COMP 226: Assignment 2

Rahul Savani

rahul.savani@liverpool.ac.uk

Assignment 2

Continuous Assessment Number	2 (of 2)
Weighting	10%
Assignment Circulated	Thursday 18 March 2020 (week 8)
Deadline	17:00 Friday 1st May 2020 (end of week 11)
Submission Mode	Electronic only

Full details of the assignment can be found in the handout:

https://www2.csc.liv.ac.uk/~rahul/teaching/comp226/_downloads/a2.pdf

Summary of Assessment

- **Goal**: implement and optimize a well-defined trading strategy within the `backtester_v5.3` framework.
- **Assessed via automatic testing of 6 functions that you need to implement**
- The input and output behaviour of each function is fully specified, with examples
- A code template is provided as a starting point

Marking Criteria

- **Individual marks are available for each of 6 functions** that should be implemented
- If all 6 function implementations **pass all the automated tests then a mark of 100% will be achieved**
- **Partial credit for a function may be awarded** if some but not all automated tests for that function are passed
- **Marks for some functions will only be awarded if other functions are correctly implemented**

Late Submission Penalty

- Standard UoL policy
- Note that **no resubmissions after the deadline will be considered**

Getting started with the backtester framework

- **Download** `backtester_v5.3.zip` and **unzip it**
- Go to the resulting `backtester_v5.3` directory, open R, and set this directory as the working directoty (e.g. using `setwd` if required)
- Now you can run `main.R`, which allows you to run a number of example strategies

```
source('main.R')
```

`main.R`

We go through `main.R` in order and see what each part does:

1. Sourcing the framework and example strategies
2. Loading data
3. Loading an example strategy
4. Subsetting the data
5. Running a backtest

Sourcing the framework and example strategies

```
source('framework/data.R')
source('framework/backtester.R')
source('framework/processResults.R')
source('framework/utilities.R')
source('example_strategies.R')
```

Loading data

```
# load data
dataList <- getData(directory="EXAMPLE")
```

```
> length(dataList)
[1] 5

> for (x in dataList) print(class(x))
[1] "xts" "zoo"
[1] "xts" "zoo"
[1] "xts" "zoo"
[1] "xts" "zoo"
[1] "xts" "zoo"
```

Inspecting the data

All the series have the same start and end dates:

```
> for (x in dataList) print(paste(start(x),end(x)))
[1] "1970-01-02 1973-01-05"
[1] "1970-01-02 1973-01-05"
[1] "1970-01-02 1973-01-05"
[1] "1970-01-02 1973-01-05"
[1] "1970-01-02 1973-01-05"
```

```
> head(dataList[[1]],n=2)
             Open   High    Low  Close Volume
1970-01-02 0.7676 0.7698 0.7667 0.7691   3171
1970-01-03 0.7689 0.7737 0.7683 0.7729   6311
```

Example strategies

example_strategies.R:

```
example_strategies <- c("fixed",
                        "big_spender",
                        "bankrupt",
                        "copycat",
                        "random",
                        "rsi_contrarian",
                        "bbands_trend_following",
                        "bbands_contrarian",
                        "bbands_holding_period",
                        "simple_limit")
```

Loading an example strategy

```
# choose strategy from example_strategies
strategy <- "fixed"

# check that the choice is valid
is_valid_example_strategy <- function(strategy) {
    strategy %in% example_strategies
}
stopifnot(is_valid_example_strategy(strategy))

# load in strategy and params
load_strategy(strategy) # function from example_strategies.R
```

load_strategy

```
load_strategy <- function(strategy) {
    # load strategy
    strategyFile <- file.path('strategies', paste0(strategy,'.R'))
    cat("Sourcing",strategyFile,"\n")
    source(strategyFile) # load in getOrders
    # set params via global assignment
    params <<- example_params[[strategy]]
    print("Parameters:")
    print(params)
}
```

fixed.R

```
getOrders <- function(store, newRowList, currentPos, info, params) {
    allzero      <- rep(0,length(newRowList))
    marketOrders <- allzero
    if (is.null(store)) {
        # take position during first iteration and hold
        marketOrders <- params$sizes
        store <- 1 # not null
    }
    return(list(store=store,marketOrders=marketOrders,
                            limitOrders1=allzero,
                            limitPrices1=allzero,
                            limitOrders2=allzero,
                            limitPrices2=allzero))
}
```

getOrders

In the backtester framework **all strategies are implemented in a function called** `getOrders`, which uses arguments are the same for all strategies:

```
getOrders <- function(store,newRowList,currentPos,info,params)
```

