The Magic in R

Stefan Glogger

August 2017

Overview

Please look at the titles to get an overview of what is done when. You can also refer to the introducing sentences of each chapter. Furthermore an overview is provided as a chart.

Data Import

We import the sentiment data. We also import the prices of each index over the relevant time frame.

Sentix

Read the raw sentiment data and save it in the list *sentixRaw* with each list element containing the results of the survey for the different indices. As the number of rows (dates of observation) in data differ, we extract the unique dates (*datesSentix*) and reduce the data to it. We also determine *min(datesSentix)* and *max(datesSentix)*, which we use lateron to get the stock data.

```
# install.packages("openxlsx")
library(openxlsx)
```

```
folderSentix <- (file.path(getwd(), "Data", "Sentix"))

sheets <- c("DAX","DAXm","TEC","TECm","ESX50","ESX50m","SP5","SP5m","NASDAQ","NASDAQm","NIKKEI","NIKKEIm
relevant_rows <- c("Datum","P+","Pn","P-","I+","In","I-","G+","Gn","G-")
```

```
sentixRaw <- list()

for(i in sheets){
  sentixRaw[[i]] <- read.xlsx(file.path(folderSentix, "sentix_anzahlen_bis_02092016xlsx.xlsx"),sheet=i,d
  sentixRaw[[i]] <- sentixRaw[[i]][,relevant_rows]
  sentixRaw[[i]] <- sentixRaw[[i]][order(sentixRaw[[i]][,1]),]
}
```

```
unlist(lapply(sentixRaw, nrow))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ
##     803     803     803     803     803     803     803     803     803
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm   TBOND  TBONDm
##     803     803     803     802     802     802     802
```

```
datesSentix <- unique(sentixRaw[[1]]$Datum)
for(i in names(sentixRaw)[2:length(sentixRaw)]){
  if(!(setequal(datesSentix, sentixRaw[[i]]$Datum)))
    stop("Sentix Data of different indices have not same dates. Handle manually.")
}

for(i in names(sentixRaw)){
  sentixRaw[[i]] <- unique(sentixRaw[[i]])
}
unlist(lapply(sentixRaw, nrow))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ
##     802     802     802     802     802     802     802     802     802
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm   TBOND  TBONDm
##     802     802     802     802     802     802     802
```

```
rm(folderSentix, sheets, relevant_rows, i)
detach("package:openxlsx", unload = T)
```

Stocks

We take data mainly from Yahoo Finance. We take closing course from *min(datesSentix)* to *max(datesSentix)* for several indexes and store in the data frame *stocks* the closing stock price at each date of the sentiment data (*datesSentix*).

We take the following as sources of the data:

