Portfolio Optimization

Jesse Keränen

11/24/2021

Prologue

In this file i try to find optimal portfolio from given set of assets. I don't take a stand whether investor should invest in certain stock or not. Rather than that I try to define which amount of capital investor should invest in each stock she has decided to invest into. This decision is done in Markowitz's classical mean variance framework. The Basic idea is to maximize return on given risk level (or vice versa). To estimate single optimal portfolio I use Sharpe ratio.

Mentions

One thing worth of mentioning and paying information is frequency of data. I use monthly data, but in when calculating Sharpe ratio I use annualized returns and standard deviations. Most of the time when I speak about portfolio returns, I mean expected returns instead of realized returns.

Data

Here the data is collected. Data is collected from Guandl data base. For more detailed information refer to VaR file. Returns are simple returns calculated from adjusted monthly prices.

```
library(ggrepel)
library(ggplot2)
library(pheatmap)
library(Quandl)
library(tidyverse)
library(data.table)
library(plotly)


Quandl.api_key("bx1qdehfWXg6SNKnicQC")

names <- c("AAPL", "MSFT", "TSLA", "GOOGL", "AMZN")

# For monthly data use collapse = "monthly"
prices <- as.data.table(Quandl(c("WIKI/AAPL.11","WIKI/MSFT.11",
        "WIKI/TSLA.11", "WIKI/GOOGL.11", "WIKI/AMZN.11"), start_date
        = "2010-06-01", end_date = "2015-03-01", collapse = "monthly"))

colnames(prices) <- c("Date", "AAPL", "MSFT", "TSLA", "GOOGL", "AMZN")
```

```
# Tidying data table
prices <- melt(prices, id.vars = "Date", measure.vars = names, variable.name
               = "Company", value.name = "Adj_Close")

# Calculate simple price changes
prices[, "Returns" := ((prices$Adj_Close)-lag(prices$Adj_Close))
       /lag(prices$Adj_Close)]

# This should be changed. Now calculating returns code doesn't
# recognize that company changes and calculate return between last
# price of previous company and first price of latter company.
prices <- prices[Date=="2010-06-30", Returns := NA]
prices <- drop_na(prices)
help(ggplotly)
```

Expected returns and covarance matrix

Here expected returns are calculated for each asset as well as correlations between each asset. Expected returns are simply calculated by taking arithmetic mean of the monthly returns.

```
mean_std <- prices[, .(means = mean(Returns), stds = sd(Returns)), by= Company]

prf <- as.data.table(split(prices[,4] ,prices$Company))

corr_matrix <- cor(as.data.table(split(prices[,4] ,prices$Company)))
corr_matrix
```

```
##             AAPL       MSFT         TSLA        GOOGL       AMZN
## AAPL  1.00000000 0.32674129  0.015066068  0.329066216 0.33228890
## MSFT  0.32674129 1.00000000  0.031560082  0.336164908 0.09420324
## TSLA  0.01506607 0.03156008  1.000000000 -0.006398831 0.03458292
## GOOGL 0.32906622 0.33616491 -0.006398831  1.000000000 0.27594008
## AMZN  0.33228890 0.09420324  0.034582916  0.275940077 1.00000000
```

```
cov_matrix <- cov(as.data.table(split(prices[,4] ,prices$Company)))
cov_matrix
```

```
##               AAPL         MSFT          TSLA         GOOGL         AMZN
## AAPL  0.0052389324 0.0013807740  0.0002069998  0.0017124912 0.0018714985
## MSFT  0.0013807740 0.0034087441  0.0003497711  0.0014111502 0.0004279719
## TSLA  0.0002069998 0.0003497711  0.0360327358 -0.0000873319 0.0005108136
## GOOGL 0.0017124912 0.0014111502 -0.0000873319  0.0051694856 0.0015437988
## AMZN  0.0018714985 0.0004279719  0.0005108136  0.0015437988 0.0060548656
```

