Can an Artificial Neural Network Model Increase Stock Market Returns?

Fernando Arreguin

Western Governors University

Data Analytics Graduate Capstone – C772

Dr. William Sewell

In the futures market, a trader can take on two positions. In a “long position,” an investor can buy a commodity at a specific price, then sell the commodity back after a given time. In a “short position,” the trader sells at a time and price with the obligation of buying it in after a certain period (Williams, 2011). With the enormous amount of historical data and the speed at which the market reacts, data mining might give a competitive edge over human stock performance analysis. Financial advisers state maintaining a diverse portfolio of assets is the safest approach to investing. The long-term objective is to maximize profit by buying and selling various assets (Mullaney, 2009). This project aims to answer: Can Random Forest and Artificial Neural Network (ANN) Predict Stock Market Returns?

- H_0: Random Forest and Artificial Neural Network (ANN) can predict stock market returns.
- H_a: Random Forest and Artificial Neural Network (ANN) can predict stock market returns.

This project is based on a case study by Luis Torgo (Torgo, 2010). This project will use Torgo’s tools and functions and update some of the methods used in his case study.

Since there is potential for profit in the price of an index going up and down, ANN will generate a set of action triggers. Action triggers have three positions, “buy,” “sell,” “hold.” If an index’s price is forecasted to go up *p%* in *k* days, the ANN will trigger a buy signal. If the price is forecasted to go down *p%* in *k* day, then the ANN will trigger a sell signal. If the price is not predicted to go up or down *p%* in *k* days, then the ANN will triggrt a hold signal. The ANN will also ignore any triggers if there is a *p%* change in *k* days while already in an open position.

Collecting historical data of the Nasdaq Composite (^IXIC)

Historical time series data is readily available online. This project will use the Nasdaq Composite Index's (^IXIC) historical information on Yahoo Finance (Yahoo Finance).

Yahoo Finance offers options to download many stock information ranging from their launch to the present day. This information is freely available as a CSV file.

Rstudio has a package that reads CVS files and turns them into objects. The constructor function `xts()` (Ulrich n.d) creates an object of class `xts`. Since the data downloaded from Yahoo Finance is time series based, an `xts` object is an optimal choice. The ability to turn a column into an index makes the `xts()` especially suitable for time-series data. The time-series data will end on 2021-02-17, the date this data was downloaded.

The imported data show there are 12,612 records and six columns," Open," "High," "Low," "Close," "Adj.Close," and "Volume."

```
> #### Reading .CSV ####
> IXIC <- as.xts(read.csv.zoo('IXIC.csv',header=T))
> dim(IXIC)
[1] 12619     6
> head(IXIC)
             Open   High    Low  Close Adj.Close Volume
1971-02-05 100.00 100.00 100.00 100.00    100.00      0
1971-02-08 100.84 100.84 100.84 100.84    100.84      0
1971-02-09 100.76 100.76 100.76 100.76    100.76      0
1971-02-10 100.69 100.69 100.69 100.69    100.69      0
1971-02-11 101.45 101.45 101.45 101.45    101.45      0
1971-02-12 102.05 102.05 102.05 102.05    102.05      0
```

The column needs to be renamed so that the delimiter "." in "Adj.Close" does not interfere with processing.

```
> names(IXIC) <- c("Open","High","Low","Close","Adjusted","Volume")
> head(IXIC)
             Open   High    Low  Close Adjusted Volume
1971-02-05 100.00 100.00 100.00 100.00   100.00      0
1971-02-08 100.84 100.84 100.84 100.84   100.84      0
1971-02-09 100.76 100.76 100.76 100.76   100.76      0
1971-02-10 100.69 100.69 100.69 100.69   100.69      0
1971-02-11 101.45 101.45 101.45 101.45   101.45      0
1971-02-12 102.05 102.05 102.05 102.05   102.05      0
```

The “Volume” Column is missing information at the beginning of the data.

```
> head(IXIC)
             Open   High    Low  Close Adjusted Volume
1971-02-05 100.00 100.00 100.00 100.00   100.00      0
1971-02-08 100.84 100.84 100.84 100.84   100.84      0
1971-02-09 100.76 100.76 100.76 100.76   100.76      0
1971-02-10 100.69 100.69 100.69 100.69   100.69      0
1971-02-11 101.45 101.45 101.45 101.45   101.45      0
1971-02-12 102.05 102.05 102.05 102.05   102.05      0
> tail(IXIC)
               Open     High      Low    Close Adjusted      Volume
2021-02-09 13966.82 14044.95 13966.55 14007.70 14007.70  8658980000
2021-02-10 14093.35 14109.12 13845.47 13972.53 13972.53 10165550000
2021-02-11 14045.21 14058.91 13916.85 14025.77 14025.77 10000200000
2021-02-12 13979.21 14102.04 13937.71 14095.47 14095.47  7352960000
2021-02-16 14152.21 14175.12 13995.45 14047.50 14047.50  7646300000
2021-02-17 13911.65 13976.43 13804.26 13965.49 13965.49  7227140000
```

