**Overview**

There are several ways to do portfolio optimization out there, each with its advantages and disadvantages. Today I am going to show another method to perform portfolio optimization that works very well in large datasets because it produces very robust weights, which results in a good out-of-sample performance. This technique is called Parametric Portfolio Policies (PPP) and it was proposed by Brandt, Santa-Clara and Valkanov in 2009 .

Portfolio Optimization Techniques

Today we will use an online convex optimization technique to build a very simple algorithm for portfolio allocation. Of course this is just an illustrative post and we are going to make some simplifying assumptions. The objective is to point out an interesting direction to approach the problem of portfolio allocation.

The algorithm used here is the Online Gradient Descendent (OGD) and we are going to compare the performance of the portfolio with the Uniform Constant Rebalanced Portfolio and the Dow Jones Industrial Average index. You can skip directly to Implementation and Example if you already know what an online algorithm is.

For those who don’t know what Online Convex Optimization is…

From now on, we will say that p represents a point in dimension N, where N is the number of possible stocks to invest. Each of it’s coordinates represent the amount invested in the respective stock. For example, suppose that N = 3 and p_t = [0.25, 0.25, 0.5]^\prime. Then, our portfolio at time $latex  t$ are composed of 25\% from stock one, 25\% from stock two, and 50\% from stock three.

Online Convex Optimization is based on the idea of minimizing the so-called Regret function, which is basically an idea of how far from a good player you are (some kind of optimal player that you want to be). In the set-up of portfolio management, the Regret function is defined as:

\displaystyle Regret=\sum_{i=1}^tlog(p^*r_i)-\sum_{i=1}^tlog(p_ir_i)

The first part on the right hand side is the cumulative log return of the portfolio p^*, and the second part, is the cumulative log return of our portfolio until time t, which we have played strategy p_i,~~~i=1,2,\dots,t. What is interesting is that p^* is a very special case of portfolio. It represents the Best Constant Rebalanced Portfolio (CRP)…it means that if someone could see the future and knew upfront all the returns of all the stocks and had to play a fixed portfolio proportion at time zero, p^* would be his choice. In the worst case scenario, the best CRP is equal to the best stock in the period, and the best CRP is the portfolio that we want to track.

The OGD algorithm is quite simple, we are going to update our position every week based on the gradient of the Regret function in the currently strategy. Here it follows:

1. First play p_1 = \frac{1}{n} \textbf{1}
2. For t > 1,
3. b_{t-1} = \sum_{\tau = 1}^{t-1}\nabla(\textit{Regret})
4. \tilde{p}_t = p_{t-1} - \eta b_{t-1}
5. p_t = \prod_{S_n}(\tilde{p}_t)
6. \prod_{S_n}(q) = arg\min_{x \in S_n} (x - q)^\prime(x-q)

This algorithm is proved to have Regret of order O(GD\sqrt{T}) if \eta = \frac{G}{D\sqrt{T}}. There is another similar algorithm which takes second order information and is quite similar to an online version of the Newton’s method. The name, not surprisingly is Online Newton Step. See Agrawal et al (2006) for further details.

Implementation and Example

In our set-up the set of possible portfolios will not allow us to borrow money or short selling stocks, so every position should be bigger then zero and the sum of all positions should be one. This set is called simplex and define polytope. Basically it’s a triangle in higher dimension.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | library(quantmod)    symbols = c('MMM', 'AXP', 'AAPL', 'BA', 'CAT', 'CVX', 'CSCO', 'KO', 'DIS', 'DD', 'XOM', 'GE',    'GS', 'HD', 'IBM', 'INTC', 'JNJ', 'JPM', 'MCD', 'MRK', 'MSFT', 'NKE', 'PFE', 'PG',    'TRV', 'UTX', 'UNH', 'VZ', 'V', 'WMT')    for (i **in** 1:length(symbols)) getSymbols(symbols[i], from = '2007-01-03', to = '2017-06-21')    # Building weekly returns for each of the stocks  data = sapply(symbols, **function**(x) Ad(to.weekly(get(x))))  data = Reduce(cbind, data)  data\_returns = apply(data, 2, **function**(x) diff(log(x))) #log returns  colnames(data\_returns)= symbols  data\_returns[is.na(data\_returns)] = 0 # VISA hasnt negotiations between 2007 and 2008 |