First we can illustrate how risk and return of our portfolio distribute with different weights. We create n sets of random numbers.

```
n <- 15000

port_return <- vector('numeric', length = n)
port_std <- vector('numeric', length = n)
```

```
port_sharpe <- vector('numeric', length = n)

for (i in 1:n){
  rand_weights <- runif(unique(prices$Company))
  rand_weights <- rand_weights/sum(rand_weights)

  port_return[i] <- mean_std$means %*% rand_weights *12
  port_std[i] <- sqrt(rand_weights %*% cov_matrix %*% matrix(rand_weights)) *sqrt(12)
  port_sharpe[i] <- port_return[i]/port_std[i]
}
dt <- data.table(port_return, port_std, port_sharpe)

# Try to make it so that hovering over point shows the weights.
fig <-ggplot(dt, aes(dt$port_std, dt$port_return, color=dt$port_sharpe)) + geom_point() + labs(x = "Sta

ggplotly(fig)
```

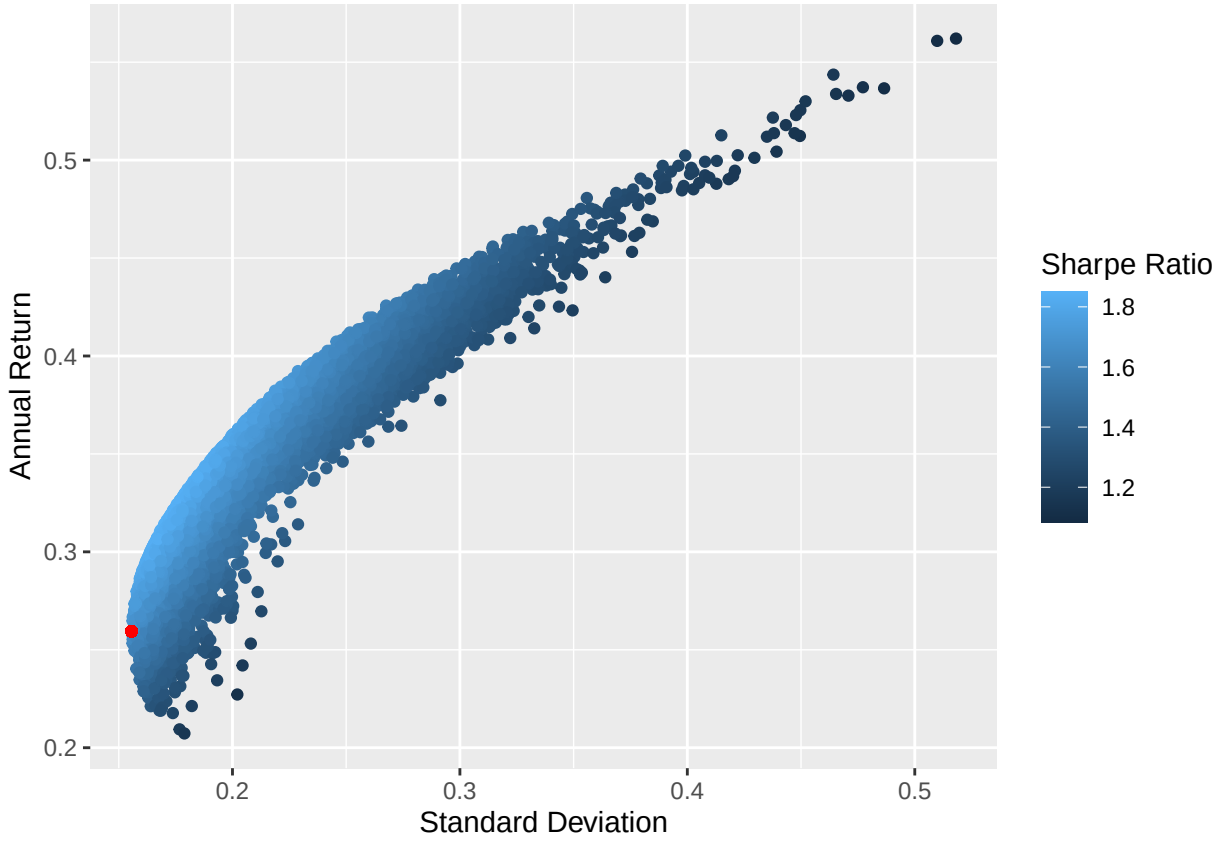
From this image we could approximate efficient frontier and see about where the highest Sharpe ratio lies. Even thought, with quite high possibility we could find a portfolio seemingly close to actual portfolio that provides highest possible Sharpe ratio, it isn't too efficient to create thousands of portfolios and then try to find best of these. We can use actual optimization methods to find probably more precise estimate of the optimal portfolio with possibly less calculations.

```
r <- 1000
weights <- c(0.2, 0.2, 0.2, 0.2, 0.2)

objective <- function(returns, x, cov){
  return (list(-(returns %*% x)*12/(sqrt((x %*% cov %*%
        matrix(x)))*sqrt(12)),x, sum(x)-1))
}
p <- objective(mean_std$means, weights, cov_matrix)

alpha <- function(return, x, f, objective, cov){
  constrains <- f(return, x, cov_matrix)
  return (sum(min(constrains[[2]], 0))^2 + sum(constrains[[3]]^2))
}

penalized_function <-function(x, alpha, objective){
  return (objective(mean_std$means, x, cov_matrix)[[1]] + r*alpha(mean_std$means, x,
          objective, cov_matrix))
}

opt <- optim(weights,(function(x) penalized_function(x, alpha, objective)))
```