The “Volume” data does not begin until 1984-02-11.

```
> IXIC["1984-10-09/1984-10-12"]
             Open   High    Low  Close Adjusted   Volume
1984-10-09 244.09 244.09 244.09 244.09   244.09        0
1984-10-10 243.29 243.29 243.29 243.29   243.29        0
1984-10-11 244.20 244.70 244.20 244.70   244.70 62860000
1984-10-12 245.50 246.20 245.50 246.20   246.20 58860000
>
```

Since some technical indicators require volume information, the records with the missing “Volume” information will be trimmed not to create any bias and unnecessary processing. The dataset now has 9,162 records and six columns.

```
> IXIC <- as.xts(IXIC["1984-10-11/"])
> dim(IXIC)
[1] 9162    6
```

The `quantmod` library (Ryan 2019) provides a `candlechart()` to view the data. A chart was generated showing the latest 12 months of the IXIC dataset (Wiley, 2009).

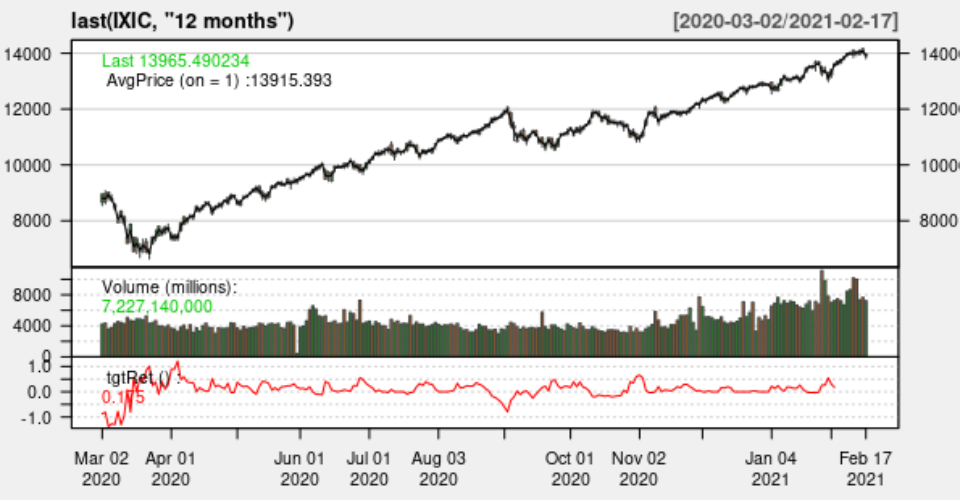
Random Forest to Analyze Technical Indicators

Training strategies involve predicting the volume of a stock in the future. For a stock purchase to be profitable, its price needs to increase or decrease a certain p% over the predicted time. The tendency for an indicator to increase or decrease must be read as a single value. If the goal is to predict the variation above or below p%, then indicator "T" it the absolute value of that variation. An example of the "T" indicator is if there is a tendency for today's average price to increase 4% in 10 days, then the "T" value is 1.4%. If there is a tendency for the average price to go decrease 4% in 10 days, then the "T" indicator is also 1.4%.

Torgo's case study provides a function to generate this indicator.

```
> #### K-Days T indicator ####
> T.ind <- function(quotes,tgt.margin=0.025,n.days=10) {
+   v <- apply(HLC(quotes),1,mean)
+
+   r <- matrix(NA,ncol=n.days,nrow=NROW(quotes))
+   for(x in 1:n.days) r[,x] <- Next(Delt(Cl(quotes),v,k=x),x)
+
+   x <- apply(r,1,function(x) sum(x[x > tgt.margin | x < -tgt.margin]))
+   if (is.xts(quotes)) xts(x,time(quotes)) else x
+ }
```

The following chart shows the value of the "T" indicator over the last 12 months in relation to the IXIC candle chart.



"T" will be the target variable. Technical indicators will be used to determine the value of "T". The R package `TTR` provides technical indicators for this analysis. The technical indicators in this project will be (FT Press Delivers collections, 2011):

- The Average True Range which is an indicator of volatility of the series.
- The Stochastic Momentum Index indicates movement.
- The Welles Wilder natural movement index.
- The Arun indicator tries to identify the start of a trend.
- The Bollinger Band that compares whether prices are high or low on a relative basis.
- The Chaikin Volatility which compares the spread between a security's high and low prices.
- The Close Location Value that relates to the session Close.
- The Arms's Ease of Movement identifies the ease of movement of a security.
- The MACD oscillator gauges the momentum of price changes.
- The Money Flow Index.