The OGD algorithm is implemented next. It uses the quadprogXT package to solve the projection onto the viable set. We are going to test two different values for \eta to check the algorithm sensibility.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | library(quadprogXT)    OGD = **function**(base, eta) {      # Gradient of Regret Function    gradient = **function**(b, p, r) b + r/(p%\*%r)      # Projection onto viable Set    proj = **function**(p) {        Dmat = diag(length(p))      Amat = cbind(diag(rep(1, length(p))), -1)      bvec = c(rep(0, length(p)), -1)        fit = solveQPXT(Dmat = Dmat, dvec = p, Amat = Amat, bvec = bvec)        return(fit$solution)    }      T = nrow(base)    N = ncol(base)      r = as.matrix(base) + 1 # this is because the algo doesnt work directly with log returns    p = matrix(0, nrow = N, ncol = T); p[,1] = 1/N # initial portfolio    b = matrix(0, nrow = N, ncol = T); b[,1] = 0      for (i **in** 2:T) {      b[,i] = gradient(b[,i-1], p[,i-1], r[i-1,]) # calculating gradient      p.aux = p[,i-1] + eta\*b[,i] # what we would like to play      p[,i] = proj(p.aux) # projection in the viable set    }      return(list('portfolio' = p,'gradient' = b))  }    # testing two etas  portfolio1 = OGD(base = data\_returns, eta = 1/100)  portfolio2 = OGD(base = data\_returns, eta = 1/1000) |

Lets build a small function to calculate the compound return of portfolios taking their log return as input.

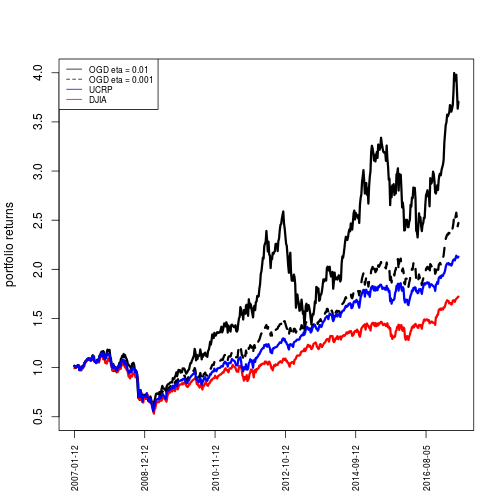
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | compound\_return = **function**(portfolio, r) {      return\_OGD = c(); return\_OGD[1] = portfolio$portfolio[,1]%\*%r[1,]    portfolio\_value = c(); portfolio\_value[1] = 1 + portfolio$portfolio[,1]%\*%r[1,]      for (i **in** 2:nrow(r)) {      return\_OGD[i] = portfolio$portfolio[,i]%\*%r[i,]      portfolio\_value[i] = portfolio\_value[i-1]\*(1 + return\_OGD[i])    }     return(list('value' = portfolio\_value, 'return' = return\_OGD))  } |

Now let’s check the performance of the OGD algorithm with both 0.01 and 0.001 values for \eta. We are going to compare the cumulative return first with the Uniform Constant Rebalanced Portfolio and the DJIA index. Them we will check the compound returns of DJIA stocks individually to see if the algorithm tracked the best stocks like we want them to do.

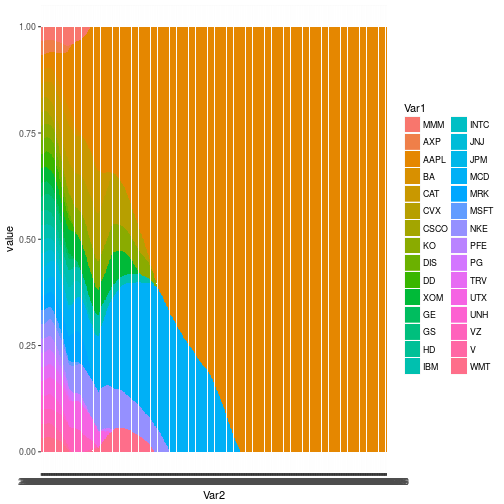
|  |  |  |
| --- | --- | --- |
| 1  2 | # Dow Jones  getSymbols('^DJI', src = 'yahoo', from = '2007-01-03', to = '2017-06-21') | |
| 1 | ## [1] "DJI" |