If we use Sharpe ratio to measure how good our portfolio is we have found the optimal portfolio. If we also believe Tobin's separation theorem we wouldn't have to bother our heads about other portfolios lying on the efficient frontier. Nevertheless this would require us to be able to lend and borrow at risk free rate, which in real life isn't too realistic assumption. Since investors have different risk bearing levels, we have to calculate optimal portfolio for other risk levels as well. Since we know that the return of the portfolio is weighted average of the returns of the assets in it, we know that return of the portfolio can't exceed the return of the return of the asset with highest expected return. However we can't make similar assumption about the minimum variance portfolio, because of the correlations. Yet we can estimate minimum variance portfolio by chancing little bit our objective function. After finding the return (or risk) of the minimum variance

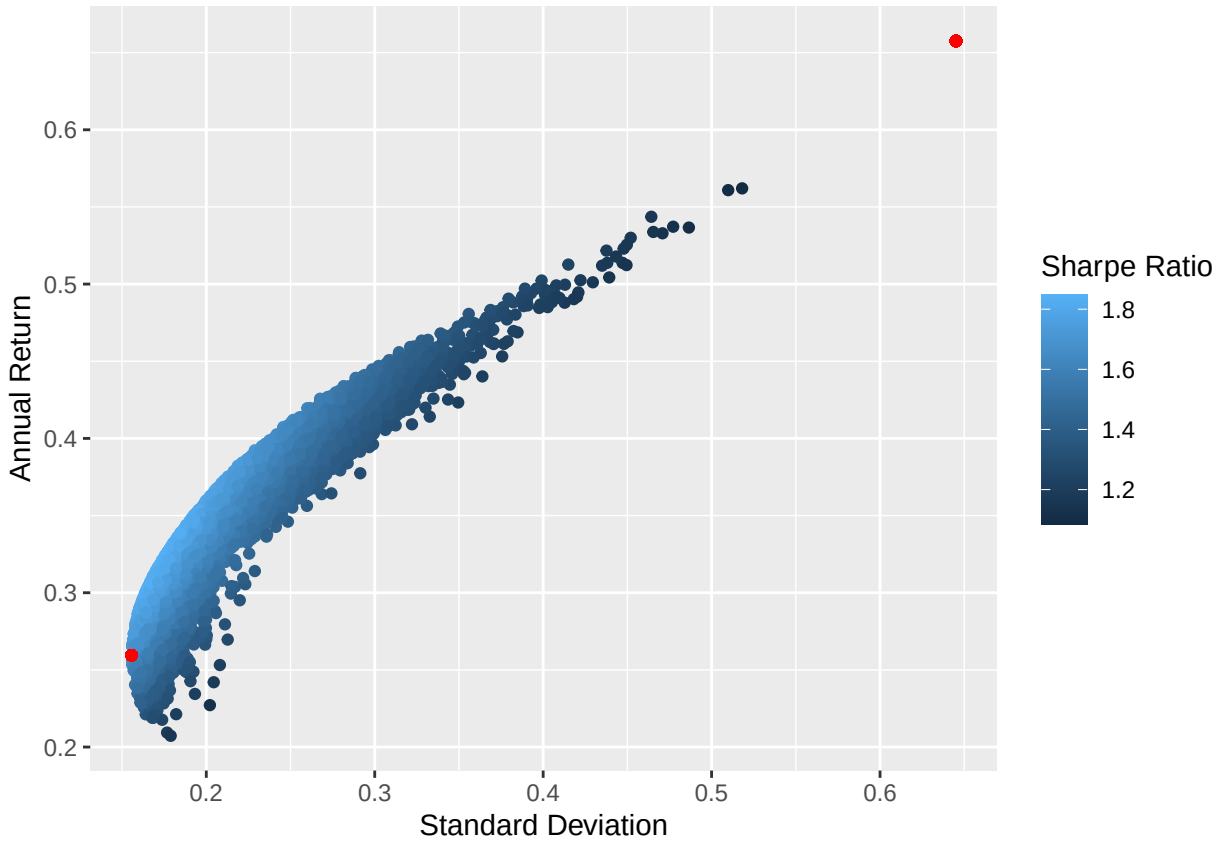
portfolio we can increment our desired return (or risk) level little by little and calculate optimal portfolio for each level until we reach return of the highest return asset. This way we can approximate our efficient frontier. Lets start with finding minimum variance portfolio.

```
# Now the objective function minimizes portfolio variance.
objective <- function(returns, x, cov){
  return (list((sqrt(x %*% cov %*% matrix(x))*sqrt(12)),
               x, sum(x)-1))
}

opt <- optim(weights,(function(x) penalized_function(x, alpha, objective)))

min_var_ret <- mean_std$means %*% matrix(opt[[1]]) * 12
min_var_std <- opt[[2]]

ggplot(dt, aes(dt$port_std, dt$port_return, color=dt$port_sharpe)) + geom_point() + labs(x = "Standard
  geom_point(aes(min_var_std, min_var_ret), color = "red")
```