- The Parabolic Stop-and-Rever tracks the trend as it develops.
- The Volatility indicator.
- Williams Accumulation / Distribution, which is a measure of market movement.

A Random Forest can be used to estimate the importance of variables involved in prediction. Random Forests decides the importance in a nearly bias-less method (Fischetti, 2016). The `randomforest()` function can determine which technical indicators have the most influents on predicting price change. The Random Forest calculates the order of importance by plotting them with the `varImpPlot()`. Once the most important features have been identified, the `specifymodel()` function creates a `quantmod` object with the technical indicators to generate the ANN model. The packages for the technical indicators are turned into functions. Creating function streamlines future analysis of different stocks.

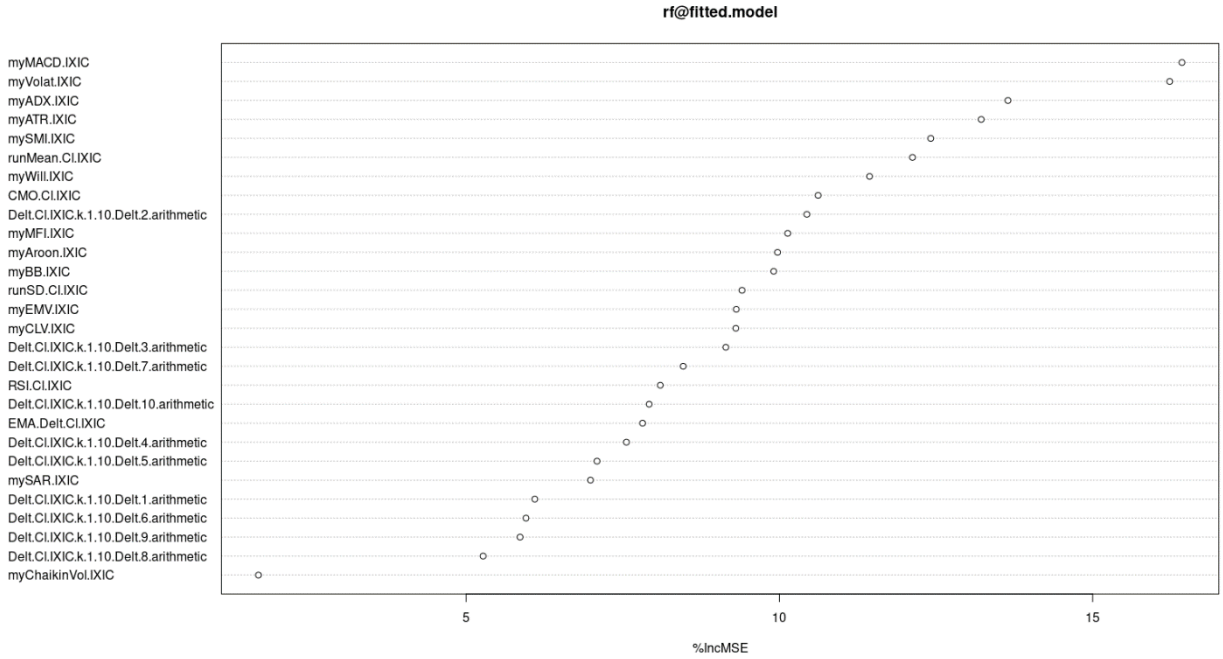
```
> ### Using Random Forest to Choose Optimal Technical Indicators ####
> myATR <- function(x) ATR(HLC(x))[,'atr']
> mySMI <- function(x) SMI(HLC(x))[,'SMI']
> myADX <- function(x) ADX(HLC(x))[,'ADX']
> myAroon <- function(x) aroon(x[,c('High','Low')])$oscillator
> myBB <- function(x) BBands(HLC(x))[,'pctB']
> myChaikinVol <- function(x) Delt(chaikinVolatility(x[,c("High","Low")]))[,1]
> myCLV <- function(x) EMA(CLV(HLC(x)))[,1]
> myEMV <- function(x) EMV(x[,c('High','Low')],x[,'Volume'])[,2]
> myMACD <- function(x) MACD(Cl(x))[,2]
> myMFI <- function(x) MFI(x[,c("High","Low","Close")], x[,"Volume"])
> mySAR <- function(x) SAR(x[,c('High','Close')]) [,1]
> myVolat <- function(x) volatility(OHLC(x),calc="garman")[,1]
> myWill  <- function(x) williamsAD(HLC(x))
> myRSI <- function(x) RSI(Cl(x))[,1]
```