- DAX *^GDAXI*
- TEC *^TECDAX*
- ESX50 *^STOXX50E*
- SP500 *^GSPC*
- NASDAQ *^NDX*
- NIKKEI *^N225*
- BUND from Sebastian: Den Bund-Future habe ich bei onvista in 5-Jahresst?cken geladen und zusammengebaut. Dezimaltrennzeichen umgestellt im .csv —- not from yahoo, manually from bundesbank *BBK01.WT0557*
- TBOND from Sebastian: Beim T-Bond ist es die 10 Year Treasury Note, auf welche das TBOND Sentiment abzielt. Diese habe ich bei FRED geladen: https://fred.stlouisfed.org/series/DGS10

```
# install.packages("quantmod")
library(quantmod)
```

```
## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: TTR

## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
# ?getSymbols
```

```
stocks <- data.frame(Datum = datesSentix)

# DAX
dax <- new.env()
getSymbols("^GDAXI", env = dax, src = "yahoo", from = min(datesSentix), to = max(datesSentix))
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.

##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
```

```
## options("getSymbols.yahoo.warning"=FALSE).

## Warning: ^GDAXI contains missing values. Some functions will not work if
## objects contain missing values in the middle of the series. Consider using
## na.omit(), na.approx(), na.fill(), etc to remove or replace them.

## [1] "GDAXI"
```

```
DAX <- data.frame(dax$GDAXI[datesSentix,"GDAXI.Close"])
colnames(DAX) <- "Close" # somehow the column name cannot be given directly
DAX$Datum <- as.Date(row.names(DAX))

stocks$DAX <- merge(stocks, DAX, by = "Datum", all.x = T)$Close


# TEC
tec <- new.env()
getSymbols("^TECDAX", env = tec, src = "yahoo", from = min(datesSentix), to = max(datesSentix))
```

```
## Warning: ^TECDAX contains missing values. Some functions will not work if
## objects contain missing values in the middle of the series. Consider using
## na.omit(), na.approx(), na.fill(), etc to remove or replace them.

## [1] "TECDAX"
```

```
TEC <- data.frame(tec$TECDAX[datesSentix, "TECDAX.Close"])
colnames(TEC) <- "Close"
TEC$Datum <- as.Date(row.names(TEC))

stocks$TEC <- merge(stocks, TEC, by = "Datum", all.x = T)$Close


# ESX50
esx50 <- new.env()
getSymbols("^STOXX50E", env = esx50, src = "yahoo", from = min(datesSentix), to = max(datesSentix))
```

```
## Warning: ^STOXX50E contains missing values. Some functions will not work if
## objects contain missing values in the middle of the series. Consider using
## na.omit(), na.approx(), na.fill(), etc to remove or replace them.

## [1] "STOXX50E"
```

```
ESX50 <- data.frame(esx50$STOXX50E[datesSentix,"STOXX50E.Close"])
colnames(ESX50) <- "Close"
ESX50$Datum <- as.Date(row.names(ESX50))

stocks$ESX50 <- merge(stocks, ESX50, by = "Datum", all.x = T)$Close


# SP500
sp500 <- new.env()
getSymbols("^GSPC", env = sp500, src = "yahoo", from = min(datesSentix), to = max(datesSentix))
```

```
## [1] "GSPC"
```

```
SP500 <- data.frame(sp500$GSPC[datesSentix,"GSPC.Close"])
colnames(SP500) <- "Close"
SP500$Datum <- as.Date(row.names(SP500))
```

```
# sum(is.na(SP500$Close))

stocks$SP5 <- merge(stocks, SP500, by = "Datum", all.x = T)$Close


# NASDAQ
nasdaq <- new.env()
getSymbols("^NDX", env = nasdaq, src = "yahoo", from = min(datesSentix), to = max(datesSentix))
```

```
## [1] "NDX"
```

```
NASDAQ <- data.frame(nasdaq$NDX[datesSentix,"NDX.Close"])
# sum(is.na(NASDAQ[,"NDX.Close"]))
colnames(NASDAQ) <- "Close"
NASDAQ$Datum <- as.Date(row.names(NASDAQ))

stocks$NASDAQ <- merge(stocks, NASDAQ, by = "Datum", all.x = T)$Close


# NIKKEI
nikkei <- new.env()
getSymbols("^N225", env = nikkei, src = "yahoo", from = min(datesSentix), to = max(datesSentix))
```

```
## Warning: ^N225 contains missing values. Some functions will not work if
## objects contain missing values in the middle of the series. Consider using
## na.omit(), na.approx(), na.fill(), etc to remove or replace them.
```

```
## [1] "N225"
```

```
NIKKEI <- data.frame(nikkei$N225[datesSentix,"N225.Close"])
colnames(NIKKEI) <- "Close"
NIKKEI$Datum <- as.Date(row.names(NIKKEI))

stocks$NIKKEI <- merge(stocks, NIKKEI, by = "Datum", all.x = T)$Close
```

Bund

```
BUND <- read.csv(file.path(getwd(), "Data", "Bundfuture", "Bundfuture2001-2017.csv"), sep = ";")
BUND[,1] <- as.Date(BUND[,1], format = "%d.%m.%Y")
BUND <- BUND[BUND[,1] %in% datesSentix,]
BUND <- as.data.frame(BUND)

stocks$BUND <- merge(stocks, BUND, by = "Datum", all.x = T)$Schluss
```