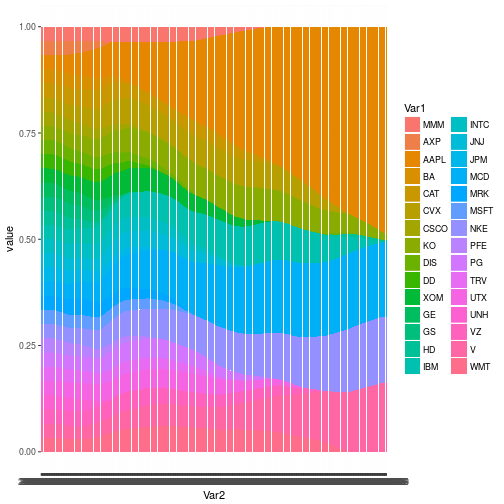
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | DJIA\_returns = as.numeric(cumprod(weeklyReturn(DJI) + 1))    # Individual stocks  stocks\_returns = apply(data\_returns + 1, 2, cumprod)    # Our portfolios  portfolio\_value1 = compound\_return(portfolio1, data\_returns)  portfolio\_value2 = compound\_return(portfolio2, data\_returns) |

In the figures below we can see the performance of the strategies that were implemented. Both portfolios had a surprisingly perform, much better than DJIA and UCRP portfolio.



Indeed, if \eta is relatively big, the algorithm quickly found Apple as the best stock, and start pursuing it. If \eta is relative small, them the algorithm takes more time to adapt, but this makes the portfolio more stable.





To finish the comparison between our portfolios and DJIA, lets do a very simple risk analysis with just empirical quantiles using historical results. For a proper risk analysis we should model the conditional volatility of the portfolio, but this will be discussed in another post.

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | risk\_analysis = **function**(return) {      report = matrix(0, ncol = 2, nrow = 2)      report[1,] = quantile(return, probs = c(0.01, 0.05))    report[2,] = c(mean(return[which(return <= report[1,1])]), mean(return[which(return <= report[1,2])]))      rownames(report) = c('VaR', 'CVaR')    colnames(report) = c('1%', '5%')      return(round(report, 3))  }    report1 = risk\_analysis(portfolio\_value1$return)  report2 = risk\_analysis(portfolio\_value2$return)  report\_DJIA = risk\_analysis(weeklyReturn(DJI)) | | |
|  | **1%** | **5%** |
| VaR | -0.091 | -0.047 |
| CVaR | -0.119 | -0.074 |
| Portfolio1 eta = 0.01 | | |

|  |  |  |
| --- | --- | --- |
|  | **1%** | **5%** |
| VaR | -0.057 | -0.039 |
| CVaR | -0.085 | -0.055 |
| Portfolio2 eta = 0.001 | | |

|  |  |  |
| --- | --- | --- |
|  | **1%** | **5%** |
| VaR | -0.060 | -0.040 |
| CVaR | -0.084 | -0.055 |
| DJIA index | | |

It’s quite evident that setting a big \eta, like we did in portfolio1, may have caused problems with VaR and Conditional VaR since Apple is a volatile stock. For portfolio2, where \eta is small the portfolio takes time to change, the historical VaR and CVaR are very similar to what we already observed in DJIA, but the returns were much better.

A portfolio is a set of weights w_{i,t}that states the proportion of wealth in asset ias time t. For example, in a Markowitz portfolio we get w_{i,t}by minimizing the portfolio variance subject to some target return. This is like estimating one weight for each asset if you think of weights as parameters of a model. If you have 1000 assets you will have to estimate 1000 parameters. Naturally, you can’t expect a good out-of-sample performance from a simple Markowitz portfolio with 1000 assets just as you can’t expect a good performance from a simple regression with 1000 variables. The Parametric Portfolio Policy (PPP) is made exactly to this high-dimensional case. It parametrizes the weights on some asset characteristics in a way that we have to estimate only one parameter for each characteristic. Consider the following portfolio policy:

\displaystyle w_{i,t} = \bar{w}_{i,t} + \frac{1}{N_t}\theta'\hat{x}_{i,t}

Here, the portfolio w_{i,t}is a function of a base portfolio \bar{w}_{i,t}, which will be a simple equal weighted portfolio in my example. \hat{x}_{i,t}is a vector of characteristics for asset iat time t, \thetais the vector of parameters and N_tis the number of available assets. Note that \hat{x}_{i,t}is standardized cross-sectionally to make sure the sum of the weights in each period is one. If we have two characteristics we only need to estimate two parameters and use the policy equation to recover all weights in the portfolio. Unfortunately using a policy like this makes the portfolio much more restricted than the Markowitz case. In a Markowitz portfolio weights can be anything as long as the sum 1 but in the PPP the portfolio must satisfy the policy equation.

Now that we have the policy we need a way to obtain \theta. We can use utility functions and try to maximize the portfolio utility. In the equation below we have the populational case in the left and the sample case in the right.

\displaystyle \max_{\theta}E[u(r_{p,t+1})] \sim \max_{\theta} \frac{1}{T}\sum_{t=0}^{T-1}u(r_{p,t+1}) 

where:

\displaystyle r_{p,t+1} = \sum_{i=1}^{N_t} w_{i,t}r_{i,t+1}

Putting all together we have:

\displaystyle \max_{\theta} \frac{1}{T}\sum_{t=0}^{T-1}u\left(\left( \bar{w}_{i,t} + \frac{1}{N_t}\theta'\hat{x}_{i,t}\right)r_{i,t+1}\right)

where the utility function will be the Constant Relative Risk Aversion (CRRA) with relative risk aversion \gamma, define as:

\displaystyle u(x)=\frac{x^{1-\gamma}-1}{1-\gamma}

**The data**

The PPP parametrizes the weights on characteristics of the stocks, and although we can get a lot of characteristics from returns such as momentum and variance, we are likely to need something else to have good results. In the original paper Brandt Santa-Clara and Valkanov used momentum, book-to-market ratio and market capitalization (mktcap) . I will use only a momentum measure and the mktcap because they are easier to get and need no complex treatment.

The code below uses the package tidyquant to download data from all S&P500 stocks from Yahoo Finance. The code takes some time to download all data. If you want to replicate the exact example. My data starts at 2008-01-02 and ends at 2018-02-06. If you use a different time you will obviously find different results.

library(tidyverse)

library(reshape2)

library(lubridate)

library(tidyquant)

prices=tq\_index("sp500") %>%

tq\_get(get = "stock.prices", complete\_cases = TRUE)

## # A tibble: 6 x 12

## symbol company weight sector share… date open high low close

##

## 1 MSFT Microso… 0.0306 Inform… 9.22e⁷ 2008-01-02 45.9 46.1 44.9 35.2

## 2 MSFT Microso… 0.0306 Inform… 9.22e⁷ 2008-01-03 45.2 45.7 44.7 35.4

## 3 MSFT Microso… 0.0306 Inform… 9.22e⁷ 2008-01-04 45.1 45.2 43.7 34.4

## 4 MSFT Microso… 0.0306 Inform… 9.22e⁷ 2008-01-07 44.3 44.6 43.9 34.6

## 5 MSFT Microso… 0.0306 Inform… 9.22e⁷ 2008-01-08 44.5 44.5 42.8 33.5

## 6 MSFT Microso… 0.0306 Inform… 9.22e⁷ 2008-01-09 42.8 44.3 42.8 34.4

## # ... with 2 more variables: volume , adjusted

I will build the portfolio optimization on the adjusted prices. We also need to select close prices and shares\_held to calculate the market capitalization. Additionally, I will optimize the portfolio on monthly returns and the data must be aggregated.

