Quantitative Researcher Task

Ronan Anderson

02/08/2022

1) Trading Strategy Contruction- Day 1 & 2

We are using the data from exchange 1 due to this dataset having a smaller amount of rows (i.e., quicker to compile). The 3 days are loaded in and labelled as follows:

```
day1=read.csv("exchange1_20210519.csv.gz", header = FALSE, na.strings=c("","  ","NA"),
              stringsAsFactors=FALSE)
day2=read.csv("exchange1_20210520.csv.gz", header = FALSE, na.strings=c("","  ","NA"),
              stringsAsFactors=FALSE)
day3=read.csv("exchange1_20210521.csv.gz", header = FALSE, na.strings=c("","  ","NA"),
              stringsAsFactors=FALSE)
```

It is worth noting that day 2 picks up exactly where day 1 leaves off with regards to time. With this in mind, and with the trading strategy being formulated from both day 1 and day 2, we have combined these two files in the dataframe "newframe". The columns are then renamed respectively, and the date/time is formatted correctly. This isn't required as such for this task, but it is useful if we were to plot any of the data. We largely concern ourselves with the order Receiving Time as this is the time that any data would be available to the company on the trading engine.

```
newframe <- rbind(day1, day2)

colnames(newframe) <- c("Matching_Time", "Receiving_Time", "Symbol",
                        "Bid_Price_1", "Bid_Qty_1", "Ask_Price_1", "Ask_Qty_1",
                        "Bid_Price_2", "Bid_Qty_2", "Ask_Price_2", "Ask_Qty_2",
                        "Bid_Price_3", "Bid_Qty_3", "Ask_Price_3", "Ask_Qty_3",
                        "Bid_Price_4", "Bid_Qty_4", "Ask_Price_4", "Ask_Qty_4",
                        "Bid_Price_5", "Bid_Qty_5", "Ask_Price_5", "Ask_Qty_5",
                        "Bid_Price_6", "Bid_Qty_6", "Ask_Price_6", "Ask_Qty_6",
                        "Bid_Price_7", "Bid_Qty_7", "Ask_Price_7", "Ask_Qty_7",
                        "Bid_Price_8", "Bid_Qty_8", "Ask_Price_8", "Ask_Qty_8",
                        "Bid_Price_9", "Bid_Qty_9", "Ask_Price_9", "Ask_Qty_9",
                        "Bid_Price_10", "Bid_Qty_10", "Ask_Price_10", "Ask_Qty_10")

timeDate <- strptime(newframe$Receiving_Time, "%Y-%m-%dT%H:%M:%OS")
op <- options(digits.secs = 3)
newframe$Receiving_Time <- timeDate
```

The trading startegy implemented here is a Simple Moving Average Crossover. The moving average periods used are up to interpretation; however, we want the long term period to represent some form of longer term average Given that there are between 30 to 40 orders per second in this data, this period will have to be relatively large. To capture a half hour period, we would need approximately 60,000 to 70,000 data points. Therefore, we will use 60,000 for the long term MA and 15,000 for a short term MA, which will hopefully capture short-term deviations from the long-term averages.

Next, we create a dataframe with the relevant information. We use the lowest bid price and the highest ask price so that our strategy is more robust and does not rely on generous market spreads. We also lead both the bid and ask prices by 35 in order to combat the latency of 100ms (approximately 30 to 40 trades per 100ms).

```
df_ema <- data.frame('Time' = newframe$Receiving_Time,
                     'Bid' = lead(newframe$Bid_Price_10, 35),
                     'Ask' = lead(newframe$Ask_Price_10, 35),
                     'SMA15000' = SMA(newframe$Bid_Price_10, n=15000),
                     'SMA60000' = SMA(newframe$Bid_Price_10, n=60000))
```