Treasury bond

```
TBOND <- read.csv(file.path(getwd(), "Data", "10 year T-Notes", "DGS10.csv"), sep = ",")
TBOND[,1] <- as.Date(TBOND[,1], format = "%Y-%m-%d")
TBOND[,2] <- as.numeric(as.character(TBOND[,2])) # was a factor first and factors are stored via index
```

```
## Warning: NAs durch Umwandlung erzeugt
```

```
colnames(TBOND) <- c("Datum", "DGS10")
TBOND <- TBOND[TBOND[,1] %in% datesSentix,]
TBOND <- as.data.frame(TBOND)

stocks$TBOND <- merge(stocks, TBOND, by = "Datum", all.x = T)$DGS10
```

```r
rm(BUND, DAX, ESX50, NASDAQ, NIKKEI, SP500, TBOND, TEC,
   dax, esx50, nasdaq, nikkei, sp500, tec)
detach("package:quantmod", unload = T)
```

Data Preparation

We look at how many people participated in the survey on average and remove TBOND.

We look at the number of dates on which not all stocks report prices and remove those to end up with the dates on which all data is available *datesAll*.

Sentix - number of participants in survey

NOTE: maybe also delete the "G" columns in the sentix data lateron (but it might produce quite interesting results)

```
cols <- 8:10
colnames(sentixRaw[[1]])[cols]
```

```
## [1] "G+" "Gn" "G-"
```

```
unlist(lapply(sentixRaw, function(x) {round(mean(rowSums(x[cols])), 0)}))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ
##     701     698     677     674     696     692     694     690     683
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm   TBOND  TBONDm
##     680     647     643     628     625     160     160
```

```
rm(cols)
```

We remove TBOND, as just very few people voted for it over time in comparison to the other indices.

```
sentixRaw[["TBOND"]] <- NULL
sentixRaw[["TBONDm"]] <- NULL
stocks <- stocks[,-which(colnames(stocks)=="TBOND")]
```

```
unlist(lapply(sentixRaw, function(x) {sum(is.na.data.frame(x))}))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ
##       0       0       0       0       0       0       0       0       0
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm
##       0       0       0       0       0
```

Stocks - na's

There might be dates missing (we just have to look at stocks as we found the *datesSentix* as those dates, for which all sentiment is there).

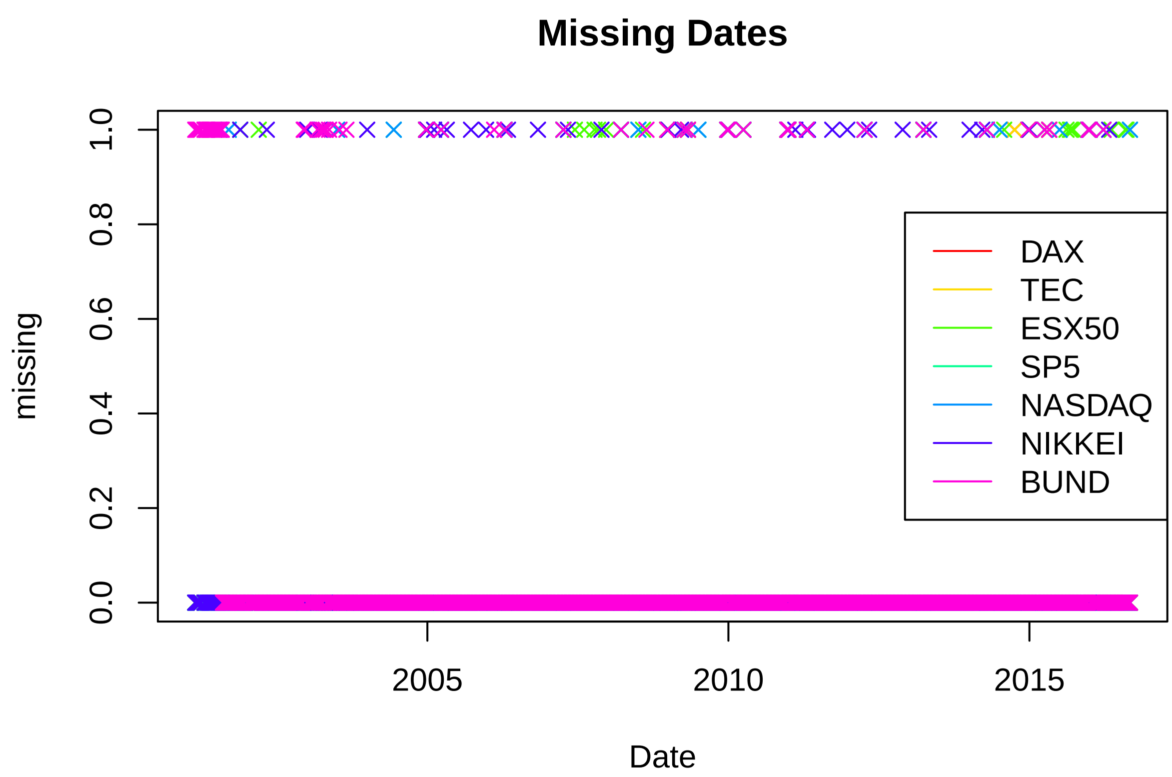
```
colSums(is.na.data.frame(stocks))
```

```
##  Datum    DAX    TEC  ESX50    SP5 NASDAQ NIKKEI   BUND
##      0     25     22     41     26     26     32     56
```