## == Treat Data == ##

prices = prices %>%

select(symbol, date, adjusted, close ,shares\_held) %>%

group\_by(year(date), month(date), symbol) %>%

summarize( adjusted = tail(adjusted, 1),

close = tail(close, 1),

mktcap = tail(close \* shares\_held, 1)) %>%

mutate(date = make\_date(`year(date)`, `month(date)`))

Next we need to transform the data into time-series panel with the function acast and calculate log-returns from the adjusted prices. Note that the characteristics in tare used to build a portfolio for t+1. Therefore, the characteristics must be lagged one month from the returns. I also removed stocks with NAs. The PPP works fine in unbalanced panels but I will use them balanced to keep it simple. Just keep in mind that if you use it in unbalanced panels N_tand the base portfolio \bar{w}_{i,t}must be adjusted.

## == Reshape variables to matrices == ##

# = dropped first observatio in the mktcap because we = #

# = loose one observatio to calculate the returns = #

mktcap=acast(prices,date~symbol,value.var = "mktcap")[-1,]

prices=acast(prices,date~symbol,value.var = "adjusted")

## == Calculate log returns == ##

ret=diff(log(prices))

## == Remove Stocks with NAs == ##

drop = which( is.na(colSums(ret)) | is.na(colSums(mktcap)) )

ret = ret[, - drop]

mktcap = mktcap[ , - drop]

## == Remove last observation of mktcap because it has == ##

## == to be lagged one month from the returns == ##

mktcap = mktcap[- nrow(mktcap), ]

The next step is to generate a variable that is the cumulative return of the last 12 months that will be used as a momentum characteristic. We loose some observations in this procedure and the rest of the data must be adjusted. The characteristics must also be standardized cross-sectionally to have mean zero and variance one. The next chunk also defines the risk aversion coefficient as 5. Feel free to play with the risk aversion. If you increase it to much the portfolio will be the same as the equal weighted and if you decrease the risk aversion you are likely to have portfolios with more variance and leverage.

## == Generate 12 month cumulative returns == ##

## == I used sub because returns are in log == ##

m12 = matrix(NA, nrow(ret), ncol(ret))

for(i in 13:nrow(m12)){

m12[i, ] = apply(ret[(i - 12):(i - 1), ],2 , sum )

}

## == Remove last 12 observations because we == ##

## == don't the first 12 m12. == ##

ret = ret[-c(1:12), ]

m12 = m12[-c(1:12), ]

# = only 11 in the mktcap because we already = #

# = removed one = #

mktcap = mktcap[-c(1:11), ]

## == This is the weigth of each stock in a == #

## == equal weighted portfolio == #

nt = wb = 1/ncol(ret)

## = Risk aversion == ##

rr = 5

## == Some plots == ##

par(mfrow = c(2, 2))

matplot(ret[ ,1:10], type = "l", ylab = "monthly returns")

matplot(m12[ ,1:10], type = "l", ylab = "12 month returns")

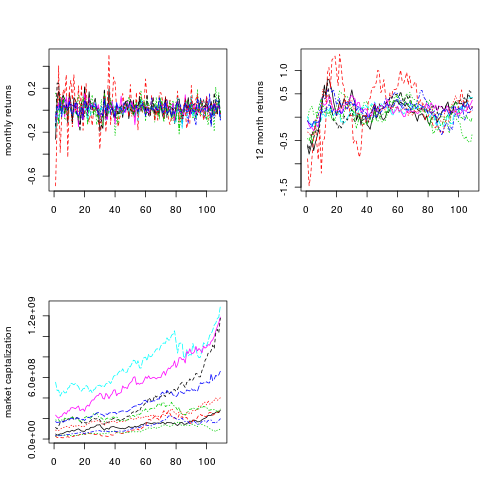
matplot(mktcap[ ,1:10], type = "l", ylab = "market captalization")

## == Characteristics must be standardized == ##

## == on the crossection == ##

m12=t(apply(m12,1,scale))

mktcap=t(apply(mktcap,1,scale))



**Optimizing**

Now let’s define the optimization function and use it to find our portfolio. I will use the function in 60 expanding windows to simulate a portfolio that is re-estimated and updated every month. The interpretation is that in the end of every month we optimize with all the data we have until then and build our portfolio for the next month.

