Hints for Implementing the ideas in the above paper using R software.

by H D Vinod, Fordham University, July 12, 2024

This is an Online Appendix for a paper entitled "May 2024 Buy-Sell Guide for Dow Jones 30 Stocks and Modified Omega Criterion," by H. D. Vinod of Fordham University, NY, forthcoming in *Journal of Risk and Financial Management* (JRFM).

This MS Word file contains R code and related details. We assume the reader is familiar with the syntax of R software. We list several R Functions needed for implementing the ideas in my paper.

A common input to many functions is a matrix having n rows and p columns. Investor returns data for *n* periods is a column of *mtx*. There are *p* columns for *p* stocks.

The paper computes rank of 30 stocks by six algorithms implemented by an "in-sample" version and randomized unbiased "out-of-sample" version. Two versions of six algorithms lead to 12 sets of ranks. The ranks are further averaged to find a summary recommendation to buy or sell combining the insights of 12 algorithms.

The material in typewriter font size 10 in this document can be copied and pasted on R console. Many print commands are commented out using # before them. Users are invited to remove the "#" to see the output written to the console.

#

#start with R memory cleanup

#

rm(list=ls())

seed <- 42

set.seed(seed)

print(c("seed=",seed),quote=FALSE)

options(prompt = " ", continue = " ", width = 64,

useFancyQuotes = FALSE)

print(date())

#

# create data mtx with 3 stocks for testing where x3 has a missing data

#

x1=c(2,5,6,9,13,18,21,5,11,14,4,7,12,13,6,3,8,1,15,2,10,9)

x2=c(3,6,9,12,14,19,27,9,11,2,3,8,1,6,15,10,13,14,5,7,4,12)

x3=c(2,6,NA,11,13,25,25,11,9,10,12,6,4,3,2,1,7,8,5,15,14,13)

mtx=cbind(x1,x2,x3)

mtx=mtx[complete.cases(mtx),]

mtxnam=colnames(mtx) #vector of stock names for future use

#

# include some preliminary R code for R functions

#

#

# function sharpe(mtx) for sharpe ratios of all columns

#

sharpe=function(mtx){

# rank stocks by sharpe ratios

fn=function(x){xx=x[!is.na(x)]

return(mean(xx)/sd(xx)) }

sha=apply(mtx,2,fn)#apply fn to each column of mtx

#print(sha)

rsha=rank(-sha) #negative sign, largest=best is ranked 1

#print(rsha)

#colnames(rsha)=colnames(mtx)

return(rsha)

}

#

#function omega(mtx) to compute ranks by the gain to pain ratio

#

omega=function(mtx,zer=0){

# rank stocks by omega=gain to pain ratios

fn=function(x){xx=x[!is.na(x)]

if(zer!=0) zer=median(xx)

gain=sum(xx[xx>=zer])

#print(gain)

pain=abs(sum(xx[xx<zer]))

#print(pain)

ifelse(pain!=0,

return(gain/pain),

return(gain))

}

ome=apply(mtx,2,fn)

#print(sha)

rome=rank(-ome) #negative sign, largest=best is ranked 1

#print(rsha)

#colnames(rsha)=colnames(mtx)

return(rome)

}

#

# function outOFs2(mtx) for out of sample evaluation repeated "reps" times

# keeping randomly chosen "pctOut" data as out-of-sample for any

#specified "portfo" as stock-picking algorithm

#

outOFs2=function(mtx, pctOut=5, reps=10, seed=23, portfo,

 verbo=FALSE, ...) {

n=NROW(mtx)

p=NCOL(mtx)

if (pctOut>50) {print("percent left-out exceeds 50, reset as 50")

 pctOut=50}

if(verbo){

 print(c("outOFs2 mtx: n,p",n,p)) }#end verbo

outrank=matrix(NA,nrow=2, ncol=p)

outrank[1,]=portfo(mtx, ...)

# select pctOut% of n observations for out-of-sample

n5=max(1,round(n\*pctOut/100,0)) #rounded integer to 0 digits for out-of-sample

if(n5==1) n5=2 #leave-out only one fails (a limitation of R syntax).

print(c("out-of-sample",n5,"observations"))

 #reps number of times we repeat leaving out n5 observations

avgRnk=matrix(NA,nrow=reps,ncol=NCOL(mtx))#average ranks place holder

for (irep in (1:reps)){

 if(verbo) print(c("replication begins",irep))

 #randomize for out-of-sample

 seed1=seed+1

 set.seed(seed1)

 sampN=sample(1:n,replace=FALSE)

 rowselec=sampN[1:n5]

 if(length(rowselec)<1) stop("too few rows for oos testing")