- `store`: contains all data you save from one period to the next
- `newRowList`: new day's data (a list of rows from the series)
- `currentPos`: the vector of current positions in each series
- `info`: not needed for this assignment
- `params`: a list of parameters (needed for parameter optimization)

Subsetting the data

For assignment 2 you will need to subset the data There is an example in `main.R`:

```
inSampDays <- 200
# in-sample period
dataList <- lapply(dataList, function(x) x[1:inSampDays])
```

So we are only using the first 200 days. You will need to do something similar for assigment 2.

Running a backtest

```
# Do backtest
results <- backtest(dataList,getOrders,params,sMult=0.2)
pfolioPnL <- plotResults(dataList,results)
```

Getting the Profit Drawdown (PD) Ratio

In assignment 2 you will need to optimize the PD ratio.

The the PD ratio is stored in `pfolioPnL$fitAgg`, e.g., for the first example in the assignment 2 handout we have:

```
> print(pfolioPnL$fitAgg)
[1] -153.44
```

Market orders

- Recall that market orders specify volume and direction
- In the backtester framework, trading decisions are made after the close of day `k`, and trades are executed on day `k+1`.
- For each day, the framework supports `one market order` for each series, and `two limit orders` for each series.
- These orders are returned from `getOrders` as follows:

```
return(list(store=store,marketOrders=marketOrders,
                        limitOrders1=limitOrders1,
                        limitPrices1=limitPrices1,
                        limitOrders2=limitOrders2,
                        limitPrices2=limitPrices2))
```

Market orders - example

Market orders will be `executed at the open` on day `k+1`. They incur **slippage** (20% of the overnight gap for assignment 2). Market orders are specified by

- `size` (number of units to trade)
- `direction` (buy/sell)

The `sizes` and `directions` of market orders are encoded in the vector `marketOrders` of the return list of `getOrders`. E.g.

```
c(0,-5,0,1,0)
```

means place a market order for 5 units short in series 2, and 1 unit long in series 4.

Limit orders

- We will not use limit orders for assignment 2.
- You can leave `limitOrders1`, `limitPrices1`, `limitOrders2`, `limitPrices2` as zero vectors when you do assignment 2 (i.e., you do not need to edit that part of the template).
- We will introduce the limit order functionality in detail in COMP396.

The code template and data for assignment 2

You are now ready to start working on assignment 2. To do so you should read and work through the rest of this document very carefully.

As a first step, try to run `main_template.R`. It uses `strategies/a2_template.R` which is a code template that should serve as your starting point. If you try to source `main_template.R` you will get an error as follows:

```
Error in if (store$iter > params$lookbacks$long) { :
   argument is of length zero
```

When the strategy is fully implemented it will require a parameter called `lookbacks`.

6 functions to be implemented

1. `getTMA`
2. `getPosSignFromTMA`
3. `getPosSize`
4. `getOrders`
5. `getInSampleResult`
6. `getInSampleOptResult`

Part 1: strategy implementation

- The strategy uses three moving averages with three different lookbacks (window lengths).
- The short lookback should be smaller than the medium window, which in turn should be smaller than the long lookback.
- In every trading period, the strategy will compute the value of these three moving averages. You will achieve this be completing the implementation of the function `getTMA`.

getPosSignFromTMA

The system is out of the market (i.e., flat) when the relationship between the short moving average and the medium moving average does not match the relationship between the medium moving avergage and long moving average.

MA		MA		MA	Position
short	>	medium	>	long	short
short	<	medium	<	long	long

The function `getPosSignFromTMA` should use a function `getTMA`. The position size, i.e., the number of units to be long or short, will be determined by the function `getPosSize`.

