Portfolio Backtesting

Daniel P. Palomar and Rui ZHOU
Hong Kong University of Science and Technology (HKUST)
2018-09-20

Contents

This vignette illustrates the usage of the package `portfolioBacktest` for automated portfolio backtesting. It can be used by a researcher/practitioner to check a set of different portfolios, as well as by a course instructor to evaluate the students in their portfolio design in a fully automated and convenient manner.

1 Installation

The package can currently be installed from GitHub:

```
# install.packages("devtools")
devtools::install_github("dppalomar/portfolioBacktest")

# Getting help
library(portfolioBacktest)
help(package = "portfolioBacktest")
package?portfolioBacktest
?portfolioBacktest
```

2 Usage of the package

2.1 Loading data

We start by loading the package and some random sets of stock market data:

```
library(xts)
library(portfolioBacktest)
data(prices)
```

The dataset `prices` is a list of objects `xts` that contains the prices of random sets of stock market data from the S&P 500, HSI, NKY, SHZ, and UKC, over random periods of two years with a random selection of 50 stocks of each universe.

```
length(prices)
#> [1] 50
str(prices[[1]])
#> An 'xts' object on 2008-11-27/2010-11-02 containing:
#>   Data: num [1:504, 1:50] 15.7 15.4 14.7 14.9 15.8 ...
#>  - attr(*, "dimnames")=List of 2
#>   ..$ : NULL
#>   ..$ : chr [1:50] "MSCI UN Equity" "MNST UW Equity" "LKQ UW Equity" "UDR UN Equity" ...
#>   Indexed by objects of class: [Date] TZ: UTC
#>   xts Attributes:
#>  NULL

colnames(prices[[1]])
#>  [1] "MSCI UN Equity" "MNST UW Equity" "LKQ UW Equity"  "UDR UN Equity"
#>  [5] "LB UN Equity"   "MS UN Equity"   "IFF UN Equity"  "TMO UN Equity"
#>  [9] "BIIB UW Equity" "NOC UN Equity"  "CPB UN Equity"  "VMC UN Equity"
#> [13] "ULTA UW Equity" "NVDA UW Equity" "FAST UW Equity" "WMB UN Equity"
#> [17] "VRTX UW Equity" "EBAY UW Equity" "RHI UN Equity"  "XRAY UW Equity"
#> [21] "GRMN UW Equity" "ALGN UW Equity" "FTI UN Equity"  "NBL UN Equity"
#> [25] "LLY UN Equity"  "FIS UN Equity"  "L UN Equity"    "STT UN Equity"
#> [29] "CVX UN Equity"  "IR UN Equity"   "PKG UN Equity"  "CDNS UW Equity"
#> [33] "XLNX UW Equity" "JCI UN Equity"  "IBM UN Equity"  "VRSN UW Equity"
#> [37] "WFC UN Equity"  "SIVB UW Equity" "PM UN Equity"   "ZBH UN Equity"
#> [41] "RTN UN Equity"  "CINF UW Equity" "ALXN UW Equity" "UTX UN Equity"
#> [45] "AAPL UW Equity" "ADM UN Equity"  "BBY UN Equity"  "AMZN UW Equity"
#> [49] "MRO UN Equity"  "IPGP UW Equity"
```

