Package: portsort (via r-universe)

March 25, 2025

Type Package

Date 2018-09-12

Title Factor-Based Portfolio Sorts

Version 0.1.0

Author Alex Dickerson [aut,cre], Jonathan Spohnholtz [aut,cre]

Maintainer Alex Dickerson <a.dickerson@warwick.ac.uk>

Description Designed to aid both academic researchers and asset managers in conducting factor based portfolio sorts. Provides functionality to sort assets into portfolios for up to three factors via a conditional or unconditional sorting procedure.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Depends xts, zoo, R (>= 2.10)

Suggests PortfolioAnalytics, PerformanceAnalytics, knitr

VignetteBuilder knitr

Imports stats

NeedsCompilation no

Date/Publication 2018-09-30 15:50:03 UTC

Additional_repositories https://cranhaven.r-universe.dev

Repository https://cranhaven.r-universe.dev

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/portsort

RemoteSha 9374eec3413be2b83be236fe889e5ef0589bc77d

RemoteSubdir portsort

Contents

`conditional.sort` *Conditional Portfolio Sort*

Description

Calculates out-of-sample mean sub-portfolio returns and the composition of each sub-portfolio using the conditional portfolio sorting method.

Usage

```
conditional.sort(Fa,Fb=NULL,Fc=NULL,R.Forward,dimA,dimB=NULL,dimC=NULL,type = 7)
```

Arguments

`Fa`	xts-object containing data for the first dimension of sort
`Fb`	xts-object containing data for the second dimension of sort (optional)
`Fc`	xts-object containing data for the third dimension of sort (optional)
`R.Forward`	xts-object containing forward returns
`dimA`	vector of break points between 0 and 1
`dimB`	vector of break points between 0 and 1 (optional)
`dimC`	vector of break points between 0 and 1 (optional)
`type`	pass-through parameter to the `quantile` function

Details

The conditional sort function sorts assets based on each factor (Fa to Fc) from low to high in a dependent fashion at each time *t*. Based on the sorted assets in each sub-portfolio at time *t*, mean out-of-sample sub-portfolio returns are computed for time *t+1*. After each dimension of sort, the subsequent sort is done only within each prior sorted sub-portfolio. Hence, the first factor that is sorted on yields greater influence on the overall sorting procedure. The function outputs out-of-sample returns for each sub-portfolio in columns and a list of the sub-portfolio constituents at each rebalancing point.

Value

`returns`	Out-of-sample sub-portfolio returns
`portfolio`	List of the sub-portfolio constituents over time

Note

The function implicitly handles NA/NaN or Inf values at each rebalancing point (at time *t*) by excluding them from the `quantile` function. Furthermore, if there are any NA, NaN or Inf values in the R.Forward object when computing out-of-sample returns, these are also excluded. The function outputs returns in columns. For example, if a double sort is conducted with both Fa and Fb including 3 breakpoints (a 3v3) sort, column 1 will contain out-of-sample returns for the 'Low-Low' sub-portfolio, column 4 will contain out-of-sample returns for the 'Mid-Low' sub-portfolio whilst column 9 will contain the 'High-High' sub-portfolio returns.

Author(s)

Jonathan Spohnholtz and Alexander Dickerson

Examples

```
# Load the included data
library(portsort)
data(Factors)

# Specifiy the sort dimension - in this case, a double sort on lagged returns and Bitcoin volumes
# with 4 breakpoints (a 4v4 sort)
dimA = c(0,0.25,0.5,0.75,1)
dimB = c(0,0.25,0.5,0.75,1)

# Specify the factors for the double sort
# Lagged returns, lagged volumes are stored in the Factors list

R.Forward = Factors[[1]]; R.Lag = Factors[[2]]; V.Lag = Factors[[3]]

# Subset the data from late 2017
R.Forward = R.Forward["2017-12-01/"]
R.Lag = R.Lag["2017-11-30/2018-09-05"]
V.Lag = V.Lag["2017-11-30/2018-09-05"]

Fa = R.Lag
Fb = V.Lag

# Conduct a conditional sort
sort.output <- conditional.sort(Fa,Fb,Fc=NULL,R.Forward = R.Forward,dimA = dimA,dimB = dimB)
```

Factors *Cryptocurrency Returns and Volume Data*

Description

The data set includes lagged log returns, lagged volume denominated in Bitcoin and forward log returns aggregated every 24-hours for a cross-section of 26 cryptocurrency pairs from the 1st January 2017 to 9th September 2018. The data was downloaded from CryptoCompare - a free API accessible at https://min-api.cryptocompare.com

Usage

```
data("Factors")
```

Format

A list of three xts objects including lagged returns (R.Lag), lagged volumes (V.Lag) and forward returns (R.Forward).

