**Задача 2**

Составить оптимальный портфель 15 активов разных классов. Данные за последние 2 года. Результат должен быть похож как в книге на стр.287

Sys.setlocale("LC\_TIME","C")

Необходимые пакеты R:

> library(fPortfolio)

> library(robustbase) – необходимо установить R версии 3.4 и выше

> library(corpcor)

Устанавливаем их:

> install.packages("truncnorm")

> install.packages("fPortfolio")

> install.packages("robustbase")

> install.packages("corpcor")

Меры риска в целевой функции портфеля вычисляются с использованием PerformanceAnalytics (Carl and Peterson 2010). Начальная совокупность в DEoptim генерируется с использованием функции random\_portfolios в PortfolioAnalytics

(Boudt et al., 2011) .1

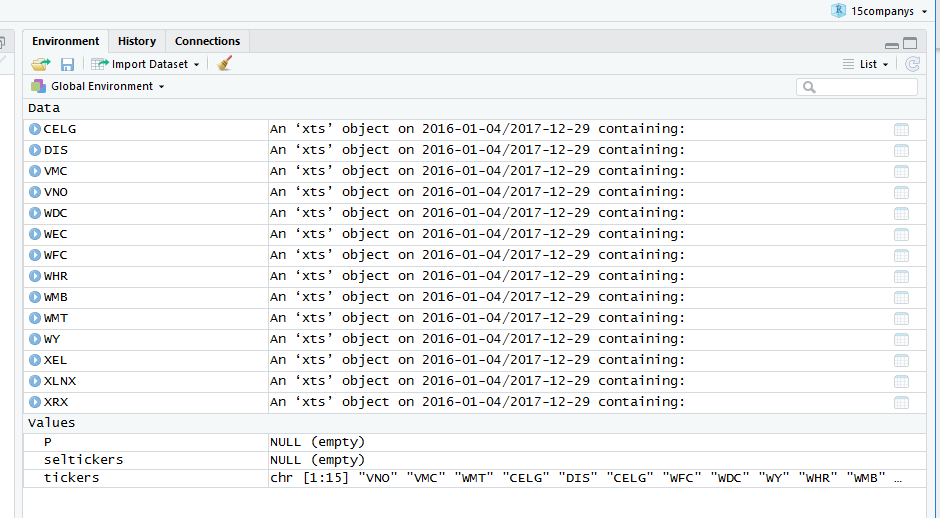
Устанавливаем 15 компаний по которым будем брать данные с биржи e S&P 500:

tickers = c( "VNO" , "VMC" , "WMT" , "CELG" , "DIS" , "CELG" , "WFC" , "WDC" ,"WY" , "WHR" , "WMB" , "WEC" , "XEL" , "XRX" , "XLNX" )

Получаем данные по 15 компаниям за период с 1 января 2016 г по 31 декабря 2017 г.:

getSymbols(tickers, from = "2016-01-01", to = "2017-12-31")

P <- NULL; seltickers <- NULL



Парсим значения даты закрытия стоимости актива на конец месяца по всем 15 компаниям из seltickers:

for(ticker in tickers) {

tmp <- Cl(to.monthly(eval(parse(text=ticker))))

if(is.null(P)){ timeP = time(tmp) }

if( any( time(tmp)!=timeP )) next

else P<-cbind(P,as.numeric(tmp))

seltickers = c( seltickers , ticker ) }

Придаём значение даты:

P = xts(P,order.by=timeP)

Придаём названия тикеров:

colnames(P) <- seltickers

Создаём ряд логарифмированных переменных от P:

R <- diff(log(P))

Убираем январь 2016:

R <- R[-1,]

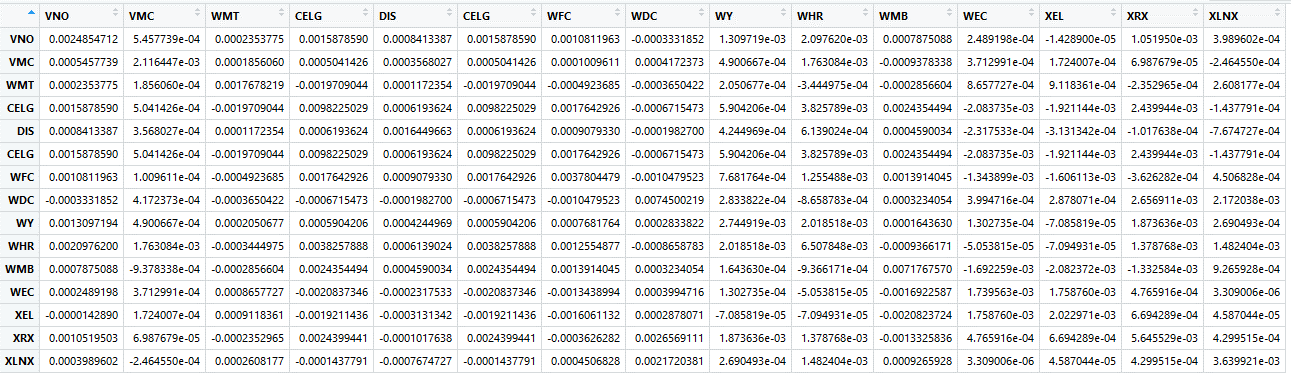
> dim(R)