Placing the model into an object generates a warning but was still able to execute.

```
> library(randomForest)
> rfImp.model <- specifyModel(T.ind(IXIC) ~ Delt(Cl(IXIC),k=1:10) +
+                               myWill(IXIC) + myATR(IXIC) + mySMI(IXIC) + myADX(IXIC) + myAroon(IXIC) +
+                               myBB(IXIC)  + myChaikinVol(IXIC) + myCLV(IXIC) +
+                               CMO(Cl(IXIC)) + EMA(Delt(Cl(IXIC))) + myEMV(IXIC) +
+                               myVolat(IXIC)  + myMACD(IXIC) + myMFI(IXIC) + RSI(Cl(IXIC)) +
+                               mySAR(IXIC) + runMean(Cl(IXIC)) + runSD(Cl(IXIC)))
Warning message:
Using formula(x) is deprecated when x is a character vector of length > 1.
  Consider formula(paste(x, collapse = " ")) instead.
```

Once the random forest model was built. `VarImpPlot()` functions displayed the technical indicators in order of percent increasing mean squared error. A higher %IncMSE signifies more influence on the target variable (Fischetti, 2016).

```
> set.seed(1234)
> rf <- buildModel(rfImp.model,method='randomForest',
+                  training.per=c(start(IXIC),index(IXIC["2019-02-08"])),
+                  ntree=50, importance=T)
>
> varImpPlot(rf@fitted.model,type=1)
> imp <- importance(rf@fitted.model,type=1)
> rownames(imp)[which(imp > 10)]
 [1] "Delt.Cl.IXIC.k.1.10.Delt.2.arithmetic" "myWill.IXIC"                       "myATR.IXIC"
 [4] "mySMI.IXIC"                            "myADX.IXIC"                        "CMO.Cl.IXIC"
 [7] "myVolat.IXIC"                          "myMACD.IXIC"                       "myMFI.IXIC"
[10] "runMean.Cl.IXIC"
```



The top 8 influential technical indicators were chosen to analyze the model. 8 is an arbitrary number for the ANN model. The process gave a warning again but was still able to execute.

```
> #### Updated Model ####
> rfImp.model <- specifyModel(T.ind(IXIC) ~
+                                 myADX(IXIC) +
+                                 myATR(IXIC) +
+                                 myVolat(IXIC) +
+                                 myMACD(IXIC) +
+                                 mySMI(IXIC) +
+                                 CMO(Cl(IXIC)) +
+                                 myWill(IXIC) +
+                                 runMean(Cl(IXIC)) )
Warning message:
Using formula(x) is deprecated when x is a character vector of length > 1.
  Consider formula(paste(x, collapse = " ")) instead.
>
> Tform <- as.formula('T.ind.IXIC ~ .')
```

The data set will split into a training set, a validation set, and a test set. The training set will generate the model, the validation set will validate the model, and the test set generates the signals, which will determine the signals for decision making.

```
> #### Splitting Data####
> Tdata.train <- as.data.frame(modelData(rfImp.model,
+                                        data.window=c('1984-10-11','2017-02-10')))
>
> Tdata.val <- na.omit(as.data.frame(modelData(rfImp.model,
+                                              data.window=c('2017-02-11','2019-02-07'))))
>
> Tdata.test <- na.omit(as.data.frame(modelData(rfImp.model,
+                                              data.window=c('2019-02-08','2021-02-02'))))
```

Generating the ANN Model

A feedforward neural network is most effective in forecasting financial time series data (Atsalakis 2009). The neural network consists of several nodes connected by links; networks usually have an input, a hidden, and an output layer. In the first layer, the number of nodes consists of the model's explanatory variables. Since the model has eight explanatory variables, the neural network will have eight nodes on the first layer. Every neuron receives information from the previous layer and is transformed to the input of the next layer. Neural networks are sensitive to the data's variance, so the dataset must be normalized. The `scale()` function is used to normalize the dataset; when the analysis completes, the `unscaled()` function will denormalize the data.

```
> #### Normalizing data ####
> library(nnet)
> norm.data <- scale(Tdata.train)
> norm.vali <- scale(Tdata.val)
> norm.test <- scale(Tdata.test)
> best.network <- matrix(c(5, 0.5))
> best.rmse <- 2
```

To optimize the number of nodes and the neurons' weight, a `for` loop will test a neural network starting with five nodes and incrementally test up to 15 nodes with three weights for the neurons on each iteration of the nodes. The root mean squared error between the actual and predicted models determines the number of neurons and weights (Fischetti, 2016). In this project, seven nodes with the decay of .02 produces the lowest root mean squared errors

```
> #### Determining the Optimal Parameters ####
> for(i in 5:15) {
+     for (j in 1:3) {
+       set.seed(1234)
+       nn.fit <- nnet(Tform,norm.data,size= i ,decay=0.01 * j ,maxit=1000,linout=T,trace=F)
+       norm.preds <- predict(nn.fit,norm.vali)
+       nn.mse <- rmse(as.numeric(norm.vali),as.numeric(norm.preds))
+       if (nn.mse<best.rmse) {
+         best.network[1, 1] <- i
+         best.network[2, 1] <- j
+         best.rmse <- nn.rmse
+       }
+     }
+ }
> best.network
     [,1]
[1,]    7
[2,]    2
```