With the moving averages calculated, we can implement the trading strategy as follows, where we buy BTC when the short-term MA crosses above the long-term MA, and sell when it crosses under.

```
df_ema[60000:length(newframe$Bid_Price_10),] %>%
  mutate(
    signal = case_when(
      SMA15000 >= SMA60000 ~ "buy",
      SMA15000 < SMA60000 ~ "sell"
    ),
    previous_signal = lag(signal, 1),
    decision = case_when(
      signal == previous_signal ~ "hold",
      TRUE ~ signal
    )
  ) -> df_with_decisions

df_with_decisions %>%
  filter(decision != 'hold') %>%
  select(Time, Bid, Ask, SMA15000, SMA60000, decision) -> df_bought_sold

head(df_bought_sold, n=4)
```

```
##                      Time   Bid   Ask SMA15000 SMA60000 decision
## 1 2021-05-19 00:39:45.289 42822 42849 42909.05 43198.12     sell
## 2 2021-05-19 02:40:56.767 41109 41149 40947.69 40947.69      buy
## 3 2021-05-19 02:52:31.892 40762 40790 40939.37 40939.38     sell
## 4 2021-05-19 03:21:52.750 41001 41034 40687.16 40687.16      buy
```

Lastly, we can backtest the trading strategy with $100 as given. Note that the transaction fee of 1bp is also accounted for in the "amount" variable. The transactions and their values are stored in the dataframe "df_action". This strategy is a long only strategy, and therefore if a sell signal arises as the first signal, its transaction fee is ignored and it is removed from the data frame "df_action".

```
total <- 100
base <- 100
df_action <- data.frame(Action = NA)
for(i in 1:nrow(df_bought_sold)){

  if(df_bought_sold[i,]$decision == "buy"){

    price_bought <- df_bought_sold[i,]$Ask
    if(i == 2){total <- base}
    amount <- total - (total * 0.0001)
    shares <- amount/price_bought
    total <- price_bought * shares
    df_action[i, 'Action'] <- paste0("BUY: ", round(shares, digits=7), " shares at $",
```

```
                                    price_bought, " for $", round(total, 4))

  } else if(total > 0 && df_bought_sold[i,]$decision == "sell"){

    price_sold <- df_bought_sold[i,]$Bid
    if(i == 1){shares <- base/price_sold}
    total <- shares * price_sold
    total <- total - (total * 0.0001)
    df_action[i, 'Action'] <- paste0("SELL: ", round(shares, digits=7), " shares at $",
                                    price_sold, " for $", round(total, 4))

  }
}

if(df_bought_sold[1,]$decision == "sell"){df_action <- df_action[-1,]}

head(df_action, n=3)
```

```
## [1] "BUY: 0.0024299 shares at $41149 for $99.99"
## [2] "SELL: 0.0024299 shares at $40762 for $99.0397"
## [3] "BUY: 0.0024134 shares at $41034 for $99.0298"
```

```
cat("\n")
```

```
tail(df_action, n=3)
```

```
## [1] "BUY: 0.0022062 shares at $41670 for $91.9325"
## [2] "SELL: 0.0022062 shares at $41088 for $90.6394"
## [3] "BUY: 0.0022145 shares at $40925 for $90.6303"
```

```
cat("\n")
```

```
total_return <- total/base
print(paste0("Total Return: ", round(total_return*100, 3)-100, "%"))
```

```
## [1] "Total Return: -9.37%"
```

By trading across the 48 hour period of day 1 and day 2, the strategy has executed 94 trades for a return of -9.37%.

