Trading Rule Back Testing

Khizar Tahir

Introduction

Algorithmic trading is a system of trading that follows a defined algorithm in financial markets. This type of system is ideal for minimizing human intervention, resulting in fewer errors and fast and efficient decision making. Due to abundant past data availability, it is possible to provide expectations about future performance from past performance using a process called backtesting [1]. Backtesting allows a user to simulate a desired trading strategy using past data to determine whether it might be successful and it leads to a set of trading signals [2]. It relies on the assumption that stocks move in similar patterns as they did historically. The result of backtesting could be that we accept or reject a proposed strategy.

Bias can be unintentionally implemented if the strategy is not tested on different time periods with a representative set of stocks. In addition, incorporating information into the backtested model that would not have been available during the period being tested is known as look-ahead bias [3]. This could inadvertently skew results to be closer to the desired outcome of the test. Look-ahead bias is counteracted by applying the lag function to the signal factor. Insufficient sample bias could also be erroneously included by not backtesting enough observations. It is important to ensure that a large number of observations are included in the analysis to get a fair model [3].

Project Overview

In this project, we will be backtesting our algorithmic trading strategy using the DVI Indicator as the signal factor. The DVI indicator was introduced in 2009 as a good predictor for the S&P 500 over the past 30 years. It oscillates between 0 and 1 and the basic understanding is that if the DVI is < 0.5, enter long and enter short if it’s >= 0.5. We will apply that strategy in our functions.

The project has 3 functions corresponding to the 3 components:

1. **myfun1** which performs the Base Level Backtest. It takes as input the TICKER for security, begin date for back testing, end date for back testing, and threshold for DVI (set to default value of 0.5). It outputs a summary table containing the total number of long trades, the total number of short trades, the percent of time the portfolio is long, the percent of time the portfolio is short, and the cumulative return from the strategy.
2. **myfun2** which simulates Multiple Back Test periods. It takes as input the TICKER for security, testing period in years, date range (years) for overall data availability, threshold for DVI (set to default value of 0.5). It then outputs a Table containing the outputs of Function 1 for all iterations performed using as many possible linear combinations of date range as possible using the test period given; and a Plot containing the cumulative return from each Test period.
3. **myfun3** which simulates Multiple DVI Thresholds. It takes as inputs the TICKER for security, begin date for back testing, end date for back testing, lower value for threshold, higher value for threshold, and increments for threshold. It Outputs a Table containing the DVI threshold, the total number of long trades, the total number of short trades and the cumulative return from the strategy. It also produces a plot with threshold value on x-axis and its corresponding cumulative return on y-axis.

Require Libraries

As with all analyses, the first step in this assignment is to download all libraries we need to complete our backtesting. Quantmod allows us to download stock data within a specified timeframe. TTR allows us to access trading signals, PerformanceAnalytics is used for advanced metrics, lubridate is used for date/time adjustments, and finally we use ggplot2 for required plots.

```r
library(quantmod)
library(TTR)
library(dplyr)
library(PerformanceAnalytics)
library(lubridate)
library(ggplot2)
```

Function 1: Base Level Back Test

Function #1, defined as myfun1, performs a simple back test that implements the rule of long when DVI < 0.5 and enters short when DVI >= 0.5. The function is coded to accept full format dates as well as just years. Next, we create Price and Retn objects to capture the daily price and return values and subset the price object to obtain just the closing prices. The dvi object simply calculates the DVI indicator on our prices. After determining the signal factor, we apply the trading rule. Our Position object applies our rule. If the DVI is less than the inputted threshold (which also has a default value of 0.5), we enter long (+1/buy) and if the DVI is >= the threshold, we enter short (-1/sell) position. We use the lag function here to ensure that yesterday’s signal is applied to today’s returns. This assists in mitigating the effects of look-ahead bias, as discussed in the introduction.