The appropriate number of neurons and weights are used to model the test data set. The predictions are then denormalized.

```
> #### Building the Model with optimal Nodes and Weight ####
> set.seed(1234)
> nn.fit <- nnet(Tform,norm.test,size=best.network[1, 1],decay=0.01 * best.network[2, 1],maxit=1000,linout=T,trace=F)
> norm.preds <- predict(nn.fit,norm.test )
> nn <- unscale(norm.preds,norm.test)
```

The denormalized prediction data set are turned into trade signals. The `trading.signal()` package from the `DMwR` library discretizes the dataset's values generated by the ANN into a set of Buy, Sell, Hold signals. The `DMwR` library provides functions by Torgo. The signals generated from the ANN predictions are compared to the signals from the `Tdata.test` dataset.

```
> #### Evaluate Signals ####
> sigs.nn <- trading.signals(nn,0.1,-0.1)
> true.sigs <- trading.signals(Tdata.test[,"T.ind.IXIC"],0.1,-0.1)
```

Since this is an event-based prediction, we can generate a confusion matrix and evaluate the signals' precision and recall—the function `sig.PR()` calculates the precision and recall of each signal class.

```
> sigs.PR(sigs.nn, true.sigs)
     precision     recall
s   0.08695652 0.03030303
b   0.29629630 0.17582418
s+b 0.23376623 0.11464968
```

The `confusionMatrix()` function shows the frequency of the signals (Viswanathan 2016). The overall statistics show there is 65.5% accuracy between the ANN predictions and the true values.

```
> confusionMatrix(sigs.nn ,true.sigs)
Confusion Matrix and Statistics

          Reference
Prediction   s   h   b
         s   2  12   9
         h  47 310  66
         b  17  21  16

Overall Statistics

               Accuracy : 0.656
                 95% CI : (0.6125, 0.6976)
    No Information Rate : 0.686
    P-Value [Acc > NIR] : 0.9315

                  Kappa : 0.1267

 Mcnemar's Test P-Value : 4.44e-10

Statistics by Class:

                     Class: s Class: h Class: b
Sensitivity           0.03030   0.9038   0.1758
Specificity           0.95161   0.2803   0.9071
Pos Pred Value        0.08696   0.7329   0.2963
Neg Pred Value        0.86583   0.5714   0.8318
Prevalence            0.13200   0.6860   0.1820
Detection Rate        0.00400   0.6200   0.0320
Detection Prevalence  0.04600   0.8460   0.1080
Balanced Accuracy     0.49096   0.5920   0.5415
```

Analyzing the performance of the ANN model

The `trade.simulator()` function can carry out a trading simulation with signals of any model. With the trade `simulator.function()`, a user can define a trading policy and its parameters. The parameters include the trader's initial capital and the cost for each transaction the market quotes for a specified time. The result of the simulator is an object of class `tradeRecord` with information from the simulation. The R package performance lyrics analyzes the return of some fading algorithm. Some of the functions from the performance analytics package will be used on the `tradeRecord` object.

```
> #### getting the quotes for the testing period
> # learning the model and obtaining its signal predictions
> date <- rownames(Tdata.test[1,])
> market <- IXIC[paste(date,'/',sep='')][1:len.ts]
```

This project will use the policies define by Torgo. `Policy.1` will open a position if the market is forecasted to pass a certain threshold. `Policy.1` will maintain that position for ten days or if the loss is 5%. `Policy.1` initially starts with $1,000,000 and invests 20%.

`Policy.2` will open a position if the price is forecasting to pass a certain threshold and maintain that position indefinitely or a 5% loss. `Policy.2` also initially starts with $1,000,000 and invests 20%. The simulation would measure the latest 500 days of the Nasdaq Composite dataset. The latest 500 days is partitioned out in the `Tdata.test` object of which the Buy, Sell, Hold signals have been generated. The code to define policy one and policy two functions is available in this project's file.