[1] 23 15

Создаём ковариационную матрицу:

mu <- colMeans(R)

sigma <- cov(R)



Предварительная кластеризация компаний на 5 групп:

x <- t(R) #транспонируем таблицу

kmeans(x, 5, 10000) #разбиваем на 5 кластеров, максимум 10000 иттераций

K-means clustering with 5 clusters of sizes 1, 6, 2, 4, 2

Cluster means:

фев 2016 мар 2016 апр 2016 май 2016 июн 2016 июл 2016 авг 2016 сен 2016 окт 2016 ноя 2016

1 -0.188141517 0.004990653 0.18780324 0.13353144 -0.02420767 0.10272181 0.15325638 0.095179987 -0.05107063 0.05009394

2 0.029084932 0.092143055 0.01768408 0.00619793 -0.01819384 0.05532859 -0.03052738 -0.045030138 -0.03321517 0.07900341

3 0.005070873 -0.007366209 0.03263197 0.02019877 -0.06752504 0.12881844 -0.04979793 -0.020921850 -0.02273814 0.14826140

4 -0.002215658 0.039080582 -0.04883306 0.05443078 0.04143293 0.01930692 -0.02594951 0.001633828 -0.02207420 -0.01550940

5 -0.055898484 0.115661190 -0.14770881 0.08386671 -0.01699496 0.04359059 -0.03125553 0.126721103 -0.01834908 0.02080789

дек 2016 янв 2017 фев 2017 мар 2017 апр 2017 май 2017 июн 2017 июл 2017 авг 2017

1 0.014230447 -0.076729822 -0.01748908 0.043162232 0.034543422 -0.068573170 0.0570807438 0.04834973 -0.06668056

2 0.040067100 0.021178585 0.01456947 -0.022584371 0.004190435 -0.026695467 0.0245552758 -0.01097557 -0.04402785

3 -0.023564669 0.003449783 0.06143966 0.007421168 -0.003058574 -0.080896212 0.1267453889 0.04177333 0.02566292

4 0.045392768 -0.012432796 0.03704144 0.005722335 0.035008809 0.049464333 -0.0349676859 0.02375925 0.02484329

5 -0.001697679 0.173180989 0.01730542 0.028692077 0.027806511 -0.002888571 -0.0002786305 0.01263667 0.04359562

сен 2017 окт 2017 ноя 2017 дек 2017

1 0.009374022 -0.05162657 0.019114362 0.048380312

2 0.030498233 -0.01047814 0.025976221 0.020450280

3 0.048402874 -0.36754958 -0.001387503 0.034410520

4 -0.003138446 0.06671276 0.030310732 -0.032569093

5 0.004853255 -0.03055084 -0.072834816 -0.004442134

Clustering vector:

VNO VMC WMT CELG DIS CELG WFC WDC WY WHR WMB WEC XEL XRX XLNX

2 2 4 3 2 3 2 5 2 2 1 4 4 5 4

Within cluster sum of squares by cluster:

[1] 0.00000000 0.24933300 0.00000000 0.11010902 0.08872268

(between\_SS / total\_SS = 65.0 %)

Available components:

[1] "cluster" "centers" "totss" "withinss" "tot.withinss" "betweenss" "size" "iter"