Once we have all of our necessary data compiled, we merge all of this information into an xts object and do some data cleaning. This allows us to then create summary statistics on our findings. The freq object creates a frequency table of all Positions which then allows us to create the long_count and short_count objects to sum the frequencies of long and short positions. From this, we create a proportion table which is expressed as the long_pcnt and short_pcnt objects calculating the percentage of long and short positions (to 2 decimal points). Finally, we calculate the cumulative return (to 3 decimal points) from the entire strategy and summarize all of our findings in the summary1 table.

```
myfun1 <- function(tic, start, end, thresh = 0.5){
  if (nchar(as.character(start)) == "8"){
    start <- as.Date(as.character(start), format = "%Y%m%d") #converting dates to appropriate format
    end <- as.Date(as.character(end), format = "%Y%m%d")
  }
  else if(nchar(as.character(start)) == "4"){
    start <- as.character(start)
    end <- as.character(end)
  }
  Price <- getSymbols(tic, auto.assign = F, periodicity = "daily")
  Price <- Price[,4]
  Price <- Price[paste(start, end, sep = "/")]
  Retn <- periodReturn(Price, period = 'daily', type = 'arithmetic')
  dvi <- DVI(Price)
  Position <- Lag(ifelse(dvi$dvi < thresh, 1, -1))
  dfxts <- merge(Price, Retn, dvi$dvi, Position)
  dfxts <- na.omit(dfxts)
  colnames(dfxts) <- c("Price", "Retn", "dvi", "Position")
  dft <- as.data.frame(dfxts) # for plotting
  dft$Date <- rownames(dft) # for plotting
  df1 <<- dft # variable for graphing
  freq <- table(dfxts$Position)
  long_count <- freq[["1"]]
  short_count <- freq[["-1"]]
  prop <- prop.table(table(dfxts$Position))
  long_pcnt <- paste(round(prop[["1"]]*100, digits = 2),"%", sep = "")
  short_pcnt <- paste(round(prop[["-1"]]*100, digits = 2), "%", sep = "")
  cumret1 <- round(unname(Return.cumulative(Retn*Position)), digits = 3)
  summary1 <- data.frame("LongCount" = long_count, "ShortCount" = short_count,
                         "LongPcnt" = long_pcnt, "ShortPcnt" = short_pcnt,
                         "CumRet" = cumret1)
  return(summary1)
}
```

Running Function 1

Here, we are running Function 1 for ticker: “JNJ”, start date: “20140101”, end date: “20171231”, threshold: 0.5

```
result1 <- myfun1("JNJ", "20140101", "20171231", 0.5)
print(result1)
```

```
##   LongCount ShortCount LongPcnt ShortPcnt CumRet
## 1       312        338      48%       52% -0.167
```

Function 1 Result

We see that for the given date range (2014-2017), and a DVI threshold of 0.5, the strategy went into a **loss** of 0.167; with less number of Long positions than Short. This was just one backtest. In the next 2 sections/functions, we will backtest more and explore the results of this strategy in multiple ways.

Function 2: Simulate Multiple Back Test Periods

Function 2, defined as myfun2, runs the strategy multiple times. It takes the range of years to run strategy on and the testing period. It is programmed to run Function 1 for all possible linear lengths of the testing period from the start of range to its end. E.g. if the range is 2010-2016 with the period being 3 years, the first iteration will be for 2010-2012, the next 2011-2013 and so on until 2016. This is achieved using a for loop. For each iteration, we reuse our Function 1, and append the outcome to a dataframe named df_temp. At the end of the loop, df_temp contains summary values from Function 1 for all of our iterations.

Additionally, we plot a bar graph using ggplot function with the Testing period on x-axis and cumulative return from that period on y-axis. Positive cumulative return bars have a Blue fill, negative return bars have Red allowing us to easily visualize in which periods was the strategy successful.

