Portfolio Backtesting

Daniel P. Palomar and Rui ZHOU
Hong Kong University of Science and Technology (HKUST)
2018-11-12

Contents

1 Installation **1**

2 Usage of the package **1**
2.1 Loading data 1
2.2 Backtesting a single portfolio 2
2.3 Backtesting multiple portfolios 5

3 Usage for grading students in a course **6**
3.1 Example of a script file to be submitted by a student 7

4 Appendix **7**
4.1 Performance criteria 7

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)  # you may want to specify data(prices, package = "portfolioBacktest")
              # in case there is any conflict with the package PerformanceAnalytics
```

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] 100
str(prices[[1]])
#> An 'xts' object on 2013-01-23/2014-12-29 containing:
#>   Data: num [1:474, 1:46] 128 130 130 129 128 ...
#>  - attr(*, "dimnames")=List of 2
#>   ..$ : NULL
#>   ..$ : chr [1:46] "1 HK Equity" "101 HK Equity" "1038 HK Equity" "1044 HK Equity" ...
#>   Indexed by objects of class: [Date] TZ: UTC
#>   xts Attributes:
#>  NULL

colnames(prices[[1]])
#>  [1] "1 HK Equity"    "101 HK Equity"  "1038 HK Equity" "1044 HK Equity"
#>  [5] "1088 HK Equity" "1093 HK Equity" "11 HK Equity"   "1109 HK Equity"
#>  [9] "1177 HK Equity" "12 HK Equity"   "1299 HK Equity" "1398 HK Equity"
#> [13] "151 HK Equity"  "16 HK Equity"   "17 HK Equity"   "175 HK Equity"
#> [17] "19 HK Equity"   "1928 HK Equity" "2 HK Equity"    "2007 HK Equity"
#> [21] "2018 HK Equity" "2313 HK Equity" "2318 HK Equity" "2319 HK Equity"
#> [25] "2382 HK Equity" "2388 HK Equity" "2628 HK Equity" "267 HK Equity"
#> [29] "27 HK Equity"   "3 HK Equity"    "3328 HK Equity" "386 HK Equity"
#> [33] "388 HK Equity"  "3988 HK Equity" "5 HK Equity"    "6 HK Equity"
#> [37] "66 HK Equity"   "688 HK Equity"  "762 HK Equity"  "823 HK Equity"
#> [41] "83 HK Equity"   "836 HK Equity"  "857 HK Equity"  "883 HK Equity"
#> [45] "939 HK Equity"  "941 HK Equity"
```

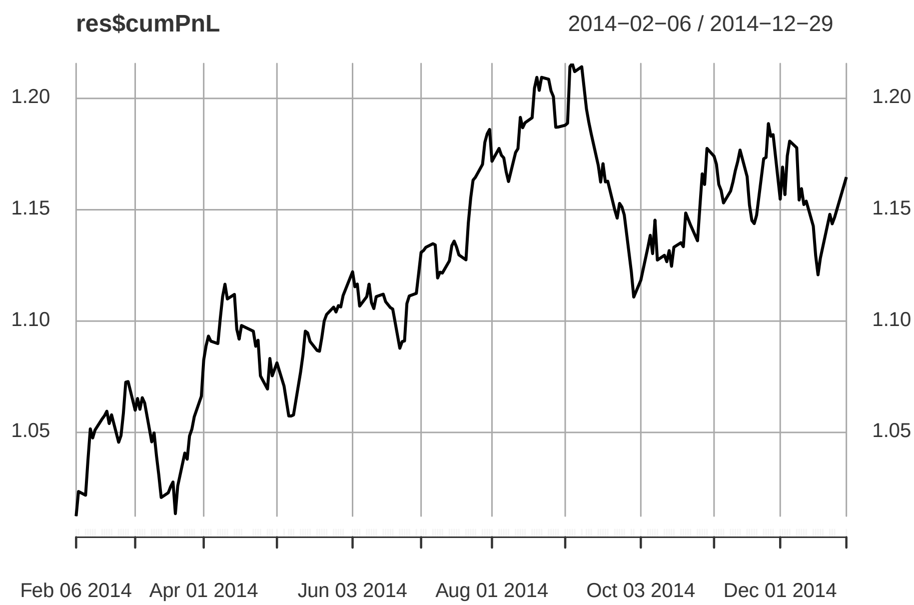
2.2 Backtesting a single portfolio

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 
#>    1.466254e+00    8.627088e-02    1.889171e-01    1.288433e-01 
#>  Sterling ratio     Omega ratio         ROT bps 
#>    2.189813e+00    1.257056e+00    2.589679e+03
```

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] 0.05
```

```r
res$performance
#>    Sharpe ratio    max drawdown expected return      volatility 
#>      1.64718707      0.02471534      0.05919997      0.03594004 
#>  Sterling ratio     Omega ratio         ROT bps 
#>      2.39527214      1.30331447    173.12764544
```

