  # macd open
  if(macd_open[i] >= macdBuyCutoff){
    strength_multiplier <- macd_open[i] / max(macd_open)
    decision_threshold <- decision_threshold - (1 * strength_multiplier) * macd_open_weight
  }
  else if(macd_open[i] <= -(macdSellCutoff)){
    strength_multiplier <- abs(macd_open[i]) / abs(min(macd_open))
    decision_threshold <- decision_threshold + (1 * strength_multiplier) * macd_open_weight
  }

  # rsi open
  if(rsi_open[i] >= rsiBuyCutoff){
    strength_multiplier <- (rsi_open[i] - rsiBuyCutoff) / (100 - rsiBuyCutoff)
    decision_threshold <- decision_threshold - (1 * strength_multiplier) * rsi_open_weight
  }
  else if(rsi_open[i] <= rsiSellCutoff){
    strength_multiplier <- (rsiSellCutoff-rsi_open[i]) / (rsiSellCutoff)
    decision_threshold <- decision_threshold + (1 * strength_multiplier) * rsi_open_weight
  }
```

Figure 2.2.3 Making Decision

Once the decision has been calculated, it is processed and the buying / selling / holding of assets is performed for that day.

```
  # Make decision
  if(decision_threshold >= buyCutoff){
    # buy signal, get the actual price
    currentAssetPrice <- recorded_close[i]

    if(budget > currentAssetPrice){
      # Make number of assets bought proportional to the signal strength
      buyMultiplier <- ((decision_threshold - 0.25) / 75) * 100
      maxAssets <- as.integer(budget / currentAssetPrice)
      assetsToBuy <- as.integer(maxAssets * buyMultiplier)
      # buy the stocks
      stocks_held <- stocks_held + assetsToBuy
      budget <- budget - (currentAssetPrice * assetsToBuy)
      last_stock_buy_price <- currentAssetPrice
    }
  }
  else if(decision_threshold <= -(sellCutoff))
  {
    if(stocks_held > 0){
      currentAssetPrice <- recorded_close[i]
      # does the current asset price meet the markup requirement?
      if(currentAssetPrice > last_stock_buy_price  * (1 + markup)){
        # Sell signal, again, proportional to strength of the signal
        sellMultiplier <- ((abs(decision_threshold) - 0.25) / 75) * 100
        assetsToSell <- as.integer(stocks_held * sellMultiplier)
        budget <- budget + (currentAssetPrice * assetsToSell)
        stocks_held <- stocks_held - assetsToSell
      }
    }
  }
}
# any assets we still have at the end of the training period
dump <- stocks_held * recorded_close[length(recorded_close)]
return(budget + dump)
```

Figure 2.2.4 Trading based on decision

If a decision is made to buy (e.g. the decision threshold passes 0.25), we first collect the recorded price of the asset at close and use this as the price at which we buy assets. We next quickly check that we can afford at least one asset, after which we calculate how many assets to purchase. This is calculated by remapping our decision threshold in the range 0.25-1 into 0-1, indicating how strong the buy signal is. We then calculate the maximum number of assets we can currently buy with our current budget and multiply these two numbers together, giving us the final number of assets to buy. Finally, we add these assets to our stocks_held variable and remove the price of these assets from our budget.

For selling, we first make sure that we have at least one asset to sell, then, we check that the price for which we are selling is greater than the price at which we last bought a stock, plus some markup. The same technique as was used with buying is used to establish how many stocks we currently plan to sell, after which the stocks are removed and the current price multiplied by the stocks sold is added to the budget.

Once the GA has finished iterating through the RSI and MACD vectors, the remaining stocks are sold regardless and added to the budget. The budget is then returned as a fitness score.

2.3 Moving Average Solution

For the sake of comparison, a moving average solution was implemented. This system looks only at the moving average, if the current predicted asset price falls below the MA line, the asset is bought; if it moves over the line, the asset is sold.