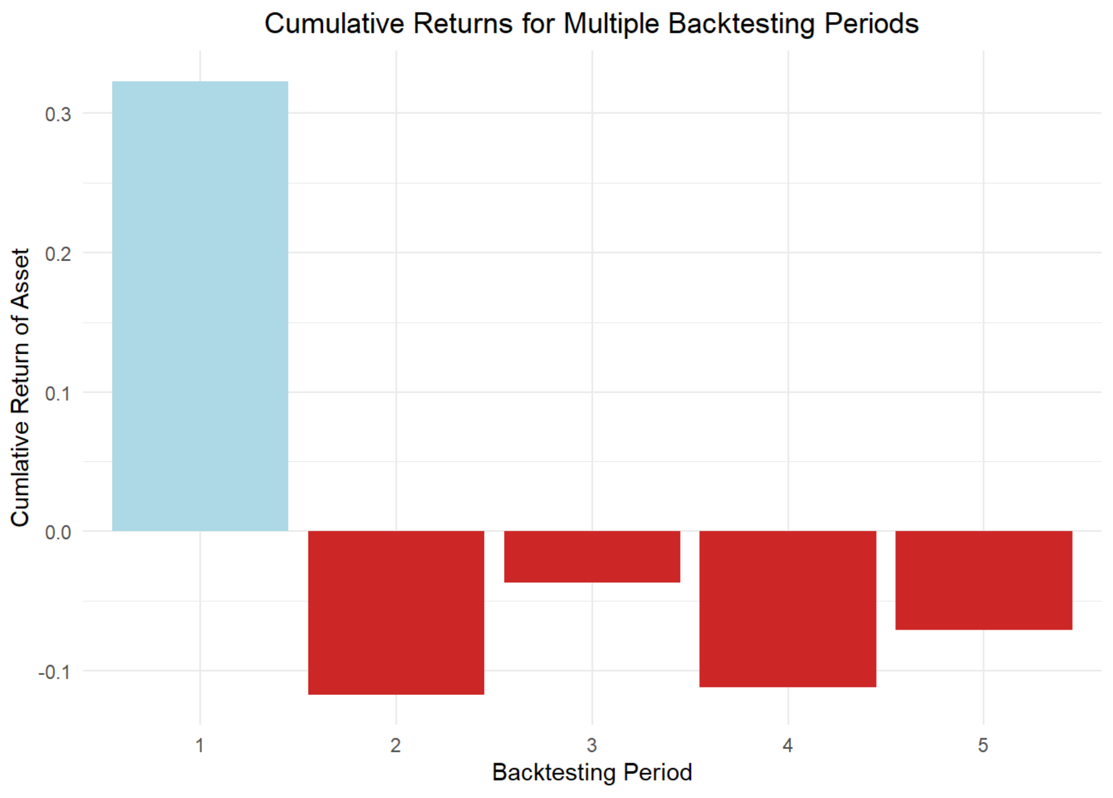
```
myfun2 <- function(tic, per, range, thresh = 0.5){
  range_start <- as.numeric(range[[1]]) # storing start date from range as a number for manipulation
  range_end <- as.numeric(range[[2]]) # storing end date from range as a number for manipulation
  df_temp <- data.frame("LongCount"= NULL, "ShortCount"=NULL, "LongPcnt"=NULL, "ShortPcnt"=NULL, "CumRet"=NULL) #
creating empty dataframe
  for (i in range_start:(range_end - per + 1)){ # loop is such that all linear combinations of testing period are
utilized.
    res1 <- myfun1(tic, i, i + per - 1, thresh) # for each defined period, we pass period range to our function
 1.
    df_temp <- bind_rows(df_temp, res1) # result is bound as a row to the empty dataframe
  }
  #Plot for #2
  plot1 <- ggplot(data=df_temp, aes(x = rownames(df_temp), y = CumRet, fill = CumRet > 0)) +
    geom_bar(stat = "sum", show.legend = FALSE) +
    scale_fill_manual(values = c("TRUE"="lightblue", "FALSE"="firebrick3")) +
    labs(fill = "Testing Period") +
    ggtitle("Cumulative Returns for Multiple Backtesting Periods") +
    xlab("Backtesting Period") +
    ylab("Cumlative Return of Asset") +
    theme_minimal()+
    theme(plot.title = element_text(hjust = 0.5))
  print(df_temp)
  print(plot1)
}
```

Running Function 2

Here, we are running Function 2 for ticker: “JNJ”, Testing period: 3, range of years: 2010-2016, threshold: 0.5

```
result2 <- myfun2("JNJ", 3, c("2010","2016"), 0.5)
```

```
##   LongCount ShortCount LongPcnt ShortPcnt CumRet
## 1       199        198   50.13%    49.87%  0.323
## 2       174        223   43.83%    56.17% -0.117
## 3       225        172   56.68%    43.32% -0.037
## 4       212        187   53.13%    46.87% -0.112
## 5       196        203   49.12%    50.88% -0.071
```



Function 2 Result

The summary table shows the Back Test summary statistics for each iteration. We see that our inputs returned 5 iterations of back testing our strategy.

The bar graph gives us some insights. We see that this strategy only gave us a Positive Return once, and Negative returns the next four iterations. At this point, we can conclude that this exact strategy might not be efficient. The next section/function will tell help us answer ‘what, if not this?’.