Visualize the missing dates (missing date = 1, not missing date = 0 on y-axis).

```
cols <- rainbow(ncol(stocks)-1)

plot(stocks[,1], is.na(stocks[,2]), main = "Missing Dates", ylab = "missing", xlab = "Date", col = cols
for(i in 2:(ncol(stocks)-1)){
    par(new=T)
    plot(stocks[,1], is.na(stocks[,i+1]), col = cols[i], axes = F, xlab = "", ylab = "", pch = 4)
```

```
}
legend("right", legend = colnames(stocks)[2:ncol(stocks)], col = cols, lty = 1)
```




```
pdf(file.path(getwd(), "Plot", "missingDates.pdf"), width = 10, height = 4)
cols <- rainbow(ncol(stocks)-1)
plot(stocks[,1], is.na(stocks[,2]), main = "Missing Dates", ylab = "missing", xlab = "Date", col = cols
for(i in 2:(ncol(stocks)-1)){
    par(new=T)
    plot(stocks[,1], is.na(stocks[,i+1]), col = cols[i], axes = F, xlab = "", ylab = "", pch = 4)
}
legend("right", legend = colnames(stocks)[2:ncol(stocks)], col = cols, lty = 1)
dev.off()
```

```
## pdf
##   2
```

```
rm(cols, i)
```

Determine, how many dates do have all data available.

```
nrow(stocks)
```

```
## [1] 802
```

```
nrow(stocks[complete.cases(stocks),])
```

```
## [1] 695
```

```r
nrow(stocks) - nrow(stocks[complete.cases(stocks),])
```

```
## [1] 107
```

```r
(nrow(stocks) - nrow(stocks[complete.cases(stocks),]))/nrow(stocks)
```

```
## [1] 0.1334165
```

So we would delete 13.3416459 % of the data.

delete

We delete dates with missing values.

```r
stocks <- stocks[complete.cases(stocks),]

datesAll <- stocks[,1]
rm(datesSentix)

sentixRaw <- lapply(sentixRaw, function(x) {x[(x[,1] %in% datesAll),]})

unlist(lapply(sentixRaw, nrow))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ 
##     695     695     695     695     695     695     695     695     695 
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm 
##     695     695     695     695     695
```

other approach (not implemented)

One way of approaching this might be via linear regression of the stock data when no stock price is available. but this assumes a linear relationship and might cause trouble.

Data Derivations

We calculate dispersion and herfindah for the sentix data.

Sentix

Dispersion

We measure dispersion of the results of the survey (at each date) as its variance.

Fix one date. Let X_i be the respond of participant i to the future state of the stock with $X_i = 1$ representing, he has positive opinion, $X_i = 0$ neutral, $X_i = -1$ negative.

Then we calculate the dispersion of X as:

$$\text{disp}(X) = \text{Var}(X), \text{where} X = (X_1, ...X_n)$$

In alignment to Dominik's code, we perform the calculation for each index, each group of persons (private, institutional and all), and both time periods (1 month, 6 month).

