Stock Price Trend Forecasting with Machine Learning Algorithms

Shane Bari, Cassandra Dale, Russell Chamberlain, Jared Siverling

December 17, 2015

This document will give an overview of the Primary Model code written in R:
- Importing/Manipulating Data
- Calculating Moving Averages
- Generating *buy* and *sell* triggers
- Calculating Profit

Load Required R Packages

R packages are collections of predefined functions, data, and compiled code. The standard set of R packages is known as the Comprehensive R Archive Network (CRAN). So far we are only using the *quantmod* package to import financial data from Yahoo, Google, Federal Reserve, etc.

More packages may become necessary to assist in construct technical trading rules as well as to provide reliable machine learning algorithms.

```r
library(quantmod)
```

Import Data

The `getSymbols()` function imports daily OHLC including Volume and Adjusted data for a given symbol, from a given source, on a given time period. In this example we are importing SPY from Yahoo Finance from January 1, 2014 to today.

```r
getSymbols('SPY', src='yahoo', from='2014-1-01', to=Sys.Date())
```

```
##            SPY.Open SPY.High SPY.Low SPY.Close SPY.Volume SPY.Adjusted
## 2014-01-02   183.98   184.07  182.48    182.92  119636900     176.8330
## 2014-01-03   183.23   183.60  182.63    182.89   81390600     176.8040
## 2014-01-06   183.49   183.56  182.08    182.36  108028200     176.2916
## 2014-01-07   183.09   183.79  182.95    183.48   86144200     177.3743
## 2014-01-08   183.45   183.83  182.89    183.52   96582300     177.4130
## 2014-01-09   184.11   184.13  182.80    183.64   90683400     177.5290
```

This data is then reformatted into matrix form for easy reference and manipulation. For example, `X[,1]` refers to the first column of the `X` matrix, which corresponds to the *open* values for each day.

```r
X <- matrix(cbind(SPY), ncol = 6)
```

```
##        [,1]   [,2]   [,3]   [,4]      [,5]     [,6]
## [1,] 183.98 184.07 182.48 182.92 119636900 176.8330
## [2,] 183.23 183.60 182.63 182.89  81390600 176.8040
## [3,] 183.49 183.56 182.08 182.36 108028200 176.2916
## [4,] 183.09 183.79 182.95 183.48  86144200 177.3743
## [5,] 183.45 183.83 182.89 183.52  96582300 177.4130
## [6,] 184.11 184.13 182.80 183.64  90683400 177.5290
```

Weighted Moving Average

`WMA()` calculates a weighted moving average based on given relative weights of a Simple Moving Average (SMA) and Exponential Moving Average (EMA) as well as the time period being averaged.

Ex: `WMA(0.70,0.30,X[,1],50)` gives a 50 day WMA, with SMA given more weight (70%) and EMA given less weight (30%)

Side Note: lines of code beginning with `#` are comments meant to increase understanding and readability of the code and will be ignored when the code is run.