## == Portfolio Optimization Function == ##

pps = function(x, wb, nt, ret, m12, mktcap, rr){

wi = wb + nt \* (x[1]\*m12 + x[2]\*mktcap)

wret = rowSums(wi\*ret)

ut = ((1 + wret)^(1 - rr))/(1 - rr)

u = -mean(ut)

return(u)

}

## == Loop the function in 60 expanding windows == ##

res\_save = matrix(NA, 60, 2)

weights = matrix(NA, 60, ncol(ret))

for(i in 1:60){

opt = optim(c(0, 0), pps, wb = wb, nt = nt, ret = ret[1:(48 + i), ],

m12 = m12[1:(48 + i), ], mktcap = mktcap[1:(48 + i), ],

rr = rr, method = "BFGS")

res = opt$par

w = wb + nt\*(res[1]\*m12[i + 49, ] + res[2]\*mktcap[i + 49, ])

weights[i, ] = w

res\_save[i, ] = res

}

The code below calculates the out-of-sample cumulative returns of our optimized portfolio and the equal weighted (EW) portfolio to use as benchmark. I also calculated the returns of a look-ahead portfolio with the best 100 stocks in the S&P500 during our period of analysis. The code ends with a plot comparing the cumulative returns of the tree portfolios.

## == Calculate out-of-sample returns == ##

ret\_fit = rowSums(weights\*tail(ret, 60))

ret\_EW = rowSums(nt\*tail(ret, 60))

## == Accumulate out-of-sample returns == ##

acc\_fit = cumsum(ret\_fit)

acc\_EW = cumsum(ret\_EW)

## == Calculate top 100 portfolio == ##

acc\_ret = apply(ret[-c(1:49), ], 2, cumsum)

top100 = order(acc\_ret[nrow(acc\_ret), ] ,decreasing = TRUE)[1:100]

acc\_top100 = rowMeans(acc\_ret[ ,top100])

df = data.frame(date = tail(as.Date(rownames(ret)), 60),

opt = acc\_fit,EW = acc\_EW,top100 = acc\_top100)

dfm = melt(df, id.vars = "date")

colnames(dfm) = c("date", "portfolio", "return")

ggplot(dfm) + geom\_line(aes(x = date, y = return, color = portfolio))



As you can see, our optimal portfolio is far superior than the EW portfolio and it beats the top100 portfolio most of the time. The next plot shows the parameters of the PPP in the 60 expanding windows. In general, our policy is to buy more of winners (high momentum) and buy less of bigger companies.

df = data.frame(date = tail(as.Date(rownames(ret)), 60),

res\_save)

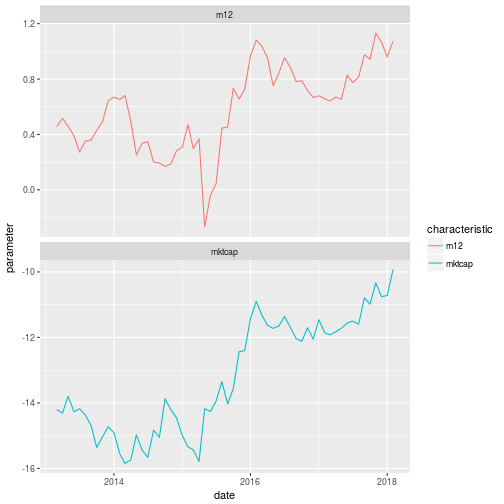
colnames(df)=c("date","m12","mktcap")

dfm = melt(df, id.vars = "date")

colnames(dfm) = c("date", "characteristic", "parameter")

ggplot(dfm) + geom\_line(aes(x = date, y = parameter, color = characteristic)) +

facet\_wrap(~ characteristic, scales = "free\_y", ncol = 1)

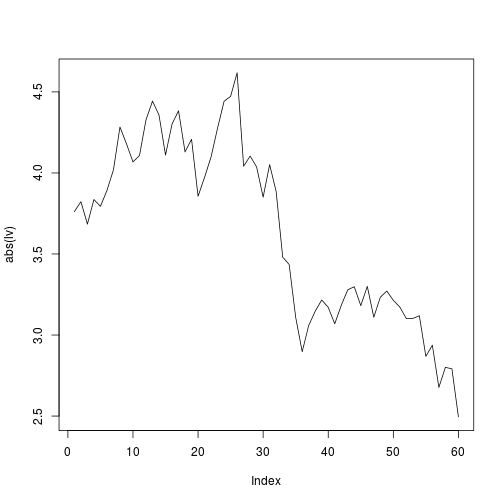


**Important Considerations**

The question now is: does the optimal portfolio I created generate realistic results??? The answer is NO! The results were very impressive, the PPP was able to beat a look-ahead portfolio, which is very unlikely to happen. The cause of this super performance is the lack of restrictions on short positions. Our portfolio assumes that we have infinite credit at no cost to borrow money and invest in stocks. Look at the plot below:

lv = apply(weights, 1, function(x) sum(x[x<0]))

plot(abs(lv),type="l")