`getTMA` **(30%)**

Input parameters	Expected behaviour
`close_prices`; `lookbacks`. The specific form that these arguments should take is specified in the template code via the 6 checks that you need to implement.	You should first implement the checks as described in the template. Hints of how to implement them are given below. The function should return a list with three named elements (named short, medium, and long). Each element should be equal to the value of a simple moving average with the respective window size as defined by `lookbacks`. The windows should all end in the same period, which should be the final row of `close_prices`.

`getPosSignFromTMA` **(15%)**

Input parameters	Expected behaviour
`tma_list` is a list with three named elements, short, medium, and long. These correspond to the simple moving averages as returned by `getTMA`. **Note**: You do not need to check the validity of the function argument in this case, or for the remaining functions either.	This function should return either 0, 1, or -1. If the short value of `tma_list` is less than the medium value, and the medium value is less than the long value, it should return 1 (indicating a long position). If the short value of `tma_list` is greater than the medium value, and the medium value is greater than the long value, it should return -1 (indicating a short position). Otherwise, the return value should be 0 (indicating a flat position).

`getPosSize` **(5%)**

Input parameters	Expected behaviour
`current_close`: this is the current close for one of the series; it does not have a default value. `constant`: this argument should have a default value of 1000.	The function should return (`constant` divided by `current_close`) rounded **down** to the nearest integer.

`getOrders` **(20%)**

Input parameters	Expected behaviour
The arguments to this function are always the same for all strategies used in the backtester framework.	This function should implement the strategy outlined above and again below in "Strategy specification".

Strategy specification

- apply the following logic independently for both series.
- in the (`params$lookbacks$long` + 1)-th period, and in every period after, the strategy computes three simple moving averages with `getTMA`, with windows that all end in the current period, and with window lengths equal to:
    - `params$lookbacks$short`
    - `params$lookbacks$medium`
    - `params$lookbacks$long`
- send **market orders** to assume a new position according to `getPosSignFromTMA` and `getPosSize`

Hints

For the checks for `getTMA` note:

- The operator `!` means not, and can be used to negate a boolean.
- `sapply` allows one to apply a function element-wise to a vector or list (e.g., to a vector list `c("short","medium","long")`).
- `all` is a function that checks if all elements of a vector are true (for example, it can be used on the result of `sapply`).
- `%in%` can be used to check if a element exists inside a vector.

Hints

To compute the moving average in `getTMA` you can use the function `SMA` from the `TTR` package.

Note: The list returned by `getTMA` should work as input to the function `getPosSignFromTMA`.

For `getPosSize`, to round **down** to the nearest integer you can use the function `floor`.

As in the template, use the negative of `currentPos` summed with the new positions you want to take to make sure that you assume the correct position.

Example output for `getTMA`

Here are 3 examples of expected behaviour for the checks:

```
> close_prices <- c(1,2,3)
> lookbacks <- list(short=as.integer(5),
                    medium=as.integer(10),
                    long=as.integer(20))
> getTMA(close_prices,lookbacks) # bad close_prices
Error in getTMA(close_prices, lookbacks) :
  E04: close_prices is not an xts according to is.xts()

> dataList <- getData(directory="A2")
Read 2 series from DATA/A2
> close_prices <- dataList[[1]]$Close[1:19]
> getTMA(close_prices,lookbacks) # bad close_prices; too short
Error in getTMA(close_prices, lookbacks) :
  E05: close_prices does not enough rows

> lookbacks <- list(5,10,25)
> getTMA(close_prices,lookbacks) # bad lookbacks; list elements not named
Error in getTMA(close_prices, lookbacks) :
  E01: At least one of "short", "medium", "long" is missing from names(lookbacks)
```

Example output for `getTMA`

Here is an example where we give the function valid arguments.

```
> lookbacks <- list(short=as.integer(5),
                    medium=as.integer(10),
                    long=as.integer(20))
> close_prices <- dataList[[1]]$Close[1:20]
> getTMA(close_prices,lookbacks)
$short
[1] 169

$medium
[1] 170.4

$long
[1] 171.05
```

Example output for `getPosSignFromTMA`

Here are three examples of correct output:

```
> getPosSignFromTMA(list(short=10,medium=20,long=30))
[1] 1
> getPosSignFromTMA(list(short=10,medium=30,long=20))
[1] 0
> getPosSignFromTMA(list(short=30,medium=20,long=10))
[1] -1
```