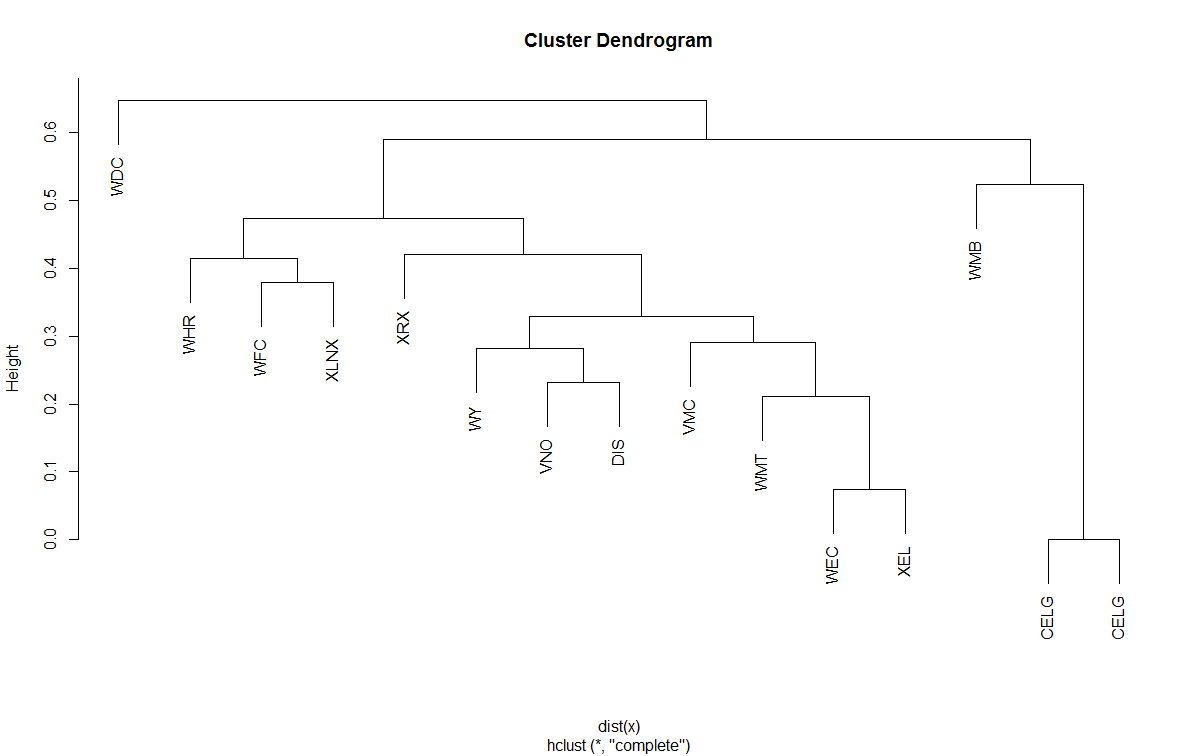
[9] "ifault"

hc <- hclust(dist(x))

View(hc)

plot(hc)

Получаем следующую картину кластеризации:



Задача оптимизации состоит в определении весов портфеля, для которых портфель имеет самый низкий CVaR, и каждая инвестиция может вносить максимум 5% в общий риск CVaR портфеля. Кроме того, веса должны быть положительными, а портфель должен быть полностью инвестирован.

Уровень портфеля CVaR и вкладов CVaR удобно вычисляется с помощью функции ES в пакете PerformanceAnalytics (Carl and Peterson 2010). Для простоты мы предполагаем здесь нормальность, но и оценки вкладов CVaR и CVaR для ненормальных распределений доступны в функции ES.

Ограничение того, что каждый актив может вносить максимум 5% в общий риск CVaR портфеля, вводится путем добавления функции штрафа к целевой функции. Таким образом, мы разрешаем алгоритму поиска рассматривать неосуществимые решения. Портфель, который неприемлем для инвестора, должен быть оштрафован настолько, чтобы его отклонили процессом минимизации, и чем больше нарушение ограничения, тем больше увеличение значения целевой функции

library("PerformanceAnalytics")

Весы должны дополнительно удовлетворить длительное и полное инвестиционное ограничение. Текущая реализация DEoptim позволяет связать ограничения на веса портфеля. Мы обозначим их нижние и верхние.

obj <- function(w) {

if (sum(w) == 0) {

w <- w + 1e-2

}

w <- w / sum(w)

CVaR <- ES(weights = w,

method = " gaussian",

portfolio\_method = "component",

mu = mu,

sigma = sigma)

tmp1 <- CVaR$ES

tmp2 <- max(CVaR$pct\_contrib\_ES - 0.05, 0)

out <- tmp1 + 1e3 \* tmp2

return(out)

}

Полное ограничение инвестиций объясняется двумя способами. Во-первых, мы стандартизируем все веса целевой функции так, чтобы они суммировались до единицы. Во-вторых, мы используем функцию random\_portfolios в PortfolioAnalytics (Boudt et al., 2011) для генерации случайных портфелей, которые удовлетворяют всем ограничениям. Эти случайные портфели будут использоваться в качестве исходной генерации в DEoptim.