```
getMovingAverageResult <- function(x){
recorded_close_test <- final_return[[6]]
predicted_close_test <- final_return[[8]]

ma_close <- SMA(append(final_return[[4]], final_return[[8]]), n=7)[test_split_index:n_instances]

budget <- init_budget
stocks_held <- 0

for(i in 1:length(predicted_close_test)){
  currentAssetPrice <- recorded_close_test[i]

  if(predicted_close_test[i] < ma_close[i]){
    # buy
    assetsToBuy <- as.integer(budget / currentAssetPrice)
    stocks_held <- stocks_held + assetsToBuy
    budget <- budget - (currentAssetPrice * assetsToBuy)
  }
  else{
    # sell
    budget <- budget + (currentAssetPrice * stocks_held)
    stocks_held <- 0
  }
}

dump <- stocks_held * recorded_close_test[length(recorded_close_test)]
final_budget <- budget + dump
final_profit <- ((final_budget - init_budget) / init_budget) * 100
return(final_profit)
}
```

Figure 2.3.1 Moving Average Solution

Figure 2.3.1 shows the implementation. First, we collect the predicted and recorded close price for the trading period, next we then calculate the moving average for the predicted close price. As with the GA solution, we create a budget and stocks held variable and begin to iterate through the training period. If our predicted value falls below the predicted moving average, we buy as many assets as our budget allows. If the predicted price rises above the moving average, we sell all of the available assets. Like the GA solution, any remaining assets at the end of the training period are sold and the final profit is calculated and returned.

2.4 Random Solution

The random solution is near identical to the moving average solution. We simply predict the price of the asset over the training period, and randomly decide each day if we are going to trade.

```r
getRandomResult <- function(x){

recorded_close_test <- final_return[[6]]
predicted_close_test <- final_return[[8]]

budget <- init_budget
stocks_held <- 0

for(i in 1:length(predicted_close_test)){
  currentAssetPrice <- recorded_close_test[i]
  randomChoice <- runif(1, -1, 1)

  if(randomChoice > 0.25){
    # buy
    assetsToBuy <- as.integer(budget / currentAssetPrice)
    stocks_held <- stocks_held + assetsToBuy
    budget <- budget - (currentAssetPrice * assetsToBuy)
  }
  else if (randomChoice < -0.25) {
    # sell
    budget <- budget + (currentAssetPrice * stocks_held)
    stocks_held <- 0
  }
}