Source

https://min-api.cryptocompare.com

Examples

```
# Load data
data(Factors)
# Unlist the data
R.Forward = Factors[[1]]; R.Lag = Factors[[2]]; V.Lag = Factors[[3]]
head(V.Lag[1:5,1:5])
```

portfolio.frequency *Calculate Sub-Portfolio Concentration*

Description

Computes the frequency that an asset appears in each sub-portfolio based on its rank.

Usage

```
portfolio.frequency(sort.output, rank)
```

Arguments

sort.output	object returned from either the conditional.sort or unconditional.sort function.
rank	input the rank of the security you would like to return the frequency for.

Details

Returns the frequency that the security appears in each sub-portfolio based on the rank input.

Author(s)

Alexander Dickerson and Jonathan Spohnholtz

Examples

```
# Load the included data
library(portsort)
data(Factors)

# Specifiy the sort dimension - in this case, a double-sort on lagged returns and Bitcoin volumes
dimA = 0:3/3
dimB = 0:3/3

# Specify the factors
# Lagged returns, lagged volumes are stored in the Factors list
R.Forward = Factors[[1]]; R.Lag = Factors[[2]]; V.Lag = Factors[[3]]

# Subset the data from late 2017
R.Forward = R.Forward["2017-12-01/"]
R.Lag = R.Lag["2017-11-30/2018-09-05"]
V.Lag = V.Lag["2017-11-30/2018-09-05"]

Fa = R.Lag
Fb = V.Lag



# Conduct an unconditional sort (in this case) or a conditional sort
sort.output = unconditional.sort(Fa = Fa, Fb = Fb , R.Forward = R.Forward, dimA = dimA, dimB = dimB)

# We want to see which security appeared the most in each sub-portfolio,
# i.e the secruity with a rank of 1.

rank = 1
portfolio.frequency(sort.output,rank)
```

`portfolio.mean.size` *Calculate Mean Sub-Portfolio Size*

Description

Primarily used in the case of an unconditional sort - this function computes the average number of securities in each sub-portfolio across time.

Usage

```
portfolio.mean.size(sort.output)
```

Arguments

sort.output	object returned from either the conditional.sort or unconditional.sort function.

Author(s)

Alexander Dickerson and Jonathan Spohnholtz

Examples

```
# Load the included data
library(portsort)
data(Factors)

# Specifiy the sort dimension - in this case, a double-sort on lagged returns and Bitcoin volumes
dimA = 0:3/3
dimB = 0:3/3

# Specify the factors
# Lagged returns, lagged volumes are stored in the Factors list
R.Forward = Factors[[1]]; R.Lag = Factors[[2]]; V.Lag = Factors[[3]]

# Subset the data from late 2017
R.Forward = R.Forward["2017-12-01/"]
R.Lag = R.Lag["2017-11-30/2018-09-05"]
V.Lag = V.Lag["2017-11-30/2018-09-05"]

Fa = R.Lag
Fb = V.Lag

# Conduct an unconditional sort (in this case) or a conditional sort
sort.output = unconditional.sort(Fa = Fa, Fb = Fb , R.Forward = R.Forward, dimA = dimA, dimB = dimB)

# We want to compute the average size of each sub-portfolio

portfolio.mean.size(sort.output)
```

---

portfolio.turnover *Calculate Sub-Portfolio Turnover*

Description

Calculates sub-portfolio turnover between each rebalancing period.

Usage

```
portfolio.turnover(sort.output)
```

Arguments

sort.output	object returned from either the conditional.sort or unconditional.sort function.

Details

This function calculates the turnover within each sub-portfolio over time and returns a list containing the turnover values and the mean turnover across time.

Value

Turnover	xts object of turnovers for each rebalancing point.
Mean Turnover	mean turnover for each sub-portfolio averaged over time.