#use function called portfo to get average ranks

#based on randomly selected n5 rows

 rnk1=portfo(mtx[rowselec,], ...)

 avgRnk[irep,]=rnk1

}#loop irep

outrank[2,]=apply(avgRnk,2,mean,na.rm=TRUE) #average reps

colnames(outrank)=colnames(mtx)

print(c("out-of-sample pct",pctOut,"replicates=",reps),quote=FALSE)

rownames(outrank)=c("in-sample","av.oos")

print(outrank)

return(outrank)

}

**Specify "portfo" argument of outOFs2(.)as stock-picking algorithm from six in the paper**

1. **Sharpe-in-sample/out-of-sample**: see Section 1.1.1
2. **Omega-in-sample/out-of-sample**: section 1.1.2, and equation (7) for Ω*sum*.
3. **Decile-in-sample/out-of-sample**: uses R package ‘generalCorr,’ function called decileVote(.).
4. **Descriptive-stats-in-sample/out-of-sample**: We compare the traditional descriptive stats of each stock’s data. Most stats are in the “the larger, the better” category and get (+1) as weight. The standard deviation represents risk and gets (–1) weight. This algorithm uses a weighted summary of these stats for stock-picking.
5. **Moments-in-sample/out-of-sample**: uses R package ‘generalCorr’ called momentVote(.).
6. **Exact-Stochastic-Dominance-in-sample/out-of-sample**: uses R package ‘generalCorr’ exactSd().

#

library(generalCorr)

options(np.messages=FALSE)#

**# Sharpe-in-sample/out-of-sample**:

print("portfo=sharpe")

a1=outOFs2(mtx,portfo=sharpe,pctOut=40,verbo=FALSE,reps=50)

**# Omega-in-sample/out-of-sample**:

print("portfo=omega")

a2=outOFs2(mtx,portfo=omega,pctOut=40,verbo=FALSE,zer=1,reps=50)

**# Decile-in-sample/out-of-sample**

decileRank=function(mtx){

# rank stocks by all deciles using generalCorr::decileVote

dv=decileVote(mtx)

#return(dv$out[20,])

#}

rdv=dv$out[20,]

#print(rdv)

#colnames(rsha)=colnames(mtx)

return(rdv)

}

print("portfo=decileRank")

a3=outOFs2(mtx,portfo=decileRank,pctOut=40,verbo=FALSE,reps=50)

**# Descriptive-stats-in-sample/out-of-sample**:

#combining descriptive stats into one rank

descrip=function(mtx){

 p=NCOL(mtx)

 summary2=function(x) {xx=x[!is.na(x)]

 c(summary(xx),sd(xx))}

a1=apply(mtx,2,summary2)

rownames(a1)=c("min","Q1","Median","Mean","Q3","Max","sd")

a2=matrix(NA,nrow=7, ncol=p) #always 7 descriptive stats

for (j in 1:7){

 a2[j,]=rank(-a1[j,]) #neg sign rank 1 for most desirable largest value

 if(j==7) a2[j,]=rank(a1[j,]) #smallest sd is most desirable=> positive

}

a3=apply(a2,2,mean,na.rm=TRUE)

return(a3)

}# end descrip function

print("portfo=descrip")

a4=outOFs2(mtx,portfo=descrip,pctOut=40,verbo=FALSE,reps=50)

# **Moments-in-sample/out-of-sample**:

momentRank=function(mtx){

p=ncol(mtx)

# rank stocks by all moments using generalCorr::momentVote

#fn=function(x){xx=x[!is.na(x)]

mv=momentVote(mtx)

#return(dv$out[20,])

#}

rmv=mv[12,1:p] #row12has choice

#print(rmv)

#colnames(rsha)=colnames(mtx)

return(rmv)

}

print("portfo=momentRank")

a5=outOFs2(mtx,portfo=momentRank,pctOut=40,verbo=FALSE,reps=50)

**# Exact-Stochastic-Dominance-in-sample/out-of-sample**:

exactSDRank=function(mtx){

p=ncol(mtx)

# rank stocks by all moments using generalCorr::momentVote

#fn=function(x){xx=x[!is.na(x)]

ex1=exactSdMtx(mtx)

ex2=summaryRank(ex1$out) #row 10 has choice

rexsd=ex2[10,1:p] #row12has choice

#print(rexsd)

#colnames(rsha)=colnames(mtx)

return(rexsd)

}

print("portfo=exactSD")

a6=outOFs2(mtx,portfo=exactSDRank,pctOut=40,verbo=FALSE,reps=50)