Next let's see what asset had the highest expected return. We find the asset with the highest return and plot it in above chart.

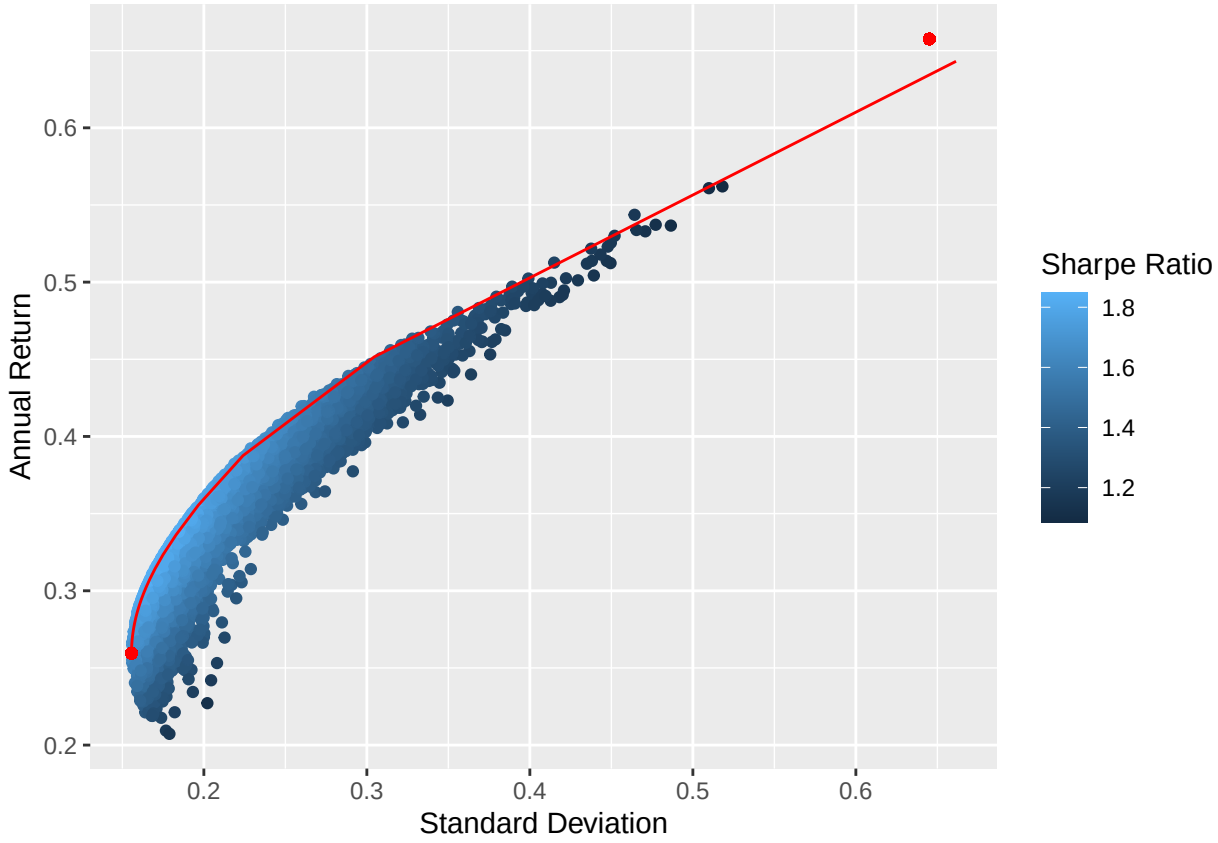
```
hr_comp <- mean_std[which.max(means)]

# Remember to annualize return and standard deviation
ggplot(dt, aes(dt$port_std, dt$port_return, color=dt$port_sharpe)) + geom_point() +
  labs(x = "Standard Deviation", y = "Annual Return", color = "Sharpe Ratio") +
  geom_point(aes(min_var_std, min_var_ret), color = "red") +
  geom_point(aes(hr_comp$means*12, hr_comp$stds*sqrt(12)), color = "red")
```