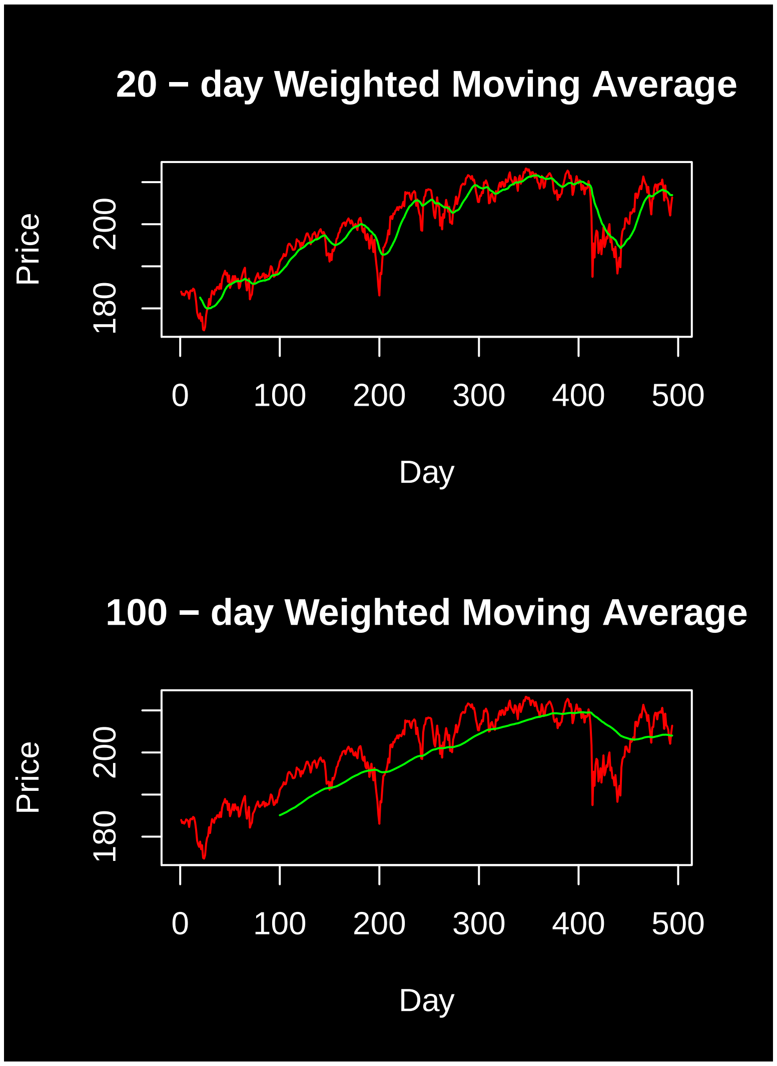
```
WMA <- function(SMAweight=0, EMAweight=0, dataRaw, timePeriod){
  # calculate SMA and EMA
  SMA = SMA(dataRaw, timePeriod)
  EMA = EMA(dataRaw, timePeriod)

  # find distance between averages and data
  dist_SMA = dataRaw - SMA
  dist_EMA = dataRaw - EMA

  # set up WMA fillable
  # if weights are both zero, function will find GMA
  # if weights do not sum to one, function will print a note and will defalut to GMA
  WMA = c()

  if (SMAweight==0 && EMAweight==0){
    for(i in timePeriod:length(dataRaw)){
      if(abs(dist_SMA[i]) > abs(dist_EMA[i])){
        WMA[i] = EMA[i]
      }else{WMA[i] = SMA[i]}
    }
  } else if ((SMAweight + EMAweight) != 1){
    print("Weighted values do not sum to one.Program has defaulted to finding GMA
          instead of WMA.")
  } else {
    for(i in timePeriod:length(dataRaw)){
      WMA[i] = SMAweight*SMA[i] + EMAweight*EMA[i]
    }
  }
  return (WMA)
}
```

Plot Data and WMA

`plotMAs()` displays a graph of the stock price data as well as the WMA for two given time periods.

```r
plotMAs <- function(dataPlot, shortTerm, shortWMA, longTerm, longWMA){
  # configure plots

  par(bg='black', mfrow=c(2,1),
      col.axis='white',col.lab='white',col.main='white',col.sub='white')

  # set up and plot
  plot(dataPlot,type='l',col='red',fg='white',
       main=paste(shortTerm,'- day Weighted Moving Average'),
       xlab='Day',ylab='Price')
  lines(shortWMA,col='green')

  plot(dataPlot,type='l',col='red',fg='white',
       main= paste(longTerm,'- day Weighted Moving Average'),
       xlab='Day',ylab='Price')
  lines(longWMA,col='green')

}
```

Ex: A visual comparison of WMA for two different time periods.



`mAvgsAll()` references `WMA()` and `plotMAs()` and returns a data matrix (3 columns) for *stock price*, *short term WMA*, and *long term WMA*. At this point the desired column from the original OHLC data can be specified by assigning a logical value of `TRUE` to one of the columns in the original function definition (at the top).