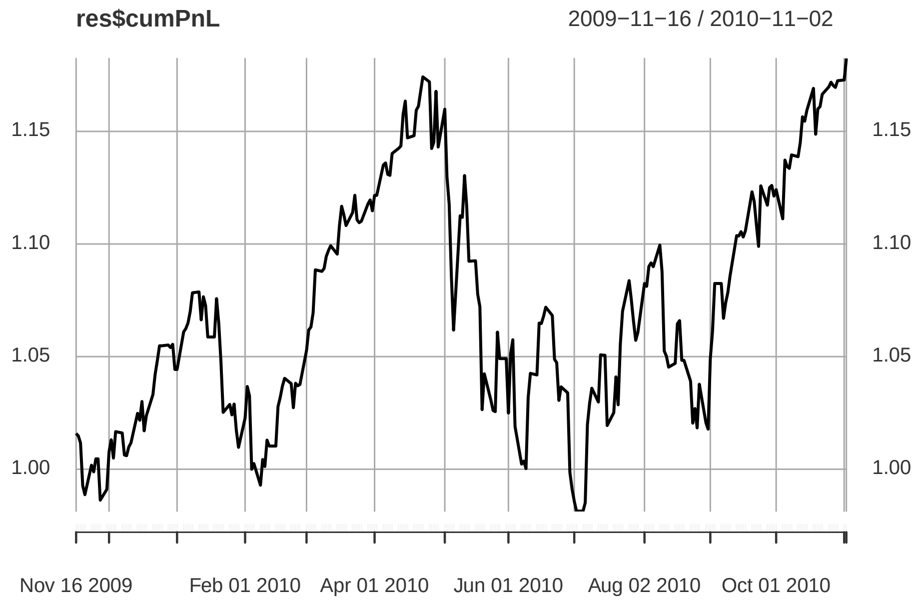
2.2 Backtesting a single portfolio

We start by defining a simple portfolio design in the form of a function that takes as input the prices and outputs the portfolio vector `w`:

```
uniform_portfolio_fun <- function(prices) {
  N <- ncol(prices)
  w <- rep(1/N, N)  # satisfies the constraints w>=0 amd sum(w)=1
  return(w)
}
```

Now we are ready to use the function `backtestPortfolio()` that will execute and evaluate the portfolio design function on a rolling-window basis:

```
res <- portfolioBacktest(uniform_portfolio_fun, prices[[1]])
names(res)
#> [1] "returns"       "cumPnL"        "performance"   "cpu_time"
#> [5] "error"         "error_message"
plot(res$cumPnL)
```




```
res$performance
#>    sharpe ratio    max drawdown expected return      volatility
#>       0.9281714       0.1641935       0.1823414       0.1964523
```

Let's try with a slightly more sophisticated portfolio design, like the global minimum variance portfolio (GMVP):

```
GMVP_portfolio_fun <- function(prices) {
  X <- diff(log(prices))[-1]  # compute log returns
  Sigma <- cov(X)  # compute SCM
  # design GMVP
  w <- solve(Sigma, rep(1, nrow(Sigma)))
  w <- w/sum(abs(w))  # it may not satisfy w>=0
  return(w)
}
res <- portfolioBacktest(GMVP_portfolio_fun, prices[[1]])
res$error
#> [1] TRUE
res$error_message
#> [1] "No-shortselling constraint not satisfied."
```

Indeed, the GMVP does not satisfy the no-shortselling constraint. We can repeat the backtesting indicating that shortselling is allowed:

```
res <- portfolioBacktest(GMVP_portfolio_fun, prices[[1]], shortselling = TRUE)
res$error
#> [1] FALSE
res$error_message
#> NULL
res$cpu_time
#> [1] 0.01
res$performance
```

```
#>    sharpe ratio    max drawdown expected return      volatility
#>      0.90688642      0.02956825      0.04228335      0.04662475
```

We could be more sophisticated and design a Markowitz mean-variance portfolio satisfying the no-shortselling constraint:

```
library(CVXR)  #install.packages("CVXR")

Markowitz_portfolio_fun <- function(prices) {
  X <- diff(log(prices))[-1]  # compute log returns
  mu <- colMeans(X)  # compute mean vector
  Sigma <- cov(X)  # compute the SCM
  # design mean-variance portfolio
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - 0.5*quad_form(w, Sigma)),
                  constraints = list(w >= 0, sum(w) == 1))
  result <- solve(prob)
  return(as.vector(result$getValue(w)))
}
```

We can now backtest it:

```
res <- portfolioBacktest(Markowitz_portfolio_fun, prices[[1]])
res$error
#> [1] FALSE
res$error_message
#> NULL
res$cpu_time
#> [1] 3.06
res$performance
#>    sharpe ratio    max drawdown expected return      volatility
#>       1.0455423       0.2064017       0.4299648       0.4112362
```

Instead of backtesting a portfolio on a single `xts` dataset, it is more meaningful to backtest it on multiple datasets. This can be easily done simply by passing a list of `xts` objects:

```
res <- portfolioBacktest(Markowitz_portfolio_fun, prices[1:5])
names(res)
#> [1] "returns"             "cumPnL"              "performance"
#> [4] "performance_summary" "cpu_time"            "cpu_time_average"
#> [7] "failure_ratio"       "error"               "error_message"
res$cpu_time
#> [1] 3.06 2.99 3.13 2.96 2.95
res$performance
#>                 dataset 1 dataset 2  dataset 3 dataset 4 dataset 5
#> sharpe ratio    1.0455423 1.8606698 0.07968131 0.3943176 2.7912812
#> max drawdown    0.2064017 0.3394737 0.30534461 0.2325974 0.1565759
#> expected return 0.4299648 1.1647991 0.02459070 0.1280147 0.9004461
#> volatility      0.4112362 0.6260106 0.30861315 0.3246489 0.3225924
```