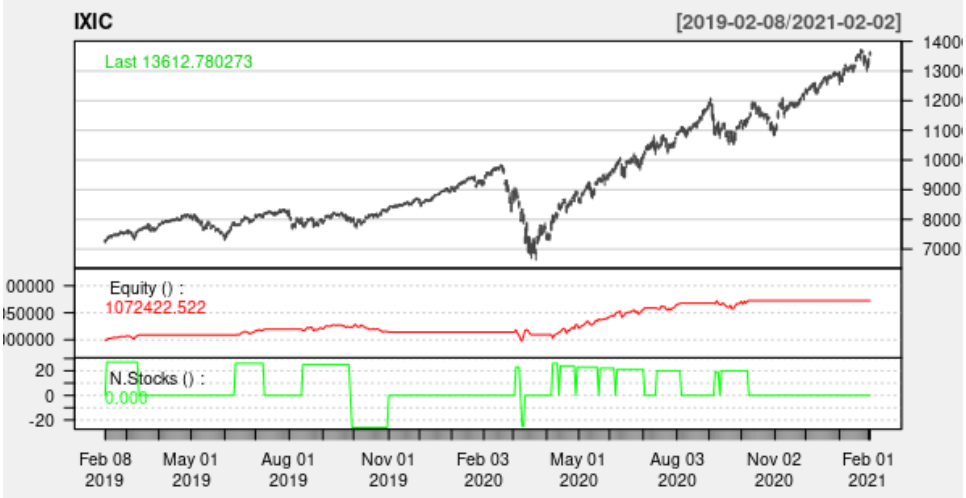
The call to the training simulator created an object of class trade record. The contents of the trade record can be viewed with the `summary()` function. The `tradingEvaluation()` function displays the performance during the simulation. The plot function generates a histogram.

The results and performance of `Policy.1` are shown; the histogram displays actions performed. `Policy.1` shows a rentability of 7.24%. The rentability signifies the profit gained in this period. The plot shows total equity of $1072422.52.

```
> t1 <- trading.simulator(market,sig,
+                         'policy.1',list(exp.prof=0.05,bet=0.2,hold.time=30))
> t1

Object of class tradeRecord with slots:

          trading: <xts object with a numeric  500 x 5  matrix>
          positions: <numeric  18 x 7  matrix>
          init.cap :  1e+06
          trans.cost :  5
          policy.func :  policy.1
          policy.pars : <list with  3  elements>
```

```
> tradingEvaluation(t1)
    NTrades       NProf     PercProf          PL         Ret   RetOverBH       MaxDD SharpeRatio     AvgProf     AvgLoss
      18.00       12.00       66.67    72422.52        7.24      -79.28    30904.58        0.08        4.44       -2.84
      AvgPL     MaxProf     MaxLoss
       2.01        8.83       -6.61
```

```
> plot(t1,market,theme='white',name='IXIC')
Rentability =  7.242252 %
```

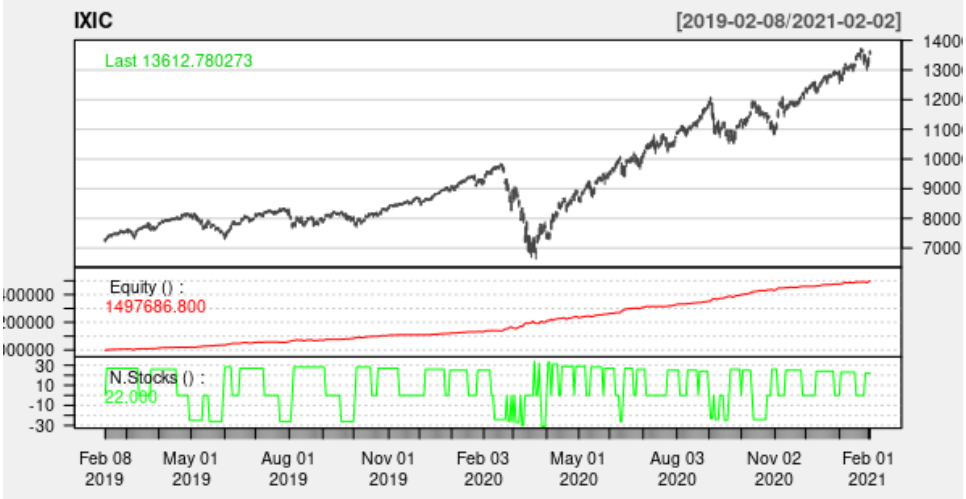


The results of true signals at `policy.1` shows similar gains.

```
> #Using the Simulated Trader on true.sigs with policy.1
> true.t1 <- trading.simulator(market,true.sigs,
+                              'policy.1',list(exp.prof=0.05,bet=0.2,hold.time=30))
> true.t1

Object of class tradeRecord with slots:

          trading: <xts object with a numeric  500 x 5  matrix>
          positions: <numeric  49 x 7  matrix>
          init.cap :  1e+06
          trans.cost :  5
          policy.func :  policy.1
          policy.pars : <list with  3  elements>
```

```
> tradingEvaluation(true.t1)
    NTrades       NProf     PercProf          PL         Ret   RetOverBH       MaxDD SharpeRatio     AvgProf     AvgLoss
      49.00       45.00        91.84   497686.80       49.77      -36.75    16417.19        0.34        4.78       -2.78
      AvgPL     MaxProf     MaxLoss
       4.17        8.83       -6.28
```