Author(s)

Jonathan Spohnholtz and Alexander Dickerson

Examples

```
# Load the included data
library(portsort)
data(Factors)

# Specifiy the sort dimension - in this case, a double-sort on lagged returns and Bitcoin volumes
dimA = 0:3/3
dimB = 0:3/3

# Specify the factors
# Lagged returns, lagged volumes are stored in the Factors list
R.Forward = Factors[[1]]; R.Lag = Factors[[2]]; V.Lag = Factors[[3]]

# Subset the data from late 2017
R.Forward = R.Forward["2017-12-01/"]
R.Lag = R.Lag["2017-11-30/2018-09-05"]
V.Lag = V.Lag["2017-11-30/2018-09-05"]

Fa = R.Lag
Fb = V.Lag

# Conduct an unconditional sort (in this case) or a conditional sort
sort.output = unconditional.sort(Fa = Fa, Fb = Fb , R.Forward = R.Forward, dimA = dimA, dimB = dimB)

# Compute Turnover by passing the sort.output object to the turnover function
sort.turnover = portfolio.turnover(sort.output)
```

`unconditional.sort` *Unconditional Portfolio Sort*

Description

Calculates out-of-sample mean sub-portfolio returns and the composition of each sub-portfolio using the unconditional portfolio sorting method.

Usage

```
unconditional.sort(Fa,Fb=NULL,Fc=NULL,R.Forward,dimA,dimB=NULL,dimC=NULL,type = 7)
```

Arguments

`Fa`	xts-object containing data for the first dimension of sort
`Fb`	xts-object containing data for the second dimension of sort (optional)
`Fc`	xts-object containing data for the third dimension of sort (optional)
`R.Forward`	xts-object containing forward returns
`dimA`	vector of break points between 0 and 1
`dimB`	vector of break points between 0 and 1 (optional)
`dimC`	vector of break points between 0 and 1 (optional)
`type`	pass-through parameter to the `quantile` function

Details

The unconditional sort function sorts assets based on each factor (Fa to Fc) from low to high independently at each time *t* and forms sub-portfolios based on the intersection between them. Based on the sorted assets in each sub-portfolio at time *t*, mean out-of-sample sub-portfolio returns are computed for time *t+1*. The function outputs out-of-sample returns for each sub-portfolio in columns and a list of the sub-portfolio constituents at each rebalancing point.

Value

`returns`	Out-of-sample sub-portfolio returns
`portfolio`	List of the sub-portfolio constituents over time

Note

The function implicitly handles NA/NaN or Inf values at each rebalancing point (at time *t*) by excluding them from the `quantile` function. Furthermore, if there are any NA, NaN or Inf values in the R.Forward object when computing out-of-sample returns, these are also excluded. The function outputs returns in columns. For example, if a double sort is conducted with both Fa and Fb including 3 breakpoints (a 3v3) sort, column 1 will contain out-of-sample returns for the 'Low-Low' sub-portfolio, column 4 will contain out-of-sample returns for the 'Mid-Low' sub-portfolio whilst column 9 will contain the 'High-High' sub-portfolio returns.

Author(s)

Jonathan Spohnholtz and Alexander Dickerson

Examples

```
# Load the included data
library(portsort)
data(Factors)

# Specifiy the sort dimension - in this case, a double sort on lagged returns and Bitcoin volumes
# with 4 breakpoints (a 4v4 sort)
dimA = c(0,0.25,0.5,0.75,1)
dimB = c(0,0.25,0.5,0.75,1)

# Specify the factors for the double sort
# Lagged returns, lagged volumes are stored in the Factors list

R.Forward = Factors[[1]]; R.Lag = Factors[[2]]; V.Lag = Factors[[3]]

# Subset the data from late 2017
R.Forward = R.Forward["2017-12-01/"]
R.Lag = R.Lag["2017-11-30/2018-09-05"]
V.Lag = V.Lag["2017-11-30/2018-09-05"]

Fa = R.Lag
Fb = V.Lag

# Conduct an unconditional sort
sort.output <- conditional.sort(Fa,Fb,Fc=NULL,R.Forward = R.Forward,dimA = dimA,dimB = dimB)
```

Index