2) Trading Strategy Testing - Day 3

This strategy is now embedded into a function named "maco_return" so that it can be easily called and tested on the day 3 data, or on any other data with the same format as the order books.

```
maco_return <- function(data) {

  colnames(data) <- c("Matching_Time", "Receiving_Time", "Symbol",
                          "Bid_Price_1", "Bid_Qty_1", "Ask_Price_1", "Ask_Qty_1",
                          "Bid_Price_2", "Bid_Qty_2", "Ask_Price_2", "Ask_Qty_2",
                          "Bid_Price_3", "Bid_Qty_3", "Ask_Price_3", "Ask_Qty_3",
                          "Bid_Price_4", "Bid_Qty_4", "Ask_Price_4", "Ask_Qty_4",
                          "Bid_Price_5", "Bid_Qty_5", "Ask_Price_5", "Ask_Qty_5",
                          "Bid_Price_6", "Bid_Qty_6", "Ask_Price_6", "Ask_Qty_6",
                          "Bid_Price_7", "Bid_Qty_7", "Ask_Price_7", "Ask_Qty_7",
                          "Bid_Price_8", "Bid_Qty_8", "Ask_Price_8", "Ask_Qty_8",
                          "Bid_Price_9", "Bid_Qty_9", "Ask_Price_9", "Ask_Qty_9",
                          "Bid_Price_10", "Bid_Qty_10", "Ask_Price_10", "Ask_Qty_10")

  timeDate <- strptime(data$Receiving_Time, "%Y-%m-%dT%H:%M:%OS")
  op <- options(digits.secs = 3)
  data$Receiving_Time <- timeDate

  df_ema <- data.frame('Time' = data$Receiving_Time,
                       'Bid' = lead(data$Bid_Price_10, 35),
                       'Ask' = lead(data$Ask_Price_10, 35),
                       'SMA15000' = SMA(data$Bid_Price_10, n=15000),
                       'SMA60000' = SMA(data$Bid_Price_10, n=60000))



  df_ema[60000:length(data$Bid_Price_10),] %>%
    mutate(
      signal = case_when(
        SMA15000 >= SMA60000 ~ "buy",
        SMA15000 < SMA60000 ~ "sell"
      ),
      previous_signal = lag(signal, 1),
      decision = case_when(
        signal == previous_signal ~ "hold",
        TRUE ~ signal
      )
    ) -> df_with_decisions

  df_with_decisions %>%
    filter(decision != 'hold') %>%
    select(Time, Bid, Ask, SMA15000, SMA60000, decision) -> df_bought_sold

  head(df_bought_sold)



  total <- 100
  base <- 100
```

```
  df_action <- data.frame(Action = NA)
  for(i in 1:nrow(df_bought_sold)){

    if(df_bought_sold[i,]$decision == "buy"){

    price_bought <- df_bought_sold[i,]$Ask
    if(i == 2){total <- base}
    amount <- total - (total * 0.0001)
    shares <- amount/price_bought
    total <- price_bought * shares
    df_action[i, 'Action'] <- paste0("BUY: ", round(shares, digits=7), " shares at $",
                                     price_bought, " for $", round(total, 4))

  } else if(total > 0 && df_bought_sold[i,]$decision == "sell"){

    price_sold <- df_bought_sold[i,]$Bid
    if(i == 1){shares <- base/price_sold}
    total <- shares * price_sold
    total <- total - (total * 0.0001)
    df_action[i, 'Action'] <- paste0("SELL: ", round(shares, digits=7), " shares at $",
                                     price_sold, " for $", round(total, 4))

  }
}

  if(df_bought_sold[1,]$decision == "sell"){df_action <- df_action[-1,]}

  total_return <- total/base
  return(print(paste0("Total Return: ", round(total_return*100, 3)-100, "%")))
}


maco_return(day3)
```

```
## [1] "Total Return: -12.474%"
```

3) Additional Qs

1. How is your approach affected by a change in position size, exchange fee, or latency?

The position size has no altering affect on the strategy apart from increasing/decreasing the transaction cost. If $10,000 is invested instead of $100, such that total and base now equal 10000, the return printed will be identical. Naturally, the exchange fee will continue at 1bp of the position, and will be larger. With regards to latency, there are no visible affects through the code; however, extremely large orders can take longer to fill and may therefore, have a larger latency.

2. What are the highest/lowest limits that can be assumed for position size, exchange fee, and latency in order for your approach to work?

The algorithm itself should remain unaffected by a change in any of these factors. However, by dropping the exchange fee to 0bps, the return for day 3 is shown as -12.026% as opposed to -12.474% when the exchange fee is 1bp. By dropping the latency to 0 ms, the return for day 3 is shown as -12.829% in comparison to -12.474%, so the latency doesn't seem to have a significant effect.