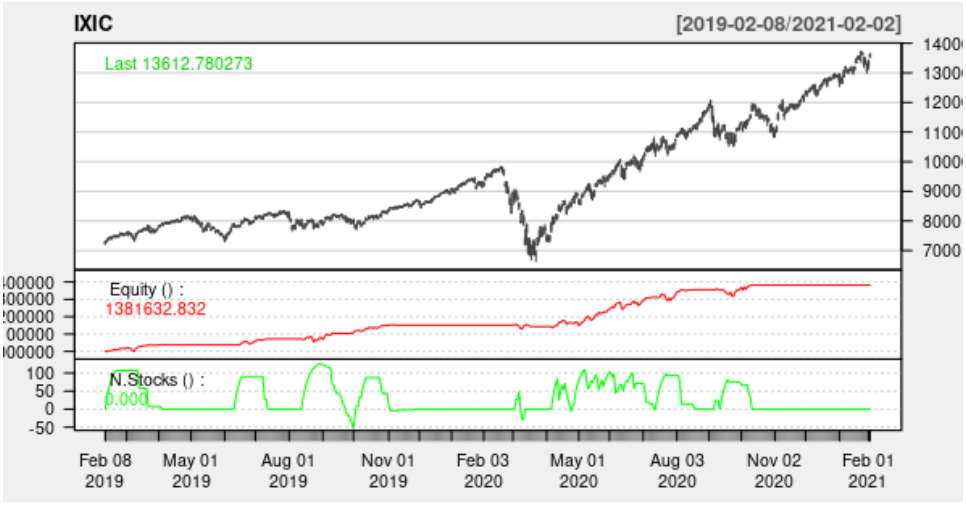
```
> plot(true.t1,market,theme='white',name='IXIC')
Rentability =  49.76868 %
```



`Policy.2` shows a 38 .16% rentability. `Policy.2` performed better than `Policy.1`. Although `Policy.1` did not perform as well as `Policy.2`, many other factors contribute to the difference in performance.

```
> t2 <- trading.simulator(market,sig,
+                         'policy.2',list(exp.prof=0.05,bet=0.2))
> summary(t2)

== Summary of a Trading Simulation with  500  days ==

Trading policy function :  policy.2
Policy function parameters:
         exp.prof  =  0.05
         bet  =  0.2

Transaction costs :  5
Initial Equity    :  1e+06
Final Equity      :  1381633   Return :  38.16 %
Number of trading positions:  122

Use function "tradingEvaluation()" for further stats on this simulation.
```

```
> tradingEvaluation(t2)
    NTrades       NProf     PercProf          PL         Ret   RetOverBH       MaxDD SharpeRatio     AvgProf     AvgLoss
     122.00       94.00        77.05   381632.83       38.16      -48.36    43795.93        0.14        4.73       -4.02
      AvgPL     MaxProf     MaxLoss
       2.72        8.83       -6.61
```