Example output for `getPosSize`

Here are two examples of correct output:

```
> current_close <- 100.5
> getPosSize(current_close)
[1] 9
> getPosSize(current_close,constant=100.4)
[1] 0
```

Example output for `getOrders`

To check your implementation of `getOrders`, see part 2 for examples of correct output for `getInSampleResult`

Part 2: in-sample tests

- There are two more functions that you need to implement: `getInSampleResult` and `getInSampleOptResult`; both return a single number
- For both functions you will need to compute **your own** in-sample period, which is based on your MWS username (so for part 2 there are different answers for different students)
- 0 marks may be given for these two functions if your implementation of `getOrders` is not correct
- **Do NOT includde the code to compute the return value** (as that code will take some time to run). Instead compute the correct number with your own, unsubmitted code, and then just enter the correct return value into the template for these two functions

In-sample periods

```
> source('in-sample_period.R')
> getInSamplePeriod('x4xz1')
[1]  230 644
```

- To get your in-sample period, use `in-sample_period.R`
- Source it and run the function `getInSamplePeriod` with your MWS username
- Use the first number in the returned vector as the start of the in-sample period and the second number as the end
- **Note**: you may need to install the package `digest`

`getInSampleResult` **(10%)**

Expected behaviour
This function should return the PD ratio that is achieved when the strategy is run with short lookback 10, medium lookback 20, and long lookback 30, on your username-specific in-sample period. The function should not contain **ANY** code except the return value; it should run and complete instantanously.

0 marks will be given if:

- function contains any code other than the return statement
- you do not submit a correct implemetation of` `getOrders`

`getInSampleOptResult` **parameter optimization**

You need to do an in-sample parameter optimization using the following parameter combinations for the:

- short lookback
- medium lookback
- long lookback

You should **not** optimize the constant used with `getPosSize`, and leave it as 1000 as defined in the template code.

Parameter combinations

Defined by two things: **parameter ranges** and a **further restriction** The ranges are:

Parameter	Minimum value	Increment	Maximum Value
short lookback	100	5	110
medium lookback	105	5	120
long lookback	110	5	130

The further restriction is the following:

Further restriction on parameter values

You should further restrict the parameter combinations as follows:

- The medium lookback should always be strictly greater than the short lookback.
- The long lookback should always be strictly greater than the medium lookback.

Hint

- There should be 28 parameter combinations
- You can (easily) adapt `backtester_v5.3/main_optimize.R`
- It is probably easiest to use three nested for loops in order to ensure that you only check valid parameter combinations (where the short < medium < long for the respective window lengths)

`getInSampleOptResult` **(20%)**

Expected behaviour
This function should return the best PD ratio than can be achieved with the stated allowable parameter combinations on your username-specific in-sample period. The function should not contain **ANY** code except the return value; it should run and complete instantanously.

0 marks will be given if:

- function contains any code other than the return statement
- you do not submit a correct implemetation of` `getOrders`

Example output for `getInSampleResult`

To help you check the correctness of your code, here are three example return values for made up usernames:

Username	Correct return value
x1xxx	-747.6
x1yyy	-231.6
x1zzx	-639.8

Example output for `getInSampleOptResult`

To help you check the correctness of your code, here are three example return values for made up usernames:

Username	Correct return value
x1xxx	4.23
x1yyy	3.42
x1zzx	4.43

Marks summary

Function	Marks
`getTMA`	30
`getPosSignFromTMA`	15
`getPosSize`	5
`getOrders`	20
`getInSampleResult`	10 (0 if `getOrders` does not work)
`getInSampleOptResult`	20 (0 if `getOrders` does not work)

Submission

- Submit **a single R file** that contains your implementation of 6 functions
- The file shoud be called MWS-username.R where you should replace MWS-username by your MWS username
- For example, if your username is "abcd" then you should submit a file named "abcd.R"
- Submission is via the department electronic submission system: http://www.csc.liv.ac.uk/cgi-bin/submit.pl

Warning

Your code will be put through the department's automatic plagiarism and collusion detection system. Student's found to have plagiarized or colluded will likely receive a mark of zero. Do not show your work to other students.