This plot shows the leverage we had in each month in the out-of-sample period, and it is a lot. In some months we had 4.5 times our wealth of borrowed money. Fortunately, it is very easy to put some restrictions in the PPP. All we have to do is add a penalty in the objective function that penalizes short positions when they are bigger than some target we choose. The function below penalizes leverage values bigger than two. The code chunk ends with the leverage comparison plot, which shows that the penalization kept the leverage very close to our target of two.

## == Model with leverage penalty == ##

pps1 = function(x, wb, nt, ret, m12, mktcap, rr){

wi = wb + nt \* (x[1]\*m12 + x[2]\*mktcap)

wret = rowSums(wi\*ret)

ut = ((1 + wret)^(1 - rr))/(1 - rr)

u = -mean(ut)

lv = abs(apply(wi, 1, function(x) sum(x[x<0])))

pen=abs(sum((lv>2)\*lv))

return(u+pen)

}

## == Loop the function in 60 expanding windows == ##

res\_save1 = matrix(NA, 60, 2)

weights1 = matrix(NA, 60, ncol(ret))

for(i in 1:60){

opt = optim(c(0, 0), pps1, wb = wb, nt = nt, ret = ret[1:(48 + i), ],

m12 = m12[1:(48 + i), ], mktcap = mktcap[1:(48 + i), ],

rr = rr, method = "BFGS")

res = opt$par

w = wb + nt\*(res[1]\*m12[i + 49, ] + res[2]\*mktcap[i + 49, ])

weights1[i, ] = w

res\_save1[i, ] = res

}

## == Calculate out-of-sample returns == ##

ret\_fit1 = rowSums(weights1\*tail(ret, 60))

## == Accumulate out-of-sample returns == ##

acc\_fit1 = cumsum(ret\_fit1)

## == Leverage analysis == ##

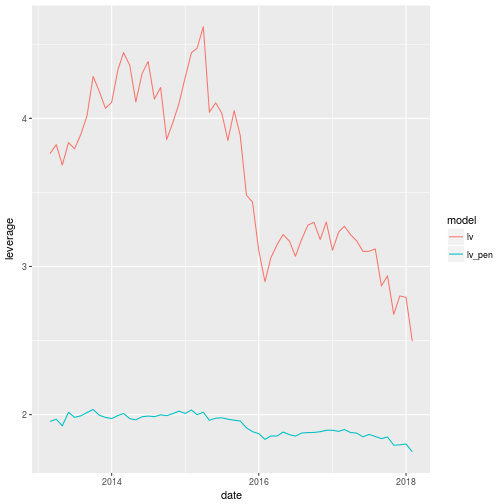
lv1 = apply(weights1, 1, function(x) sum(x[x<0]))

df=data.frame(date = tail(as.Date(rownames(ret)), 60),lv = abs(lv), lv\_pen = abs(lv1))

dfm = melt(df, id.vars = "date")

colnames(dfm) = c("date", "model", "leverage")

ggplot(dfm) + geom\_line(aes(x = date, y = leverage, color = model))



However, the inclusion of short selling restrictions comes at a cost. Look at the plot below! The returns in the penalizes portfolio are smaller, which is natural because we now have a more restricted model.

df = data.frame(date = tail(as.Date(rownames(ret)), 60),

opt = acc\_fit, opt\_pen=acc\_fit1, EW = acc\_EW,top100 = acc\_top100)

dfm = melt(df, id.vars = "date")

colnames(dfm) = c("date", "portfolio", "return")

ggplot(dfm) + geom\_line(aes(x = date, y = return, color = portfolio))