Ex: `open=TRUE` selects the *open* column; `high=TRUE` selects the *high* column.

```
mAvgsAll <- function(x,n_day, m_day, weightSMA, weightEMA, open = TRUE, high = FALSE,
                     low = FALSE, close = FALSE, adjusted = FALSE){

  if (open){
    k = 1

    #Calculate WMA (or GMA)
    WMA_n = WMA(weightSMA, weightEMA, x[,k], n_day)
    WMA_m = WMA(weightSMA, weightEMA, x[,k], m_day)

  } else if (high){
    k = 2

    #Calculate WMA (or GMA)
```

```r
    WMA_n = WMA(weightSMA, weightEMA, x[,k], n_day)
    WMA_m = WMA(weightSMA, weightEMA, x[,k], m_day)

  } else if (low){
    k = 3

    #Calculate WMA (or GMA)
    WMA_n = WMA(weightSMA, weightEMA, x[,k], n_day)
    WMA_m = WMA(weightSMA, weightEMA, x[,k], m_day)

  } else if (close){
    k = 4

    #Calculate WMA (or GMA)
    WMA_n = WMA(weightSMA, weightEMA, x[,k], n_day)
    WMA_m = WMA(weightSMA, weightEMA, x[,k], m_day)

  } else if (adjusted){
    k = 6

    #Calculate WMA (or GMA)
    WMA_n = WMA(weightSMA, weightEMA, x[,k], n_day)
    WMA_m = WMA(weightSMA, weightEMA, x[,k], m_day)

  } else {
    print ("check your true/false entrys")
  }

  # plot the data and Moving Averages
  #plotMAs(x[,k], n_day, WMA_n, m_day, WMA_m)

  #set up fillable matrix
  mAvgsAllData <- matrix(cbind(x[,k], WMA_n, WMA_m), ncol=3)
  return(mAvgsAllData)
}
```

Generate *buy* and *sell* triggers

Compare Data to WMA

`gPredDataMA()` compares the data to the short- and long-term WMA's. If the data is less than the WMA, *buy* is displayed; if the data is greater than the WMA, *sell* is displayed.

```r
gPredDataMA <- function(x){
  len <- length(x[,1])
  # case of short term, check if data above or below GMA
  if(x[(len-1),1]>x[(len-1),2]){
    if(x[len,1]>x[(len-1),2]){
      print("Prediction from Short: wait")
    } else {
      print("Prediction from Short: sell")
    }
  } else {
    if(x[len,1]<x[(len-1),2]){
      print("Prediction from short: wait")
    } else {
      print("Prediction from short: buy")
    }
  }
  # case of long term, check if data above or below GMA
  if(x[(len-1),1]>x[(len-1),3]){
    if(x[len,1]>x[(len-1),3]){
      print("Prediction from long: wait")
    } else {
      print("Prediction from long: sell")
    }
  } else {
    if(x[len,1]<x[(len-1),3]){
      print("Prediction from long: wait")
    } else {
      print("Prediction from long: buy")
    }
  }
}
```

Similarly, `gPredDataMABuy()` and `gPredDataMASell()` return appropriate logical values (`TRUE` or `FALSE`). These logical values will be important in calculating profit later on.

```r
gPredDataMABuy <- function(x){
  len <- length(x[,1])
  # case of short term, check if data above or below WMA
  if(x[(len-1),1]>x[(len-1),2]){
    if(x[len,1]>x[(len-1),2]){
      return (FALSE)
    } else {
      return (FALSE)
    }
  } else {
    if(x[len,1]<x[(len-1),2]){
      return (FALSE)
    } else {
      return (TRUE)
    }
  }
}
```

```r
gPredDataMASell <- function(x){
  len <- length(x[,1])
  # case of short term, check if data above or below WMA
  if(x[(len-1),1]>x[(len-1),2]){
    if(x[len,1]>x[(len-1),2]){
      return (FALSE)
    } else {
      return (TRUE)
    }
  } else {
    if(x[len,1]<x[(len-1),2]){
      return (FALSE)
    } else {
      return (FALSE)
    }
  }
}
```

In other words, `gPredDataMA()` answers the question "What should I do now?" while `gPredDataMABuy()` and `gPredDataMASell()` answer the questions "Should I buy right now?" and "Should I sell right now?" based on the value of the data compared to the WMA.