install.packages("PortfolioAnalytics")

library("PortfolioAnalytics")

eps <- 0.025

weight\_seq<-generatesequence(min=minw,max=maxw,by=.001,rounding=3)

rpconstraint<-constraint(

assets=N, min\_sum=(1-eps), max\_sum=(1+eps),

min=lower, max=upper, weight\_seq=weight\_seq)

assuming equal weighted seed portfolio

set.seed(1234)

rp<- random\_portfolios\_v1(rpconstraints=rpconstraint,permutations=N\*10)

rp <-rp/rowSums(rp)

Штраф, введенный в целевой функции, является недифференцируемым, и поэтому стандартные подпрограммы оптимизации на основе градиента не могут использоваться. Например, методы L-BFGS-B и Nelder-Mead в optim и nlminb не сходятся.[[1]](#footnote-1)

obj <- function(w) {

if (sum(w) == 0) {

w <- w + 1e-2

}

w <- w / sum(w)

CVaR <- ES(weights = w,

method = "gaussian",

portfolio\_method = "component",

mu = mu,

sigma = sigma)

tmp1 <- CVaR$ES

tmp2 <- max(CVaR$pct\_contrib\_ES - 0.05, 0)

out <- tmp1 + 1e3 \* tmp2

return(out)

}

out <- optim(par = rep(1/N, N), fn = obj,

method = "L-BFGS-B", lower = lower, upper = upper)

out$value

[1] 40.91801

out$message

[1] "NEW\_X"

out <- nlminb(start =rep(1/N, N), objective = obj,

lower = lower, upper = upper)

> out$objective

[1] 49.78416

> out$message

[1] "function evaluation limit reached without convergence (9)"

В отличие от методов на основе градиента, DEoptim предназначен для последовательного поиска хорошего приближения к глобальному минимуму проблемы оптимизации. Для сложных задач, рассматриваемых здесь, производительность DEoptim полностью зависит от используемого алгоритма DE.

Сначала мы рассмотрим текущий дефолтный алгоритм DE в DEoptim, который называется стратегией «локальный к лучшему» с фиксированными параметрами.

Мы определяем сходимость, когда процентное улучшение между итерациями ниже reltol = 1e-6 после steptol = 150 шагов. Для некоторых проблем может потребоваться много итераций до того, как алгоритм DE сходится. Мы устанавливаем максимальное количество итераций, разрешенных на 5000. Прогресс печатается каждые 250 итераций. Как объяснялось выше, начальная генерация устанавливается на rp.

controlDE <- list(reltol=.000001,steptol=150, itermax = 5000,trace = 250, NP=as.numeric(nrow(rp)),initialpop=rp)

set.seed(1234)

start <- Sys.time()

install.packages("DEoptim")

install.packages("ES")

out <- DEoptim(fn = obj, lower = lower, upper = upper, control = controlDE)

Получаем результаты иттераций и увидим оптимальный портфель при 250, 500,750,1000 иттераций:

Iteration: 250 bestvalit: 29.931719 bestmemit: 0.104833 0.257918 0.404040 0.079221 0.254205 0.126085 0.251679 0.154872 0.119788 0.082779 0.305357 0.359120 0.498803 0.109573 0.184483

Iteration: 500 bestvalit: 28.409666 bestmemit: 0.104833 0.296668 0.408969 0.079221 0.254205 0.126085 0.251679 0.154872 0.135798 0.082779 0.305357 0.359120 0.498803 0.113662 0.184483

Iteration: 750 bestvalit: 27.079851 bestmemit: 0.182022 0.562749 0.770015 0.191479 0.520346 0.224284 0.412793 0.331542 0.257442 0.147017 0.544855 0.723806 0.836229 0.203708 0.329996

Iteration: 1000 bestvalit: 25.188382 bestmemit: 0.096749 0.296668 0.408969 0.079703 0.267363 0.110070 0.268689 0.132734 0.135798 0.082779 0.305871 0.359120 0.498803 0.113662 0.184483

out$optim$iter

[1] 1053

> out$optim$bestval

[1] 25.18838

end <- Sys.time()

end - start

Time difference of 1.978032 hours

Таким образом, мы видим, что на итерации 5000 - стратегия «от локального к лучшему» с простым решением, которое лучше, чем полученное с использованием вышеприведенного градиента.