We produce a list named *sDisp*. Each list element (e.g. P1, P6, I1, ...) contains a data frame with the dispersion for each index (column) at each date (row).

```
sDisp <- list()

colnames(sentixRaw[[1]])
```

```
##  [1] "Datum" "P+"    "Pn"    "P-"    "I+"    "In"    "I-"    "G+"
##  [9] "Gn"    "G-"
```

```
groupP <- c("P+", "Pn", "P-")
groupI <- c("I+", "In", "I-")
groupG <- c("G+", "Gn", "G-")
sDispColumn <- function(dat, group){
  res <- numeric(nrow(dat))
  for(i in 1:length(res)){
    res[i] <- var(c(rep(1, dat[i, group[1]]), rep(0, dat[i, group[2]]), rep(-1, dat[i, group[3]])))
  }
  return(res)
}

names(sentixRaw)
```

```
##  [1] "DAX"     "DAXm"    "TEC"     "TECm"    "ESX50"   "ESX50m"  "SP5"
##  [8] "SP5m"    "NASDAQ"  "NASDAQm" "NIKKEI"  "NIKKEIm" "BUND"    "BUNDm"
```

```
(period1 <- names(sentixRaw)[2*((0:(length(sentixRaw)/2-1)))+1])
```

```
## [1] "DAX"    "TEC"    "ESX50"  "SP5"    "NASDAQ" "NIKKEI" "BUND"
```

```
(period6 <- names(sentixRaw)[2*((0:(length(sentixRaw)/2-1)))+2])
```

```
## [1] "DAXm"    "TECm"    "ESX50m"  "SP5m"    "NASDAQm" "NIKKEIm" "BUNDm"
```

```
sDispDataFrame <- function(period, group){
  res <- data.frame(Datum = datesAll)

  res$DAX <- sDispColumn(sentixRaw[[period[1]]], group)
```

```
  res$TEC <- sDispColumn(sentixRaw[[period[2]]], group)
  res$ESX50 <- sDispColumn(sentixRaw[[period[3]]], group)
  res$SP5 <- sDispColumn(sentixRaw[[period[4]]], group)
  res$NASDAQ <- sDispColumn(sentixRaw[[period[5]]], group)
  res$NIKKEI <- sDispColumn(sentixRaw[[period[6]]], group)
  res$BUND <- sDispColumn(sentixRaw[[period[7]]], group)

  return(res)
}

sDisp[["P1"]] <- sDispDataFrame(period1, groupP)
sDisp[["P6"]] <- sDispDataFrame(period6, groupP)
sDisp[["I1"]] <- sDispDataFrame(period1, groupI)
sDisp[["I6"]] <- sDispDataFrame(period6, groupI)
sDisp[["G1"]] <- sDispDataFrame(period1, groupG)
sDisp[["G6"]] <- sDispDataFrame(period6, groupG)


# we get a problem as the helping formulas are hard coded
if((ncol(sDisp[[1]])-1) != length(period1))
  stop("Fatal error. Check 'sDispDataFrame'. number of Indices changed")

rm(groupP, groupI, groupG, sDispColumn,
   period1, period6, sDispDataFrame)
```

herfindah

We compute a weighted negative Herfindahl Index, which is a measure of dispersion as given in https://www.federalreserve.gov/pubs/feds/2014/201435/201435pap.pdf. Negative value lets higher values indicate greater dispersion.

At each fixed date, the weighted negative Herfindahl Index is computed by:

$$\text{herf}(X) = -\left[\left(\frac{|\{X_i : X_i = 1\}|}{|\{X_1, ..., X_n\}|}\right)^2 + 2\left(\frac{|\{X_i : X_i = 0\}|}{|\{X_1, ..., X_n\}|}\right)^2 + \left(\frac{|\{X_i : X_i = -1\}|}{|\{X_1, ..., X_n\}|}\right)^2\right]$$

Code in analogy to Dominik's.