Compare short- and long-term WMA's

`gPredCrossMA()` is similar to `gPredDataMA()` but displays a *sell* trigger when the short-term WMA is less than the long-term WMA; likewise, a *buy* trigger is displayed when the short-term WMA is greater than the long-term WMA.

```
gPredCrossMA <- function(x){
  len <- length(x[,1])
  # check if short or long peroid is greater
  if(x[(len-1),2]>x[(len-1),3]){ #condition for short period being greater
    if(x[len,2]>x[len,3]){ #nothing has changed, so wait
      print("Prediction from cross: wait")
    } else { #shortMA has dipped below longMA, so sell
      print("Prediction from cross: sell")
    }
  } else{
    if(x[len,2]<x[len,3]){ #nothing has changed, so wait
      print("Prediction from cross: wait")
    } else { #shortMA has gone above longMA, so buy
      print("Prediction from cross: buy")
    }
  }
}
```

As before, `gPredCrossMASell()` and `gPredCrossMABuy()` return logical values, this time based on the same comparison of short- and long-term WMA's used in `gPredCrossMA()`.

```
gPredCrossMASell <- function(x){
  len <- length(x[,1])
  if(x[(len-1),2]>x[(len-1),3]){ #condition for short period being greater
    if(x[len,2]>x[len,3]){ #nothing has changed, so wait
      return(FALSE)
    } else { #shortMA has dipped below longMA, so sell
      return(TRUE)
    }
  } else{ #if shortMA starts below longMA we would never sell
    return(FALSE)
  }
}
```

```
gPredCrossMABuy <- function(x){
  len <- length(x[,1])
  if(x[(len-1),2]<x[(len-1),3]){ #condition for short period being greater
    if(x[len,2]<x[len,3]){ #nothing has changed, so wait
      return(FALSE)
    } else { #shortMA has gone above longMA, so buy
      return(TRUE)
    }
  } else{ # if shortMA starts below longMA we would never buy
    return(FALSE)
  }
}
```

Measure Profit

Profit Function Assumptions

In calculating the profit, several key assumptions are made:
- Only one stock is purchased or sold during a single transaction
- If stock is still held when profit is calculated, the function proceeds as if the final stock had been sold at that point.
- The function will ignore any *buy* or *sell* trigger if the stock price is less than it was the last time a stock was bought.
- *Buy* and *sell* triggers must alternate, meaning the function will wait to buy if stock is still held from a previous *buy* trigger.

Profit generated by comparison of data to WMA

`fProfitDataMA()` gives a summary of the total amount spent, the total profit generated, and the percent gained or lost for both short- and long-term WMA's.

```
fProfitDataMA <- function(x, timeShort, timeLong){
  buyAmt      = 0 # amount a unit of stock was purchased at
  sellAmt     = 0 # amount a unit of stock sold for
  profit      = 0 # money made or lost
  totalProfit = 0 # tracks total profit
  totalSpent  = 0 # tracks total buy amounts
  stockOn     = 0 # 0 if no stock, 1 if stock purchased INDICATIVE VARIABLE

  for (i in (timeShort+1):length(x[,1])){
    xShort <- x[-(i+1):-length(x[,1]),]

    if(gPredDataMABuy(xShort) && stockOn==0){
      buyAmt  = x[i,1]
      # keep track of total amount spent, only changes with a buy
      totalSpent  = buyAmt + totalSpent
```

```r
    stockOn = 1 # 1 indicates stock
  } else if(gPredDataMASell(xShort) && stockOn==1 && buyAmt<x[i,1]){
    sellAmt = x[i,1]
    profit  = sellAmt-buyAmt
    # keep track of total profit, only changes with a sell
    totalProfit = profit + totalProfit
    stockOn = 0 # restart stock count
  } else if(i == length(x[,1])){
    # if we are still waiting to sell when period ends, we sell for current amount
    if (x[i,2]<buyAmt){ ###SPECIAL CONDITION FOR SHORT###
      totalProfit = totalProfit + (x[i,1]-buyAmt)
    } else{
      # nothing is needed as amount has been sold
    }
    message("NOTE: final balance sold to calculate ROI")
  } else{
    # nothing else needs to happen because we are waiting for a sell prediction
  }
}
message(timeShort,"-day TOTALS:")
message("Total Spent: ", totalSpent)
message("Total Profit: ", totalProfit)
message("Relative Profit (ROI): ", round(((totalProfit/totalSpent)*100),2), " %")


buyAmt      = 0 # amount a unit of stock was purchased at
sellAmt     = 0 # amount a unit of stock sold for
profit      = 0 # money made or lost
totalProfit = 0 # tracks total profit
totalSpent  = 0 # tracks toatl buy amounts
stockOn     = 0 # 0 if no stock, 1 if stock purchased INDICATIVE VARIABLE

for (i in (timeLong+1):length(x[,1])){
  xLong <- x[-(i+1):-length(x[,1]),]

  if(gPredDataMABuy(xLong) && stockOn==0){
    buyAmt  = x[i,1]
    # keep track of total amount spent, only changes with a buy
    totalSpent  = buyAmt + totalSpent
    stockOn = 1 # 1 indicates stock
  } else if(gPredDataMASell(xLong) && stockOn==1 && buyAmt<x[i,1]){
    sellAmt = x[i,1]
    profit  = sellAmt-buyAmt
    # keep track of total profit, only changes with a sell
    totalProfit = profit + totalProfit
    stockOn = 0 # restart stock count
  } else if(i == length(x[,1])){
    # if we are still waiting to sell when period ends, we sell for current amount
    if (x[i,3]<buyAmt){ ###SPECIAL CONDITION FOR LONG###
      totalProfit = totalProfit + (x[i,1]-buyAmt)
    } else{
      # nothing is needed as amount has been sold
    }
```

```r
      message("NOTE: final balance sold to calculate ROI")
    } else{
      # nothing else needs to happen because we are waiting for a sell prediction
    }
  }
  message(timeLong,"-day TOTALS:")
  message("Total Spent: ", totalSpent)
  message("Total Profit: ", totalProfit)
  message("Relative Profit (ROI): ", round(((totalProfit/totalSpent)*100),2), " %")
}
```