Function 3: Simulate Multiple DVI Thresholds

Function 3, defined as myfun3, runs the strategy for a range of values of the DVI threshold. Each iteration is for the same Ticker and date range but with a different DVI threshold as defined by the user. Using a for loop, we start with the lower value of threshold inputted to the function, and increment using the increment value until the higher value of threshold is reached. In each iteration, the ticker, start date, end date, and threshold are passed onto Function 1 which again outputs the strategy summary statistics. Key values from these outputs are stored along with threshold value used in each iteration in a dataframe named df2.

Additionally, we use ggplot function to plot a bar graph with DVI threshold value on x-axis and its corresponding cumulative return from strategy on y-axis. Positive return bars are filled blue, and negative Red so we can visualize which thresholds result in positive/negative returns.

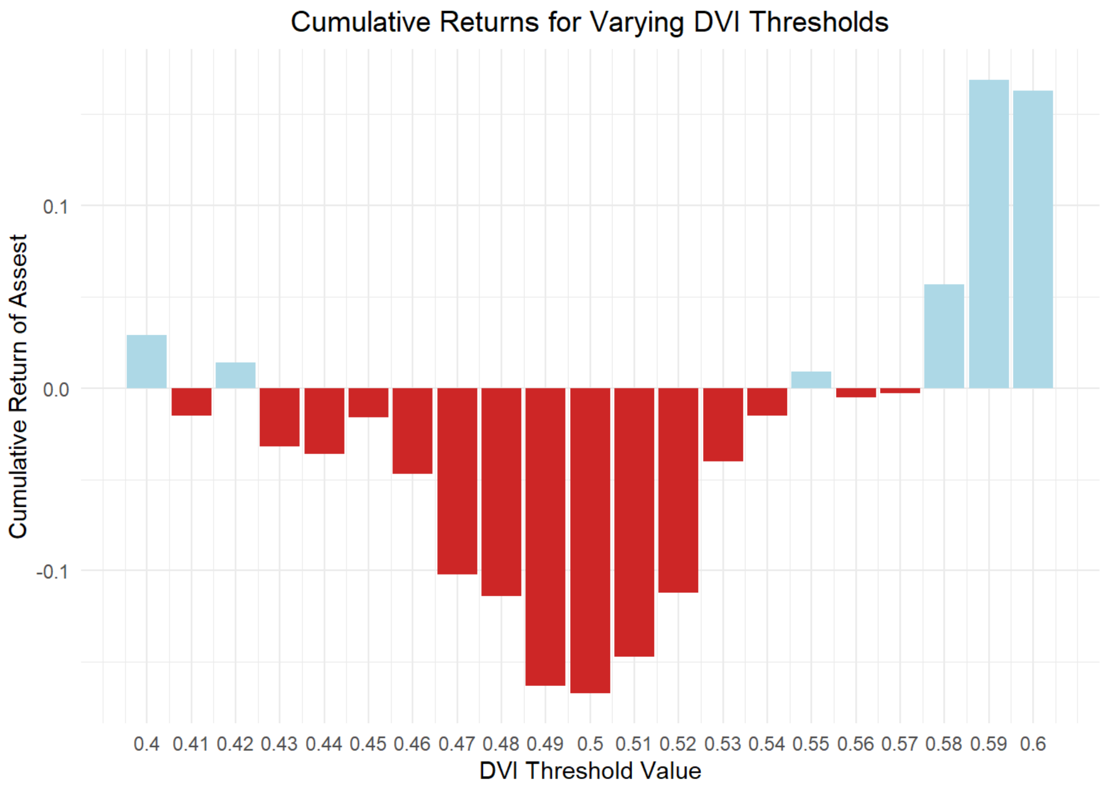
```
myfun3 <- function(tic, start, end, low, high, inc){
  df2 <- data.frame("DVI_Thresh" = NA, "LongCount"= NA, "ShortCount" = NA, "CumRet" = NA) # empty dataframe to fi
ll.
  for (i in seq(low, high, inc)){ # Loop used to cover all threshold values given the range and increment.
    res2 <- myfun1(tic, start, end, i) # each threshold passed to function 1.
    df2[nrow(df2)+1,] <- c(i, res2$LongCount, res2$ShortCount, res2$CumRet) # result inserted as a row to empty d
ataframe.
  }
  df2 <- na.omit(df2)
  rownames(df2) <- NULL
  plot3 <- ggplot(data=df2, aes(x = DVI_Thresh, y = CumRet, fill = CumRet > 0)) +
    geom_bar(stat = "sum", show.legend = FALSE) +
    scale_fill_manual(values = c("TRUE"="lightblue", "FALSE"="firebrick3")) +
    scale_x_continuous("DVI Threshold Value", labels = as.character(df2$DVI_Thresh), breaks = df2$DVI_Thresh) +
    ggtitle("Cumulative Returns for Varying DVI Thresholds") +
    xlab("DVI Threshold Value") +
    ylab("Cumulative Return of Assest") +
    theme_minimal()+
    theme(plot.title = element_text(hjust = 0.5))
  print(df2)
  print(plot3)
}
```