Недавно предложенный алгоритм DE с лучшими свойствами сходимости по сложным задачам является алгоритмом JADE, предложенным Чжан и Сандерсоном (2009). JADE объединяет стратегию «localto-pbest» с адаптивным контролем параметров.

Таким образом, первый строительный блок JADE заключается в том, что алгоритм DE не всегда использует наилучшее решение текущего поколения для решения b100p% c лучших решений с 0 <p ≤ 1. Значение по умолчанию p равно 0,2. Несмотря на то, что эта стратегия более жадная, она, как правило, сходится быстрее. Стратегия «Local-to-pbest» выбирается путем установки стратегии = 6.

controlDE <- list(reltol=.000001,steptol=150, itermax = 5000,trace = 250,

+ strategy=6, c=0,

+ NP=as.numeric(nrow(rp)),initialpop=rp)

> set.seed(1234)

> start <- Sys.time()

> out <- DEoptim(fn = obj, lower = lower, upper = upper, control = controlDE)

Iteration: 250 bestvalit: 38.768872 bestmemit: 0.191108 0.400310 0.821041 0.193644 0.386576 0.200868 0.264134 0.243216 0.303581 0.159960 0.481819 0.705562 0.542110 0.160507 0.329923

Iteration: 500 bestvalit: 30.342129 bestmemit: 0.175753 0.476163 0.782050 0.224550 0.522299 0.149587 0.342080 0.245708 0.222001 0.162280 0.555571 0.609283 0.898087 0.206094 0.354573

Iteration: 750 bestvalit: 28.152653 bestmemit: 0.176301 0.503870 0.821041 0.193644 0.508838 0.208676 0.270558 0.314154 0.289808 0.159960 0.481819 0.705562 0.604250 0.173134 0.331474

Iteration: 1000 bestvalit: 27.318648 bestmemit: 0.176301 0.503870 0.821041 0.193644 0.510030 0.208676 0.270558 0.314154 0.289808 0.159960 0.514649 0.705562 0.604250 0.173134 0.331474

Iteration: 1250 bestvalit: 25.373631 bestmemit: 0.175753 0.550632 0.797179 0.224550 0.522299 0.149587 0.419020 0.291234 0.280385 0.162280 0.555571 0.657592 0.863728 0.206094 0.344538

out$optim$bestval

[1] 25.18823

end <- Sys.time()

end - start

Time difference of 3.706323 mins

Вторая отличительная особенность Чжан и Сандерсона (2009) - познакомить с изучением успешных параметров в алгоритме. При таком подходе вероятность перекрестности при генерации g + 1 устанавливается равной (1 - c) вероятности перекрестности при генерации g плюс c раз в среднем по всем успешным перекрестным вероятностям при генерации g. Аналогичным образом, коэффициент мутации при генерации g + 1 равен 1 - c раз предыдущему коэффициенту мутации плюс c раз средний коэффициент мутации всех успешных мутаций. Возьмем c = .4.

controlDE <- list(reltol=.000001,steptol=150, itermax = 5000,trace = 250,

+ strategy=2, c=.4,

+ NP=as.numeric(nrow(rp)),initialpop=rp)

set.seed(1234)

start <- Sys.time()

out <- DEoptim(fn = obj, lower = lower, upper = upper, control = controlDE)

Iteration: 250 bestvalit: 17.438112 bestmemit: 0.165006 0.527172 0.805634 0.204930 0.472953 0.204309 0.579843 0.328976 0.268123 0.148494 0.554882 0.670138 0.995549 0.185785 0.312565

Iteration: 500 bestvalit: 16.694651 bestmemit: 0.164018 0.513814 0.786272 0.199553 0.459224 0.199556 0.567733 0.321284 0.260377 0.144380 0.547001 0.647259 0.987008 0.180862 0.304677

Iteration: 750 bestvalit: 16.692780 bestmemit: 0.161903 0.507161 0.776112 0.196974 0.453288 0.196974 0.560389 0.317132 0.257014 0.142510 0.539926 0.638871 0.974257 0.178520 0.300734

> out$optim$bestval

[1] 16.69278

> end <- Sys.time()

> end - start

Time difference of 1.953265 mins

Стратегия «local-to-1best» с адаптивным контролем параметров сходится явно слишком быстро. Это сочетание стратегии «local-to-pbest» с адаптивным контролем параметров, которое является наиболее успешным в решении нашей проблемы.

controlDE <- list(reltol=.000001,steptol=150, itermax = 5000,trace = 250,

+ strategy=6, c=.4,

+ NP=as.numeric(nrow(rp)),initialpop=rp)

set.seed(1234)

start <- Sys.time()

out <- DEoptim(fn = obj, lower = lower, upper = upper, control = controlDE)