Profit generated by comparison of short- and long-term WMA's

```r
fProfitCrossMA <- function(x, timeShort, timeLong){
  buyAmt      = 0 # amount a unit of stock was purchased at
  sellAmt     = 0 # amount a unit of stock sold for
  profit      = 0 # money made or lost
  totalProfit = 0 # tracks total profit
  totalSpent  = 0 # tracks toatl buy amounts
  stockOn     = 0 # 0 if no stock, 1 if stock purchased INDICATIVE VARIABLE

  for (i in (timeLong+1):length(x[,1])){
    xShort <- x[-(i+1):-length(x[,1]),]
    if(gPredCrossMABuy(xShort) && stockOn == 0){
      buyAmt  = x[i,1]
      # keep track of total amount spent, only changes with a buy
      sellAmt = buyAmt #we set the sellAmt=buyAmt so we do not buy again until after sell
      totalSpent  = buyAmt + totalSpent
      stockOn = 1
    } else if(gPredCrossMASell(xShort) && stockOn == 1){
      sellAmt = x[i,1]
      profit  = sellAmt-buyAmt
      # keep track of total profit, only changes with a sell
      totalProfit = profit + totalProfit
      stockOn = 0
    } else if(i == length(x[,1])){
      # if we are still waiting to sell when period ends, we sell for current amount
      if (x[i,2]<buyAmt){ ###SPECIAL CONDITION FOR SHORT###
        totalProfit = totalProfit + (x[i,1]-buyAmt)
      } else{
        # nothing is needed as amount has been sold
      }
      message("NOTE: final balance sold to calculate ROI")
    } else{
      # nothing else needs to happen because we are waiting for a sell prediction
    }
  }
  message("MA SUMMARY & TOTALS:")
  message("Total Spent: ", totalSpent)
  message("Total Profit: ", totalProfit)
  message("Relative Profit (ROI): ", round(((totalProfit/totalSpent)*100),2), " %")
}
```

Working Examples

Example 1

```
fProfitDataMA(mAvgsAll(X,45,80,.1,.9),45,80)
```

```
## NOTE: final balance sold to calculate ROI
## 45-day TOTALS:
## Total Spent: 1203.110015
## Total Profit: 10.469972
## Relative Profit (ROI): 0.87 %
## NOTE: final balance sold to calculate ROI
## 80-day TOTALS:
## Total Spent: 1017.510009
## Total Profit: 3.50997999999998
## Relative Profit (ROI): 0.34 %
```

Example 2

```
fProfitCrossMA(mAvgsAll(X,20,40,.1,.9),20,40)
```

```
## NOTE: final balance sold to calculate ROI
## MA SUMMARY & TOTALS:
## Total Spent: 1439.610002
## Total Profit: -11.350008
## Relative Profit (ROI): -0.79 %
```