dump <- stocks_held * recorded_close_test[length(recorded_close_test)]
final_budget <- budget + dump
final_profit <- ((final_budget - init_budget) / init_budget) * 100
return(final_profit)
}
```

Figure 2.4.1 Random Solution

As with the GA and MA solution, we get our predictions and recorded values and make a variable to store the current budget and stocks held. Each day, we create a randomChoice variable, which holds a number between -1 and 1. If the choice goes above 0.25, we buy as many assets as our budget allows; if the choice falls below -0.25, we sell all of our held assets. The profit is then returned in the same way as the Moving Average solution.

3. Results

3.1 GA System Results

Overall, the results (shown in Figure 3.1.1) of the system are quite positive, providing a good profit and a tangible benefit to both the moving average solution, as well as the random solution. The strength of the random solution is quite surprising, but its performance is almost doubled by the GA + NN system. In some instances of the random solution, the return was near or just above the GA system, and as such, a higher profit is clearly attainable. To ascertain why the system is not able to achieve slightly higher returns, the predictions of the GA will first be analysed.

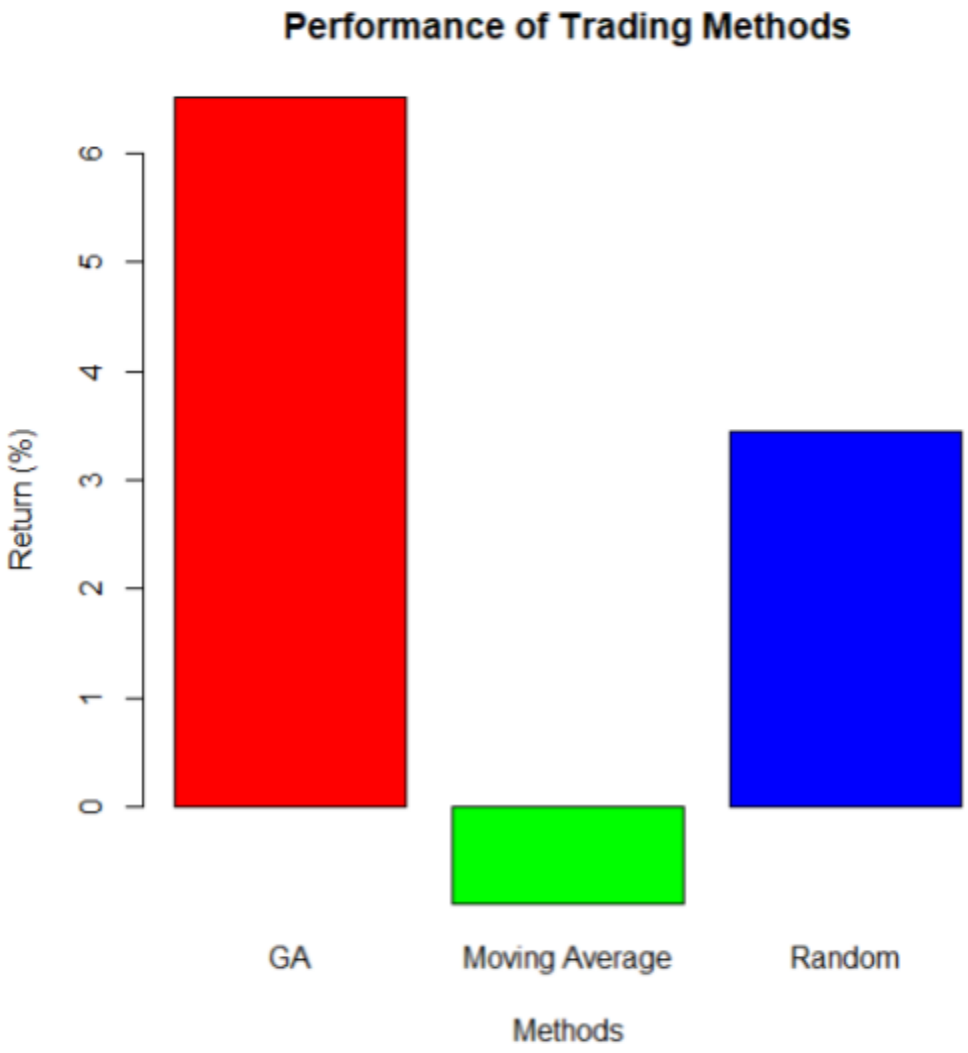

Figure 3.1.1 Performance of trading algorithms

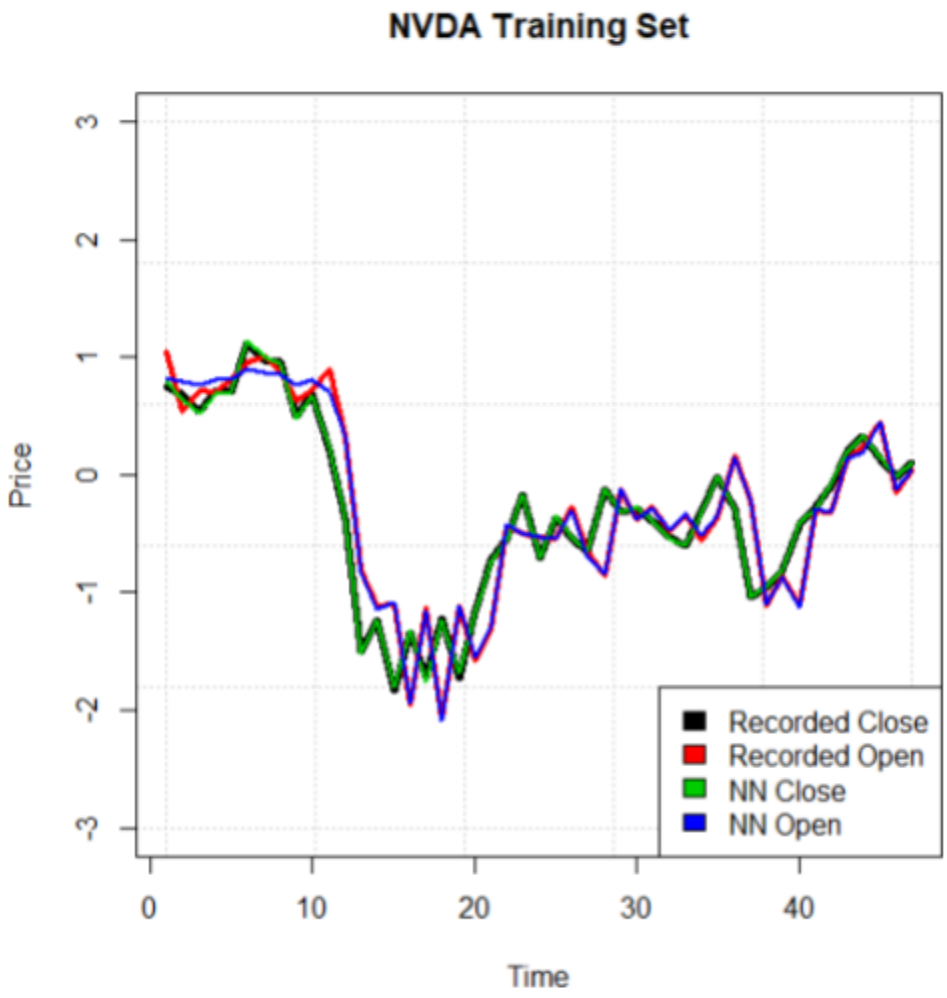

Figure 3.1.2 Neural Network Training Accuracy