Iteration: 250 bestvalit: 54.211693 bestmemit: 0.351513 0.543552 0.639250 0.327869 0.479698 0.286669 0.432713 0.472608 0.473375 0.220016 0.030773 0.793255 0.797444 0.223000 0.368042

Iteration: 500 bestvalit: 40.382974 bestmemit: 0.198159 0.718164 0.997997 0.244573 0.722089 0.211383 0.424635 0.306223 0.356329 0.175650 0.621089 0.888888 0.724851 0.296099 0.508564

> out$optim$iter

[1] 638

> out$optim$bestval

[1] 40.38297

end <- Sys.time()

> end - start

Time difference of 3.358905 mins

Мы видим, что с JADE DEoptim сходится в пределах 538 итераций до 40,38297, что является самым низким, полученным всеми способами, рассмотренными в виньетке.

Эта виньетка иллюстрирует, что для сложных задач производительность DEoptim, таким образом, полностью зависит от используемого алгоритма DE. Рекомендуется, чтобы пользователи опробовали несколько алгоритмов DE, чтобы узнать, какой из них наиболее приспособлен для решения этой проблемы.

Кроме того, DE является стохастическим оптимизатором и обычно будет находить почти оптимальное решение, зависящее от разброса. Функция optimize.portfolio.parallel в PortfolioAnalytics позволяет запускать произвольное количество наборов портфолио параллельно, чтобы развить доверительные диапазоны вокруг вашего решения. Он основан на пакете foreach от REVolution (REVolution Computing 2009).

**ВСЕ графики из главы 18 "Portfolio optimization with R/Rmetrics”.**

**library(fPortfolio)**

library(fPortfolio)

lppData <- DowJones30[1:500, 1:15] #отбор 15 компаний с индекса DowJones30 (500 наблюдений)

lppData2 <- lppData2[-1,]

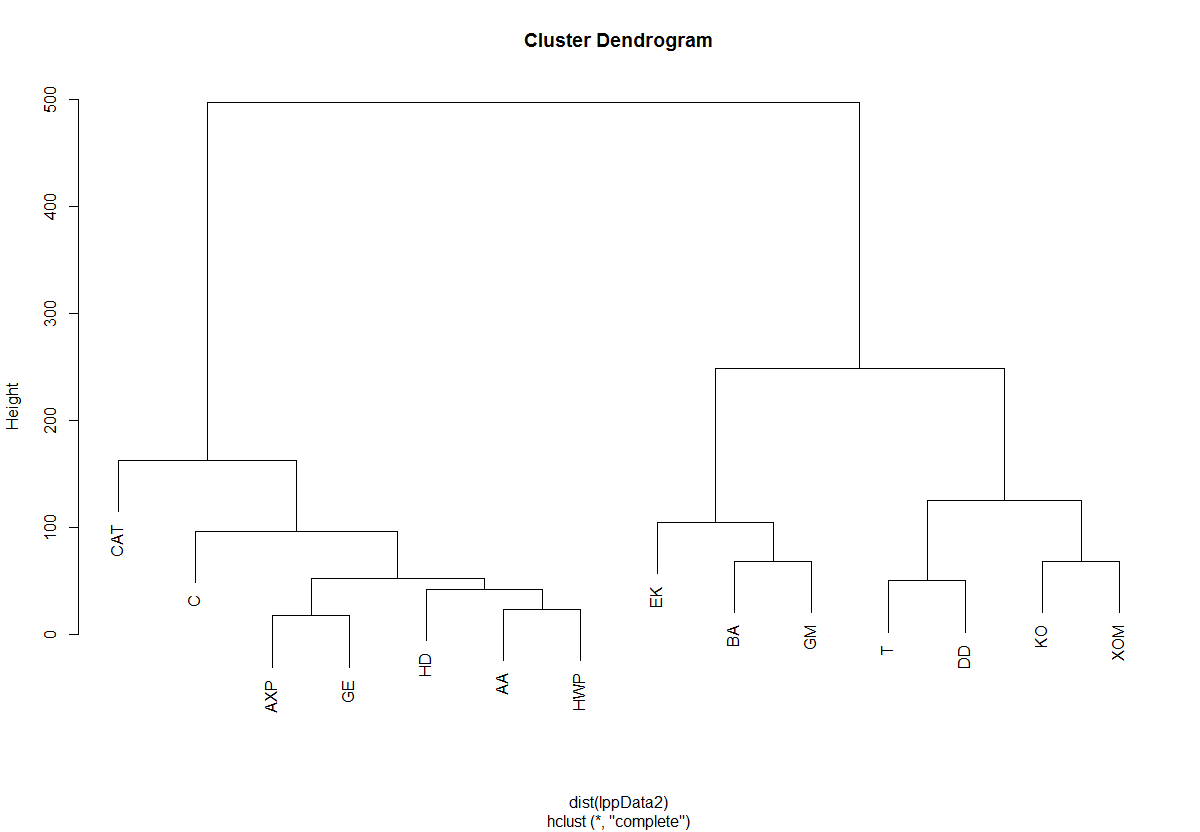
kmeans(lppData2, 5, 10000)

