Portfolio Backtesting

Daniel P. Palomar and Rui ZHOU
Hong Kong University of Science and Technology (HKUST)
2018-12-05

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(dataset)
```

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.

```r
length(dataset)
#> [1] 10
str(dataset[[1]])
#> List of 1
#>  $ prices:An 'xts' object on 2014-09-02/2016-08-30 containing:
#>   Data: num [1:504, 1:50] 18.8 19 19.3 19.2 19.1 ...
#>  - attr(*, "dimnames")=List of 2
#>   ..$ : NULL
#>   ..$ : chr [1:50] "NVDA" "FL" "CDNS" "EIX" ...
#>   Indexed by objects of class: [Date] TZ: UTC
#>   xts Attributes:
#> List of 2
#>   ..$ src    : chr "yahoo"
#>   ..$ updated: POSIXct[1:1], format: "2018-12-05 13:30:48"

colnames(dataset[[1]]$prices)
#>  [1] "NVDA"  "FL"    "CDNS"  "EIX"   "HOLX"  "MCK"   "AZO"   "INCY"
#>  [9] "IPG"   "ANSS"  "EW"    "INTC"  "HRB"   "BEN"   "LKQ"   "WFC"
#> [17] "FRT"   "ICE"   "CB"    "COST"  "BLK"   "CMCSA" "NBL"   "SRCL"
#> [25] "BMY"   "CAH"   "ED"    "D"     "CTAS"  "HP"    "ROP"   "CMA"
#> [33] "TXN"   "ALGN"  "BAC"   "TRV"   "DVN"   "BIIB"  "DE"    "ABC"
#> [41] "VTR"   "OKE"   "ADBE"  "GLW"   "NWSA"  "MAC"   "ADP"   "HD"
#> [49] "HCA"   "AAPL"
```

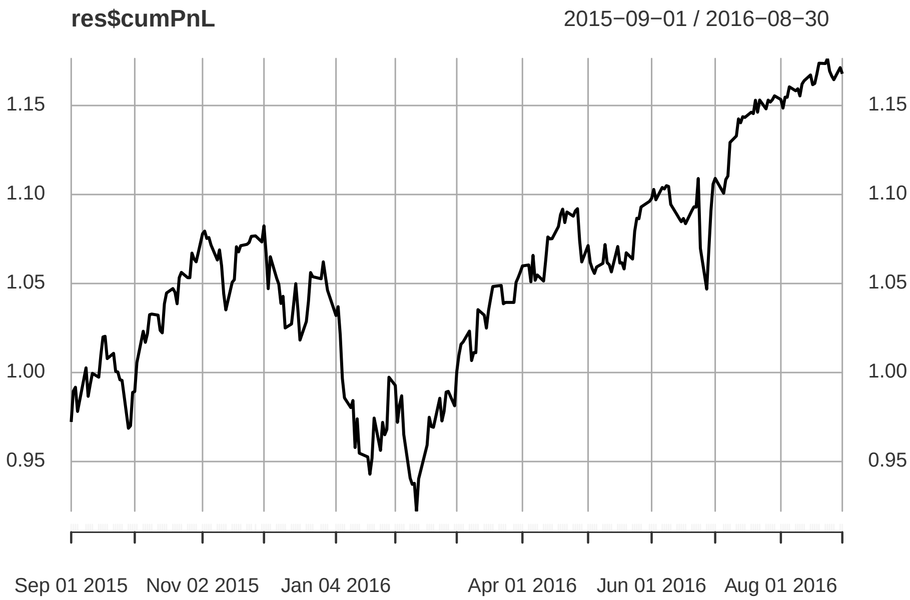
2.2 Backtesting a single portfolio

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`:

```r
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:

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




```
res$performance
#>      Sharpe ratio       max drawdown     annual return annual volatility
#>         1.0263998          0.1480544         0.1678269         0.1635102
#>    Sterling ratio        Omega ratio           ROT bps
#>         1.1335487          1.1924012      2783.8805658
```

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, dataset[[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, dataset[[1]], shortselling = TRUE)
res$error
#> [1] FALSE
res$error_message
#> NULL
res$cpu_time
```

```
#> [1] 0.11
res$performance
#>      Sharpe ratio      max drawdown     annual return annual volatility
#>       -0.18563752        0.04528734       -0.00836890        0.04508195
#>    Sterling ratio       Omega ratio           ROT bps
#>       -0.18479559        0.97358349       -9.46725948
```

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, dataset[[1]])
res$error
#> [1] FALSE
res$error_message
#> NULL
res$cpu_time
#> [1] 3.29
res$performance
#>      Sharpe ratio      max drawdown     annual return annual volatility
#>         1.6750431         0.3579369         0.7618456         0.4548215
#>    Sterling ratio       Omega ratio           ROT bps
#>         2.1284355         1.3238328      2018.4725092
```