As can be seen from figure 3.1.2, the training set accuracy is fairly good, though it does appear to be overfitting to the data. This may have a negative influence on the test set, and thus, the trading period, as overfitting tends to lead to inaccuracies, and sometimes massive divergences in the test period.

Looking at figure 3.1.3, we can see that this was somewhat true for the test period predictions. As time passes, the systems accuracy does drop, but it should be noted that the first half of the predictions at least follow the same general trend. It is difficult to say why these inaccuracies occurred, however some likely steps that could be taken to improve the accuracy would be to add additional lag values, thus increasing the window size, as well as perhaps training the system on a longer period of data. With this being said, various window sizes were experimented with, with some looking as far back as 2 weeks; the larger the window size, the greater the inaccuracies became.

Another possible approach would be to employ the use of a Recurrent Neural Network, due to their inherent benefits when dealing with time-series data.

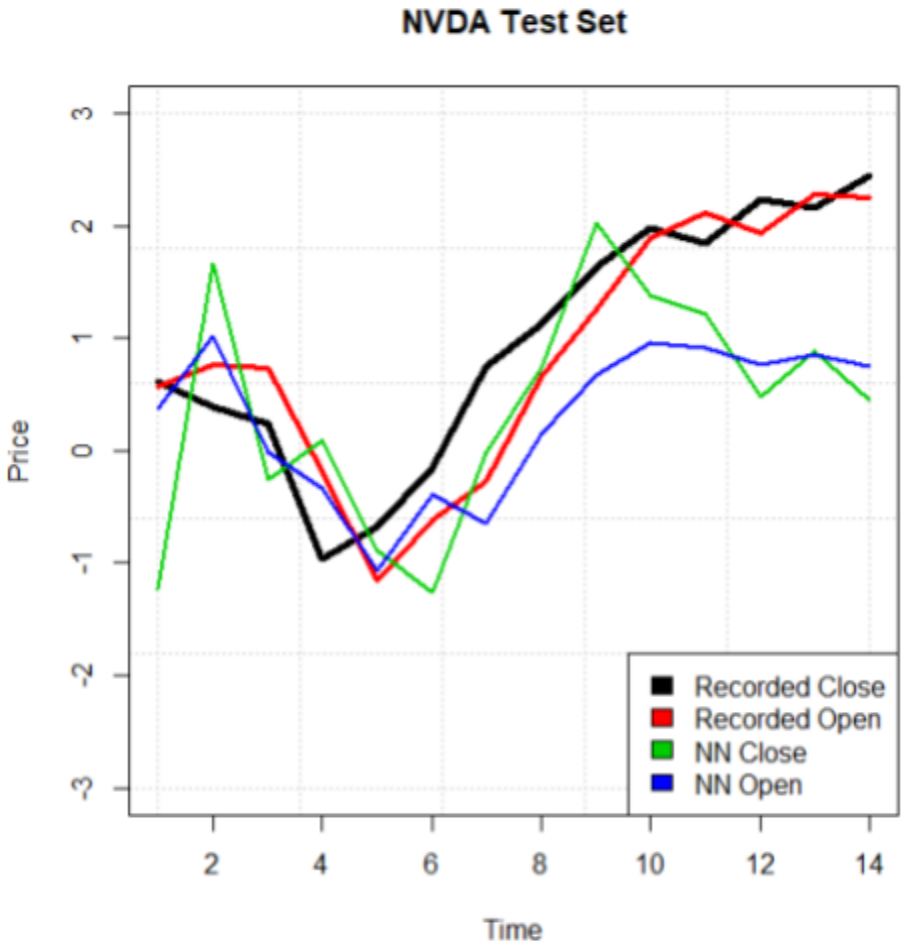

Figure 3.1.3 Neural Network Test Accuracy

4. Appendices

Isidore, Chris. “Machines Are Driving Wall Street’s Wild Ride, Not Humans.” *CNNMoney*, 6 Feb. 2018,

money.cnn.com/2018/02/06/investing/wall-street-computers-program-trading/index.htm

Skabar, Andrew, and Ian Cloete. *Neural Networks, Financial Trading and the Efficient Markets Hypothesis*. 2002, crpit.com/confpapers/CRPITV4Skabar.pdf.

Worasucheep, Chukiat, et al. *An Automatic Stock Trading System Using Particle Swarm Optimization*. June 2017, doi:10.1109.