**# Combining six algorithm results in a single table**

asix=rbind(a1,a2,a3,a4,a5,a6)

av6=apply(asix,2,mean,na.rm=TRUE)

avgrank=rank(av6)

aseven=rbind(asix,avgrank)

rownames(aseven)=c("Sharpe-in","Sharpe-out",

 "Omega-in","Omega-out", "Decile-in","Decile-out",

 "Descr-in","Descr-out","Momen-in","Momen-out",

 "Exact-in", "Exact-out","AvgRank")

print(aseven)

**# Dow Jones DJIA 30 stock special case** code

#Following code is only for p=30 stocks case from DJIA

library(xtable)

print(xtable(aseven[,1:10],digits=1))

print(xtable(aseven[,11:20],digits=1))

print(xtable(aseven[,21:30],digits=1))

char1=c("a", "A", "z", "x", "b",

 "c", "C", "S", "e", "d",

 "D", "g", "h", "H", "i",

 "I", "j", "J", "k", "m",

 "M", "K", "f", "n", "p",

 "t", "u", "v", "V", "w")

length(char1) #should be 30

sortNames=function(nam,val){

 mtx=cbind(nam,val)

 g1= generalCorr::sort\_matrix(mtx,2)

 return(g1)

}

ncom=nrow(aseven) #number of comparisons

a7b=aseven #place holder

for (i in 1:ncom) {#there are ncom comparisons

val=as.numeric(aseven[i,])

nam=char1

so1=sortNames(nam,val)

a7b[i,]=so1[,"nam"]

} #end for loop

colnames(a7b)=1:30

rownames(a7b)=c("Shi","Sho","Omi","Omo","Dci","Dco","Dsi",

 "Dso","Moi","Moo","Exi","Exo","avg")

print(a7b[,c(1:8,23:30)])

print(xtable(a7b[,c(1:8,23:30)])) #top 8 and worst 8 stocks

print((a7b[,c(1:2,29:30)]))

nambind=cbind(colnames(aseven),char1) #second col.has short names

sort\_matrix(nambind,2)

so2=sort\_matrix(nambind,2)

print(t(so2),quote=FALSE)

a8b=aseven #place holder

for (i in 1:ncom) {#there are ncom comparisons

 val=as.numeric(aseven[i,])

 nam=mtxnam

 so1=sortNames(nam,val)

 a8b[i,]=so1[,"nam"]

} #end for loop

print(a8b[,c(1,2,29,30)])

print(xtable(a8b[,c(1,2,29,30)]))

**# Hints for coding Sharpe Ratio with downside standard deviation**

dsd=function(x){xx=x[!is.na(x)]

# compute squared deviations from the mean

mxx=mean(xx)

dev=(xx-mxx)^2 #vector of squared deviations from mean

w=rep(1, length(xx))

#we want zero weight when x>xbar

w1=which(xx>mean(xx))

w[w1]=0

#print(w)

dsv=sum(dev\*w) #downside variance

if(dsv<=0) stop("error downside variance is zero or negative")

return(sqrt(dsv)) } #end dsd function

# Example

# x=c(-3,4,5,-8,2); dsd(x)

fn2=function(x){xx=x[!is.na(x)]

return(mean(xx)/dsd(xx))}

sharpe2=function(mtx){

sha=apply(mtx,2,fn2)

rsha=rank(-sha) #negative sign, largest=best is ranked 1

#print(rsha)

#colnames(rsha)=colnames(mtx)

return(rsha)#end of function sharpe2

}

print("portfo=sharpe2 uses downside sd for denominator")

a1=outOFs2(mtx,portfo=sharpe2,pctOut=40,verbo=FALSE,reps=50)

**# Hints: Sharpe Ratio with dsd divided by SE=sd(bootstrap replicates of SRdsd)**

library(meboot)

# code for downside std.dev (dsd) should be in memory

fn2=function(x){xx=x[!is.na(x)]

return(mean(xx)/dsd(xx))}

fn3=function(x,rep=20){xx=x[!is.na(x)]

num=fn2(xx) #numerator for sharpe ratio adjusted for estimation error

#denominator is from standard error (SE)

#setting rep allows option to change bootstrap replicates.

out=meboot(xx, reps=rep)$ens

out2=apply(out,1,fn2)#these are reps sharpe ratios

# their sd is standard error

se=sd(out2)

return(num/se)} #end fn3 function

sharpe3=function(mtx,rep=15){ #function for portfo

sha=apply(mtx,2,fn3,rep=rep)

rsha=rank(-sha) #negative sign, largest=best is ranked 1

#print(rsha)

#colnames(rsha)=colnames(mtx)

return(rsha)#end of function sharpe3

}

print("portfo=sharpe3 uses downside sd and further bootstrap")

a1=outOFs2(mtx,portfo=sharpe3,pctOut=40,verbo=FALSE,reps=50)