We need to again little bit change our objective function. To be more precisely we don't change our actual objective function, but we introduce new constraint.

```
n <- 100
objective <- function(returns, x, cov, risk){
  return (list(sqrt(x %*% cov %*% matrix(x))*sqrt(12), x,
        mean_std$means %*% x *12 - risk, sum(x)-1))
}

alpha <- function(return, x, objective, cov, risk){
  constrains <- objective(return, x, cov_matrix, risk)
  return (sum(min(constrains[[2]], 0))^2 +
            sum(constrains[[3]]^2 + constrains[[4]]^2))
}

penalized_function <-function(x, alpha, objective, risk){
  return (objective(mean_std$means, x, cov_matrix, risk)[[1]] + r*alpha(mean_std$means, x,
          objective, cov_matrix, risk))
}

diff <- hr_comp$means*12 - min_var_ret
pop <- matrix(nrow = n, ncol = 3)
weight_mat <- matrix(nrow = n, ncol = 5)

for(i in 1:n){
  risk <- min_var_ret + diff/i
```

```
  opt <- optim(weights, (function(x) penalized_function(x, alpha,
          objective, risk)),method = "Nelder-Mead",  control=list(maxit=1000))
  pop[i, 1] <- mean_std$means %*% matrix(opt[[1]])*12
  pop[i, 2] <- opt[[2]]
  pop[i, 3] <- pop[i,1]/pop[i, 2]
  weight_mat[i,] <- opt[[1]]

  # I believe that since we only increase our return level by little, next
  # allocation of optimal weights is to found close to previous. That's
  # why I set the initial guess to be optimal solution from previous
  # iteration
  weights <- opt[[1]]
}
pop <- as.data.table(pop)

ggplot(dt, aes(dt$port_std, dt$port_return, color=dt$port_sharpe)) + geom_point() +
  labs(x = "Standard Deviation", y = "Annual Return", color = "Sharpe Ratio") +
  geom_point(aes(min_var_std, min_var_ret), color = "red") +
  geom_point(aes(hr_comp$means*12, hr_comp$stds*sqrt(12)), color = "red") +
  geom_line(data = pop, aes(pop$V2, pop$V1), color = "red")
```



We can see that our efficient frontier lands quite nicely where we estimated it to land with random portfolios. Next would might be interesting to see how weights on the assets change on different risk levels.

```
weight_dt <- as.data.table(weight_mat)
weight_dt <- cbind(round(weight_dt, digits = 3), pop[, 2])
colnames(weight_dt) <- c(names, "Std")
```

```
weight_dt <- melt(weight_dt,id.vars = "Std", measure.vars = names, variable.name = "Company", value.nam

ggplot(weight_dt, aes(Std, Weight, fill=Company)) + geom_area()
```