hc <- hclust(dist(x))

print(hc)

plot(hc)

Получаем следующую картину кластеризации:



+++++++

lppData <- 100 \* SMALLCAP.RET[, 1:15]

> colnames(lppData)

[1] "MODI" "MGF" "MEE" "FCEL" "OII" "SEB" "RML" "AEOS" "BRC" "CTC" "TNL" "IBC" "KWD" "TOPP" "RARE"

lppSpec <- portfolioSpec()

setNFrontierPoints(lppSpec) <- 7

longFrontier <- portfolioFrontier(lppData, lppSpec)

print(longFrontier)

Title:

MV Portfolio Frontier

Estimator: covEstimator

Solver: solveRquadprog

Optimize: minRisk

Constraints: LongOnly

Portfolio Points: 5 of 6

Portfolio Weights:

MODI MGF MEE FCEL OII SEB RML AEOS BRC CTC TNL IBC KWD TOPP RARE

1 0.0000 0.8412 0.0000 0.0171 0.0006 0.0006 0.0318 0.0399 0.0000 0.0000 0.0000 0.0076 0.0000 0.0518 0.0093

2 0.0000 0.5499 0.0000 0.0835 0.0000 0.0000 0.0000 0.1312 0.0000 0.0000 0.0000 0.0536 0.0000 0.1116 0.0702

3 0.0000 0.2302 0.0000 0.1550 0.0000 0.0000 0.0000 0.2239 0.0000 0.0000 0.0000 0.0980 0.0000 0.1711 0.1218

4 0.0000 0.0000 0.0000 0.2194 0.0000 0.0000 0.0000 0.3324 0.0000 0.0000 0.0000 0.0766 0.0000 0.2235 0.1482

6 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Covariance Risk Budgets:

MODI MGF MEE FCEL OII SEB RML AEOS BRC CTC TNL IBC KWD TOPP RARE

1 0.0000 0.5110 0.0000 0.0923 0.0009 0.0006 -0.0012 0.2473 0.0000 0.0000 0.0000 0.0059 0.0000 0.1264 0.0168

2 0.0000 0.0089 0.0000 0.2766 0.0000 0.0000 0.0000 0.5061 0.0000 0.0000 0.0000 0.0068 0.0000 0.1423 0.0593

3 0.0000 -0.0049 0.0000 0.3039 0.0000 0.0000 0.0000 0.5124 0.0000 0.0000 0.0000 0.0044 0.0000 0.1260 0.0582

4 0.0000 0.0000 0.0000 0.3063 0.0000 0.0000 0.0000 0.5454 0.0000 0.0000 0.0000 -0.0025 0.0000 0.1079 0.0429

6 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Target Returns and Risks:

mean Cov CVaR VaR

1 1.0635 1.7577 2.2007 1.4210

2 2.4371 4.6470 7.1716 6.0947

3 3.8106 8.0547 12.9088 10.5372

4 5.1842 11.5655 18.7518 14.1235

6 7.9314 23.9674 38.2992 29.7762

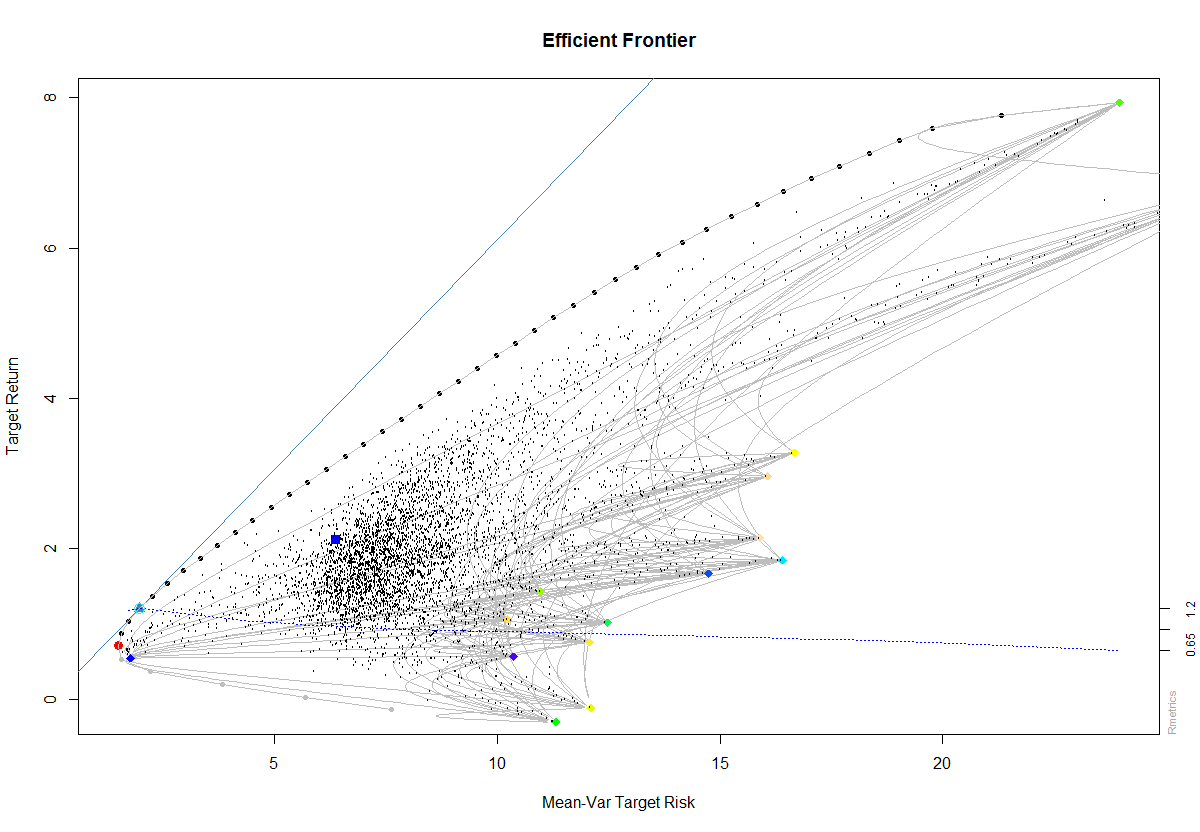
Description:

Tue Mar 06 00:17:26 2018 by user:Katya Semenova

***interactively plot the efficient frontier***

> longFrontier <- portfolioFrontier(lppData)

> plot(longFrontier)



**Список литературы**

* + 1. Diethelm Würtz, Yohan Chalabi, William Chen, Andrew EllisPortfolio Optimization with R/Rmetrics
    2. Iacus S.M. Option Pricing and Estimation of Financial Models with R
    3. <https://cran.r-project.org/doc/contrib/Shipunov-rbook.pdf>
    4. <http://aakinshin.net/ru/blog/post/r-functions/>

<https://cran.r-project.org/web/packages/distr/distr.pdf>

* + 1. <https://www.statmethods.net/advgraphs/probability.html>
    2. <http://r-analytics.blogspot.ru/2012/12/r.html#.Wpeq9LTJwkR>
    3. Статистический пакет R: теория вероятностей и математическая статистика.:http://www.ievbras.ru/ecostat/Kiril/R/Biblio/R\_rus/%C7%E0%F0%FF%E4%EE%E2%20%F7%E0%F1%F2%FC\_2.pdf
    4. <https://cran.r-project.org/web/packages/DEoptim/vignettes/DEoptimPortfolioOptimization.pdf>
    5. <https://cran.r-project.org/web/packages/PortfolioAnalytics/PortfolioAnalytics.pdf>
    6. <https://cran.r-project.org/web/packages/fPortfolio/fPortfolio.pdf>
    7. <https://s3.amazonaws.com/assets.datacamp.com/blog_assets/xts_Cheat_Sheet_R.pdf?tap_a=5644-dce66f&tap_s=10907-287229>
    8. <https://habrahabr.ru/company/infopulse/blog/310288/>
    9. <https://www.coursera.org/learn/ekonometrika/home/week/8>
    10. http://www.algorithmist.ru/2011/05/clustering-with-example-in-r.html

1. https://cran.r-project.org/web/packages/DEoptim/vignettes/DEoptimPortfolioOptimization.pdf [↑](#footnote-ref-1)