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

```r
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:

```r
res <- portfolioBacktest(Markowitz_portfolio_fun, prices[[1]])
res$error
#> [1] FALSE
res$error_message
#> NULL
res$cpu_time
#> [1] 2.95
res$performance
#>    Sharpe ratio    max drawdown expected return      volatility 
#>      -0.4187583       0.3460377      -0.1295226       0.3093016 
#>  Sterling ratio     Omega ratio         ROT bps 
#>      -0.3743020       0.9534549     -88.4547661
```

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:

```r
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$cpu_time
#> [1] 3.05 2.95 2.46 2.70 2.54
res$performance
#>                  dataset 1   dataset 2   dataset 3   dataset 4
#> Sharpe ratio    -0.4187583  -0.5146592   1.2704831   1.8411582
#> max drawdown     0.3460377   0.4686311   0.4126044   0.1817962
#> expected return -0.1295226  -0.2614145   0.6155478   0.5834570
#> volatility       0.3093016   0.5079371   0.4844990   0.3168967
```

```
#> Sterling ratio   -0.3743020   -0.5578257   1.4918593   3.2094019
#> Omega ratio       0.9534549    0.9417655   1.2301029   1.3361718
#> ROT bps         -88.4547661 -174.8816536 482.4063915 296.5086680
#>                   dataset 5
#> Sharpe ratio     0.04438826
#> max drawdown     0.19730661
#> expected return  0.01703249
#> volatility       0.38371610
#> Sterling ratio   0.08632498
#> Omega ratio      1.04278734
#> ROT bps         97.70912146
```

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

```
res$cpu_time_average
#> [1] 2.74
res$performance_summary
#>    Sharpe ratio (median)    max drawdown (median) expected return (median)
#>               0.04438826               0.34603766               0.01703249
#>      volatility (median)  Sterling ratio (median)     Omega ratio (median)
#>               0.38371610               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 =  prices[1:5], shortselling = TRUE)
#> 2018-11-01 15:59:37 - Execute func1
#> 2018-11-01 15:59:38 - Execute func2
res
#> $performance_summary
#>       Sharpe ratio  (median) max drawdown  (median)
#> func1              1.178459             0.15771401
#> func2              1.583984             0.04479846
#>       expected return  (median) volatility  (median)
#> func1               0.18891706           0.20705821
#> func2               0.06276355           0.05739085
#>       Sterling ratio  (median) Omega ratio  (median) ROT bps  (median)
#> func1               1.776008             1.211860         2589.6794
#> func2               2.395272             1.303314          173.1276
#>
#> $cpu_time_average
#> func1 func2
#> 0.018 0.026
#>
#> $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])
#> 2018-11-01 15:59:39 - Execute code from Firstname1 Surname1 (00000001)
#> 2018-11-01 15:59:41 - Execute code from Firstname2 Surname2 (00000002)
#> 2018-11-01 15:59:49 - Execute code from Firstname3 Surname3 (00000003)
res_all_students$performance_summary
#>          Sharpe ratio  (median) max drawdown  (median)
#> 00000001             2.3098857             0.1119180
#> 00000002             0.5823419             0.2506662
#> 00000003             0.6569324             0.1431620
#>          expected return  (median) volatility  (median)
#> 00000001                0.3841061           0.2054775
#> 00000002                0.1417214           0.3545257
#> 00000003                0.1348055           0.2229336
#>          Sterling ratio  (median) Omega ratio  (median) ROT bps  (median)
#> 00000001               5.5126225              1.394246          921.8511
#> 00000002               0.8236259              1.124574         1396.9909
#> 00000003               1.0377623              1.122605          267.2722
res_all_students$cpu_time_average
#>  00000001  00000002  00000003 
#> 0.6333333 2.5733333 0.6133333 
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	expected return score	ROT score	final score
00000001	100	100	100	50	95
00000003	50	50	0	0	40
00000002	0	0	50	100	15

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.