Running Function 3

We run Function 3 for ticker: “JNJ”, start date: “20140101”, end date: “20171231”, lower value of threshold: 0.4, higher value of threshold: 0.6, increment: 0.01

```
result3 <- myfun3("JNJ", "20140101", "20171231", 0.4, 0.6, 0.01)
```

```
##    DVI_Thresh LongCount ShortCount CumRet
## 1        0.40       250        400  0.029
## 2        0.41       256        394 -0.015
## 3        0.42       263        387  0.014
## 4        0.43       269        381 -0.032
## 5        0.44       274        376 -0.036
## 6        0.45       281        369 -0.016
## 7        0.46       286        364 -0.047
## 8        0.47       292        358 -0.102
## 9        0.48       301        349 -0.114
## 10       0.49       306        344 -0.163
## 11       0.50       312        338 -0.167
## 12       0.51       320        330 -0.147
## 13       0.52       324        326 -0.112
## 14       0.53       336        314 -0.040
## 15       0.54       347        303 -0.015
## 16       0.55       354        296  0.009
## 17       0.56       360        290 -0.005
## 18       0.57       366        284 -0.003
## 19       0.58       373        277  0.057
## 20       0.59       379        271  0.169
## 21       0.60       389        261  0.163
```



Function 3 Result

The summary tables shows us the number of long and short trades and the cumulative return from the strategy at each level of threshold withing our range.

The bar graph gives more insight. We observe that within our DVI threshold range, the returns are positive near the ends and are negative towards 0.5. In fact, at the threshold of 0.5, the return is Most Negative.

Insights

Points below outline our strategy for Back testing and the insights we can generate from the results:

- We started off by choosing a date range to run strategy on and set the DVI threshold to its widely used value of 0.5.
- It resulted in a negative cumulative return meaning for that period and threshold, the strategy did not work. Before rejecting this, we need to check for 2 issues which might be causing this:
  1. Maybe the period we are running this for had inconsistent result. Better to test on multple testing periods.
  2. Maybe the DVI threshold we are using is not right. Best to check for multiple values.
- To address issue 1, we ran Function 2 for multiple testing periods. Except for one, all other iterations again gave Negative returns. Meaning this strategy is not useful even with different testing periods.

- Finally, we address issue 2 through Function 3 and see that for a given date range, the returns are lowest with threshold of 0.5 or near. And the results are highest for threshold of 0.59 or 0.6.
- With the point above, we can conclude that the threshold we set earlier was not suitable for this dataset hence a slightly greater value would yield better return.
- In fact, running Function 1 for a DVI threshold of 0.59 gives a **Positive Cumulative Return** of 0.169. This is a good threshold for a dataset like this.

Conclusion

In this project, we demonstrated the use of Back testing to validate our algorithmic trading strategy. We demonstrated how testing for multiple date ranges and DVI threshold values can allow us to choose the best strategy, one that gives the maximum return - as best as can be guessed using past data. We also saw how summary tables and visualizations allow instant interpretation of results.

Sources

[1] Seth, S. (2020, August 28). Basics of Algorithmic Trading: Concepts and Examples. Retrieved November 09, 2020, from https://www.investopedia.com/articles/active-trading/101014/basics-algorithmic-trading-concepts-and-examples.asp (https://www.investopedia.com/articles/active-trading/101014/basics-algorithmic-trading-concepts-and-examples.asp)

[2] Successful Backtesting of Algorithmic Trading Strategies - Part I. (n.d.). Retrieved November 09, 2020, from https://www.quantstart.com/articles/Successful-Backtesting-of-Algorithmic-Trading-Strategies-Part-I/ (https://www.quantstart.com/articles/Successful-Backtesting-of-Algorithmic-Trading-Strategies-Part-I/)

[3] Backtesting - Overview, How It Works, Common Measures. (n.d.). Retrieved November 12, 2020, from https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/backtesting/ (https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/backtesting/)