In particular, note the additional elements in the returned list:

```r
res$cpu_time_average
#> [1] 3.018
res$performance_summary
#>    sharpe ratio (median)    max drawdown (median) expected return (median)
#>                1.0455423                0.2325974                0.4299648
#>      volatility (median)
#>                0.3246489
res$failure_ratio
#> [1] 0
```

2.3 Backtesting multiple portfolios

Backtesting multiple portfolios is equally simple. It suffices to pass a list of functions to the backtesting function `multiplePortfolioBacktest()`:

```r
res <- multiplePortfolioBacktest(portfolio_fun_list = list(uniform_portfolio_fun,
                                                           GMVP_portfolio_fun),
                                 prices =  prices[1:5], shortselling = TRUE)
#> 2018-09-20 16:53:17 - Execute func1
#> 2018-09-20 16:53:17 - Execute func2
res
#> $performance_summary
#>       sharpe ratio  (median) max drawdown  (median)
#> func1              1.6113535             0.10145617
#> func2              0.9221429             0.03029658
#>       expected return  (median) volatility  (median)
#> func1                0.18234136           0.14552105
#> func2                0.04228335           0.04621708
#>
#> $cpu_time_average
#> func1 func2
#> 0.000 0.016
#>
#> $failure_ratio
#> func1 func2
#>     0     0
#>
#> $error_message
#> $error_message$func1
#> list()
#>
#> $error_message$func2
#> list()
```

3 Usage for grading students in a course

If an instructor wants to evaluate the students of a course in their portfolio design, it can also be done very easily. It suffices to ask each student to submit a .R script (named `LASTNAME-firstname-STUDENTNUMBER-XXXX.R`) containing the portfolio function called exactly `portfolio_fun()` as well as any other auxiliary functions that it may require (needless to say that the required packages should be loaded in that script with `library()`). Then the instructor can put all those files in a folder and evaluate all of them at once.

```
res_all_students <- multiplePortfolioBacktest(folder_path = "folder_path",
                                              prices =  prices[1:3])
#> 2018-09-20 16:53:18 - Execute code from Firstname1 Surname1 (00000001)
#> 2018-09-20 16:53:20 - Execute code from Firstname2 Surname2 (00000002)
#> 2018-09-20 16:53:30 - Execute code from Firstname3 Surname3 (00000003)
res_all_students$performance_summary
#>          sharpe ratio  (median) max drawdown  (median)
#> 00000001             1.5003370             0.1591520
#> 00000002            -0.5376190             0.3260651
#> 00000003             0.4112669             0.1084156
#>          expected return  (median) volatility  (median)
#> 00000001                0.36604788            0.2319558
#> 00000002               -0.11097632            0.2064219
#> 00000003                0.06697449            0.1515797
res_all_students$cpu_time_average
#> 00000001 00000002 00000003
#>     0.69     3.04     0.71
res_all_students$failure_ratio
#> 00000001 00000002 00000003
#>        0        0        0
```

Now we can rank the different portfolios/students based on a weighted combination of the percentile of the scores of the performance measures:

```
leaderboard <- portfolioLeaderboard(res_all_students)

# show leaderboard
library(gridExtra)
grid.table(formatC(leaderboard$leaderboard, digits = 3))
```

	sharpe ratio score	max drawdown score	cpu time score	failure ratio score	final score
00000001	10	6.67	10	70	96.7
00000003	6.67	10	6.67	70	93.3
00000002	3.33	3.33	3.33	70	80

3.1 Example of a script file to be submitted by a student

Consider the student Mickey Mouse with id number 666. Then the script file should be named `Mickey-Mouse-666.R` and should contain the portfolio function called exactly `portfolio_fun()` as well as any other auxiliary functions that it may require (needless to say that the required packages should be loaded in that script with `library()`):

```
library(CVXR)

auxiliary_function <- function(x) {
  # here whatever code
}

portfolio_fun <- function(prices) {
  X <- diff(log(prices))[-1]  # compute log returns
  mu <- colMeans(X)  # compute mean vector
```

```r
  Sigma <- cov(X)  # compute the SCM
  # design mean-variance portfolio
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - 0.5*quad_form(w, Sigma)),
                  constraints = list(w >= 0, sum(w) == 1))
  result <- solve(prob)
  return(as.vector(result$getValue(w)))
}
```