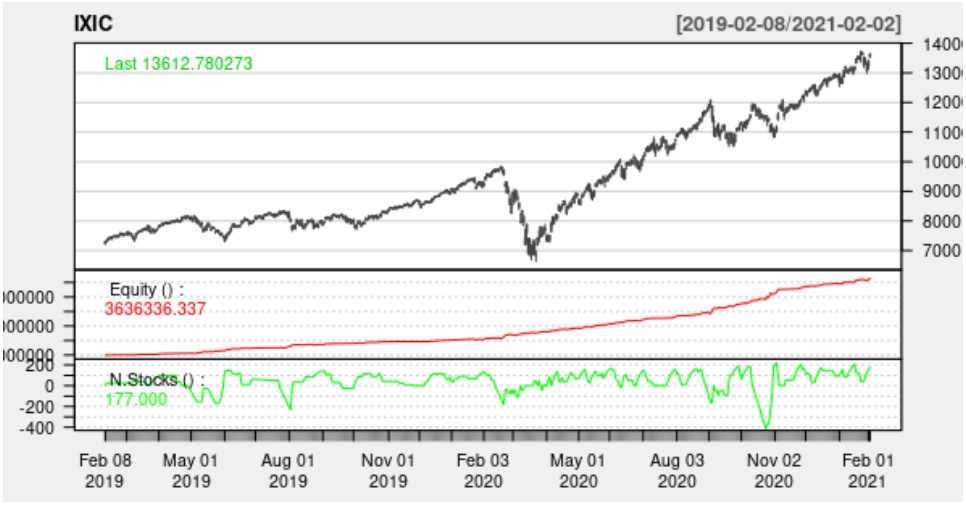


The results of true signals at `policy.2` shows similar gains and also disproportionate to the ANN signals at `policy.1`.

```
> #Using the Simulated Trader on true.sigs with policy.2
> true.t2 <- trading.simulator(market,true.sigs,
+                              'policy.2',list(exp.prof=0.05,bet=0.2))
> summary(true.t2)

== Summary of a Trading Simulation with  500  days ==

Trading policy function :  policy.2
Policy function parameters:
         exp.prof  =  0.05
         bet  =  0.2

Transaction costs :  5
Initial Equity    :  1e+06
Final Equity      :  3636336   Return :  263.63 %
Number of trading positions:  230

Use function "tradingEvaluation()" for further stats on this simulation.
```

```
> tradingEvaluation(true.t2)
    NTrades       NProf    PercProf          PL         Ret   RetOverBH       MaxDD SharpeRatio     AvgProf     AvgLoss
     230.00      214.00       93.04  2636336.34      263.63      177.11    48187.96        0.38        4.64       -2.60
      AvgPL     MaxProf     MaxLoss
       4.13        8.83       -8.06
```

```
> plot(true.t2,market,theme='white',name='IXIC')
Rentability =  263.6336 %
```



Summary of Analysis

February 8th, 2019, the value of the index closed at $7298.20. On February 2nd, 2021, the value of the index closed at $13,612.78. In the 500 days analyzed, there is a difference of $6314.58. If a long position were opened on February 8th, 2019, and closed on February 2nd, 2021, there would be a gain of 86.5% in equity. With the low values of precision and recall, and The difference in yield between using a Random Forest and ANN model and not using a Random Forest and ANN model leads us to fail to reject the H0.

It is also worth noting during this project generated inconsistent models even with the `set.seed()` function set. The `set.seed()` function initializes the random number generator to some seed number (Torgo, 2010). The seed number ensures the ANN model is the same every time the `nnet()` function is run to create the ANN model.

Further study needs to be conducted in the following:

- Building an ANN model by averaging out several ANN models. Averaging out several ANN models helps in eliminating any outlier models.

- Comparing various modeling tools such as Support Vector Machines (SVM) and Multivariate Adaptive Regression Spline (MARS). Various modeling tools can be compared using the Monte Carlo method in order to determine which modeling tool generates the most accurate predictions.
- Using neuro network models to optimize different trading strategies. Rather than using models to detect trends, neuro networks can be configured to detect other trading strategies such as Charting Elliott Wave, Lucas, Fibonacci, Gann, and Time for Profit strategies (Greenblatt, 2013).

Resources

Atsalakis, G. S., & Valavanis, K. P. (2009). Surveying stock market forecasting techniques – Part II: Soft computing methods. *Expert Systems with Applications, 36*(3), 5932–5941. https://doi-org.links.franklin.edu/10.1016/j.eswa.2008.07.006

Fischetti, T., Lantz, B., Abedin, J., Mittal, H. V., Makhabel, B., Berlinger, E., … Daroczi, G. (2016). *R: Data Analysis and Visualization*. Birmingham, England: Packt Publishing.

FT Press Delivers collections (2011) *Investor's library fundamental analysis, technical analysis, and income investing. - Title from resource description page (viewed April 5th, 2011). - "FT Press Delivers collections"--Cover. - Includes*. (2011). Upper Saddle River, N.J.: FTPress Delivers.

Greenblatt, J. (2013). *Breakthrough strategies for predicting any market: Charting Elliott wave, Lucas, Fibonacci, Gann, and time for profit* (2nd ed.). Nashville, TN: John Wiley & Sons.

Mullaney, M. (2009). *The complete guide to option strategies: Advanced and basic strategies on stocks, ETFs, indexes, and stock index futures*. Nashville, TN: John Wiley & Sons.

Torgo, L. (2010). *Data mining with R: Learning with case studies*. Caithness, UK: Whittles Publishing.

Viswanathan, V. (2016). *R: Recipes for analysis, visualization and machine learning: Get savvy with R language and actualize projects aimed at analysis, visualization and machine learning*. Birmingham: Packt.

Ulrich Joshua (n.d.). xts function. Retrieved February 24, 2021, from Rdocumentation.org website: https://www.rdocumentation.org/packages/xts/versions/0.12.1/topics/xts

Wiley, M., & Wiley, J. F. (2019*). Advanced R statistical programming and data models: Analysis, machine learning, and visualization* (1st ed.). APRESS.

Williams, R. T. (2011). *An Introduction to Trading in the Financial Markets: Market Basics*. Burlington, MA: Academic Press.

Yahoo Finance (n.d.) Symbol Lookup from Yahoo Finance. (n.d.). Retrieved February 19th, 2021, from Yahoo.com website: https://finance.yahoo.com/quote/