3. Are there any mechanisms to increase/decrease the trading frequency? What are the trade-offs of these changes?

One method which we attempted to incorporate into the strategy was to only utilise the MA crossover when the instrument was in an up trend. Many people report that in this instance, the strategy is more profitable. However, this was not feasible due to the the length of time required for such a loop to compile. This code is given below, where it only executes the MA crossover if the slope of the regression is positive.

```
#for (i in 60000:length(data$Bid_Price_10)) {
#  ts <- ts(newframe$Bid_Price_1)
#  training <- window(ts, start=i-1999, end=i)
#  ti <- as.vector(time(training))
#  M1 <- lm(training ~ ti)
#  coeff <- M1$coefficients[2]

#  if (coeff > 0) {
#    df_ema[i,] %>%
#      mutate(
#      signal = case_when(
#        SMA15000 >= SMA60000 ~ "buy",
#        SMA15000 < SMA60000 ~ "sell"
#      ),
#      previous_signal = lag(signal, 1),
#      decision = case_when(
#        signal == previous_signal ~ "hold",
#        TRUE ~ signal
#      )
#    ) -> df_with_decisions
#  }
#}
```

Another method of slowing the trade frequency could be to use a data frequency other than milliseconds. For example, many people like to use the MA crossover strategy of a data frequency of 5 mins.

4. How can we create a robust and reusable pipeline for research?

In terms of robustness, it is important to build the trading strategy based off of a relatively large spread. This ensures that even when spreads widen and less than ideal bid/offer prices are given, the strategy is still robust against this risk element. It is also important for the strategy code to compile quickly, as any time lost essentially increases the strategy's latency. For risk/robustness purposes, it is worth building stop limits into the above code, which could adjust to the current value of the portfolio.

A reusable strategy can be created as done above, where a function was implemented and can accept any data from an order book of the same format. Additionally, this strategy can be easily altered from an MA crossover to another strategy by only editing a couple of lines of code.

5. How can overfitting be prevented, so we can be confident that the strategy will perform well on live data?

One key method of preventing overfitting is to test the same strategy with the same parameters on many different data periods, which will test the strategy's versatility (cross-validation). You should also create the model based off the training set and test it on a separate test set. If the accuracy of the model on the test set has significantly decreased, then there is likely an issue of overfitting.

A very good example of overfitting is sculpting an ARIMA model to a certain dataset using many parameters because they achieve more accurate MAPE/RMSE/etc. scores. A great way of overcoming this is to use Information Criterion to select a more parsimonious model, which will also allow the model to be more universal. Given below is a chunk of code that I used in my undergraduate thesis on time series, which prints out the 7 models with the lowest AIC (d=1 is already set as the non-stationarity of my data had already been established).

```
#p <- c(rep(0,7),rep(1,7),rep(2,7),rep(3,7),rep(4,7),rep(5,7),rep(6,7))
#q <- rep(0:6, 7)
#frame <- data.frame(p, q, AIC = NA)
#for (i in 1:49) {
  #p.value <- frame[i,"p"]
  #q.value <- frame[i,"q"]
  #fit.aic <- AIC(Arima(data, order=c(p.value,1,q.value)))
  #frame[i,"AIC"] <- fit.aic
#}
#newdata <- frame[order(frame$AIC),]
#head(newdata,7)
```

6. What optimisation techniques can we take advantage of to try to make the strategy better?

Naturally, this varies based on the type of model being implemented. If parameters are required in something like a time series model, then a loop as shown in Q5 could be implemented. Many technical strategies such as an MA crossover or RSI indicator wouldn't require much optimisation in this regard. However, a hybrid version such as the one mentioned in Q3, which considers a trend line could be included and this would require more optimisation than normal. Many issues around this area can be resolved with iterative loops that search for the most suitable model.

7. What are the best methods of interpreting and/or visualizing the results?

This could be done quite easily with the above code given that the each trade and its value is already stored in the "df_action" data frame. The 'Receiving Time' dates/times are also already formatted correctly into R and are ready to be plotted if needed. So, if the times of each trade and the value at that time were pulled into a data frame, it could be plotted to show the value of the initial investment after each trade across the whole day.

The result at the end of the day is also printed from the above code.