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, dataset[1:5])
names(res)
#> [1] "returns"             "cumPnL"              "performance"
#> [4] "performance_summary" "cpu_time"            "cpu_time_average"
#> [7] "failure_ratio"       "error"               "error_message"
res$cpu_time
#> [1] 3.14 3.09 3.12 3.12 3.13
res$performance
#>                       dataset 1   dataset 2   dataset 3  dataset 4
#> Sharpe ratio          1.6750431   0.4812042  0.18838042 -0.0704581
#> max drawdown          0.3579369   0.2475224  0.21892825  0.2598196
#> annual return         0.7618456   0.1517881  0.04843494 -0.0318472
```

```
#> annual volatility    0.4548215   0.3154338  0.25711238  0.4520020
#> Sterling ratio       2.1284355   0.6132295  0.22123657 -0.1225743
#> Omega ratio          1.3238328   1.1072693  1.05812777  1.0263998
#> ROT bps           2018.4725092 146.8687867 49.28900538 99.5185526
#>                     dataset 5
#> Sharpe ratio        0.7969627
#> max drawdown        0.2855616
#> annual return       0.3465902
#> annual volatility   0.4348889
#> Sterling ratio      1.2137144
#> Omega ratio         1.1999258
#> ROT bps           288.3660319
```

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

```
res$cpu_time_average
#> [1] 2.656
res$performance_summary
#>      Sharpe ratio (median)      max drawdown (median)
#>                 0.04438826                 0.34603766
#>     annual return (median) annual volatility (median)
#>                 0.01703249                 0.38371610
#>    Sterling ratio (median)       Omega ratio (median)
#>                 0.08632498                 1.04278734
#>           ROT bps (median)
#>                97.70912146
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()`:

```
res <- multiplePortfolioBacktest(portfolio_fun_list = list(uniform_portfolio_fun,
                                                           GMVP_portfolio_fun),
                                 prices =  dataset[1:5], shortselling = TRUE)
#> 2018-12-05 14:31:20 - Execute func1
#> 2018-12-05 14:31:21 - Execute func2
res
#> $performance_summary
#>       Sharpe ratio  (median) max drawdown  (median)
#> func1               1.240415             0.11254458
#> func2               0.316964             0.03960071
#>       annual return  (median) annual volatility  (median)
#> func1              0.16782688                  0.14795379
#> func2              0.01779474                  0.04945851
#>       Sterling ratio  (median) Omega ratio  (median) ROT bps  (median)
#> func1                1.6363772              1.246941        2797.89661
#> func2                0.2985922              1.065072          41.42648
#>
#> $cpu_time_average
#> func1 func2
#> 0.020 0.034
```

```
#>
#> $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 =  dataset[1:3])
#> 2018-12-05 14:31:22 - Execute code from Firstname1 Surname1 (00000001)
#> 2018-12-05 14:31:25 - Execute code from Firstname2 Surname2 (00000002)
#> 2018-12-05 14:31:35 - Execute code from Firstname3 Surname3 (00000003)
res_all_students$performance_summary
#>          Sharpe ratio  (median) max drawdown  (median)
#> 00000001              0.5925987              0.1628546
#> 00000002              0.1673211              0.1720040
#> 00000003              0.9714862              0.1059337
#>          annual return  (median) annual volatility  (median)
#> 00000001              0.11743671                   0.1981724
#> 00000002              0.03521717                   0.2019172
#> 00000003              0.14855545                   0.1458700
#>          Sterling ratio  (median) Omega ratio  (median) ROT bps  (median)
#> 00000001                0.7211139              1.118539          238.5474
#> 00000002                0.2047463              1.046296          161.3769
#> 00000003                1.4023441              1.177569          206.5219
res_all_students$cpu_time_average
#>  00000001  00000002  00000003
#> 0.8133333 3.1933333 0.7800000
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 rank percentiles (termed scores) of the performance measures:

```
leaderboard <- portfolioLeaderboard(res_all_students, weights = list(Sharpe_ratio = 7, max_drawdown = 1

# show leaderboard
library(gridExtra)
grid.table(leaderboard$leaderboard_scores)
```

	Sharpe ratio score	max drawdown score	annual return score	ROT score	final score
00000003	100	100	100	50	95
00000001	50	50	50	100	55
00000002	0	0	0	0	0

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
  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)))
}
```

4 Appendix

4.1 Performance criteria

The definition of performance criteria used in this package is listed as below

- **expetced return:** the annualized return
- **volatility:** the annualized standard deviation of returns
- **max drawdown:** the maximum loss from a peak to a trough of a portfolio, see also here
- **Sharpe ratio:** annualized Sharpe ratio, the ratio between `annualized return` and `annualized standard deviation`
- **Sterling ratio:** the return over average drawdown, see here for complete definition. In the package, we use

$$\text{Sterling ratio} = \frac{\text{annualized return}}{\text{max drawdown}}$$

- **Omega ratio:** the probability weighted ratio of gains over losses for some threshold return target, see here for complete definition. The ratio is calculated as:

$$\Omega(r) = \frac{\int_r^\infty (1 - F(x))dx}{\int_{-\infty}^r F(x)dx}$$

  In the package, we use $\Omega(0)$, which is also known as Gain-Loss-Ratio.

- **Return over Turnover (ROT):** the sum of cummulative return over the sum of turnover.