We produce a list named *sHerf*. Each list element (e.g. P1, P6, I1, ...) contains a data frame with the dispersion for each index (column) at each date (row).

```
sHerf <- list()

colnames(sentixRaw[[1]])
```

```
##  [1] "Datum" "P+"    "Pn"    "P-"    "I+"    "In"    "I-"    "G+"
##  [9] "Gn"    "G-"
```

```
groupP <- c("P+", "Pn", "P-")
groupI <- c("I+", "In", "I-")
groupG <- c("G+", "Gn", "G-")
sHerfColumn <- function(dat, group){
  res <- numeric(nrow(dat))
  for(i in 1:length(res)){
```

```
    s <- sum(dat[i, group])
    res[i] <- -1*( (dat[i, group[1]]/s)^2 + 2*(dat[i, group[2]]/s)^2 + (dat[i, group[3]]/s)^2 )
  }
  return(res)
}

names(sentixRaw)
```

```
##  [1] "DAX"     "DAXm"    "TEC"     "TECm"    "ESX50"   "ESX50m"  "SP5"
##  [8] "SP5m"    "NASDAQ"  "NASDAQm" "NIKKEI"  "NIKKEIm" "BUND"    "BUNDm"
```

```
(period1 <- names(sentixRaw)[2*((0:(length(sentixRaw)/2-1)))+1])
```

```
## [1] "DAX"    "TEC"    "ESX50"  "SP5"    "NASDAQ" "NIKKEI" "BUND"
```

```
(period6 <- names(sentixRaw)[2*((0:(length(sentixRaw)/2-1)))+2])
```

```
## [1] "DAXm"    "TECm"    "ESX50m"  "SP5m"    "NASDAQm" "NIKKEIm" "BUNDm"
```

```
sHerfDataFrame <- function(period, group){
  res <- data.frame(Datum = datesAll)

  res$DAX <- sHerfColumn(sentixRaw[[period[1]]], group)
  res$TEC <- sHerfColumn(sentixRaw[[period[2]]], group)
  res$ESX50 <- sHerfColumn(sentixRaw[[period[3]]], group)
  res$SP5 <- sHerfColumn(sentixRaw[[period[4]]], group)
  res$NASDAQ <- sHerfColumn(sentixRaw[[period[5]]], group)
  res$NIKKEI <- sHerfColumn(sentixRaw[[period[6]]], group)
  res$BUND <- sHerfColumn(sentixRaw[[period[7]]], group)

  return(res)
}

sHerf[["P1"]] <- sHerfDataFrame(period1, groupP)
sHerf[["P6"]] <- sHerfDataFrame(period6, groupP)
sHerf[["I1"]] <- sHerfDataFrame(period1, groupI)
sHerf[["I6"]] <- sHerfDataFrame(period6, groupI)
sHerf[["G1"]] <- sHerfDataFrame(period1, groupG)
sHerf[["G6"]] <- sHerfDataFrame(period6, groupG)


# we get a problem as the helping formulas are hard coded
if((ncol(sHerf[[1]])-1) != length(period1))
  stop("Fatal error. Check 'sHerfDataFrame'. number of Indices changed")

rm(groupP, groupI, groupG, sHerfColumn,
   period1, period6, sHerfDataFrame)
```

Stocks

We calculate discrete returns for each date and each stock.

returns

Discrete returns. Be aware that we "loose" the first date now, as we have no idea of the return on day one. Therefore we might also exclude the first date for the other (sentix) variables. We will go on with carefully matching the dates to always consider information ot the actual day.

```
ret <- as.matrix(stocks[2:nrow(stocks),2:ncol(stocks)]/stocks[1:(nrow(stocks)-1),2:ncol(stocks)] - 1)
rownames(ret) <- stocks[2:nrow(stocks), 1]

mu <- colMeans(ret)
C <- cov(ret)
```

```
# sentixRaw <- lapply(sentixRaw, function(x) {x <- x[2:nrow(x), ]})
# sDisp <- lapply(sDisp, function(x) {x <- x[2:nrow(x), ]})
# sHerf <- lapply(sHerf, function(x) {x <- x[2:nrow(x), ]})
#
# stocks <- stocks[2:nrow(stocks), ]
# datesAll <- datesAll[2:nrow(datesAll)]
```

time window

bull and bear

Fix length of time window (l). Calculate return for all stocks (*retWindow*) for all possible time windows (1, l+1, l+2, ..., T). Equal weights for all returns (of the different indices). Calculate (arithmetic) average of all returns in each possible time window (*retTotal*). Choose the one with lowest (*datesEvalBear*) and highest (*datesEvalBull*).

$$\text{retWindow}_{\text{stock}} = \prod_{k=1}^{l}(1 + \text{ret}_{\text{stock}}(k)) - 1$$

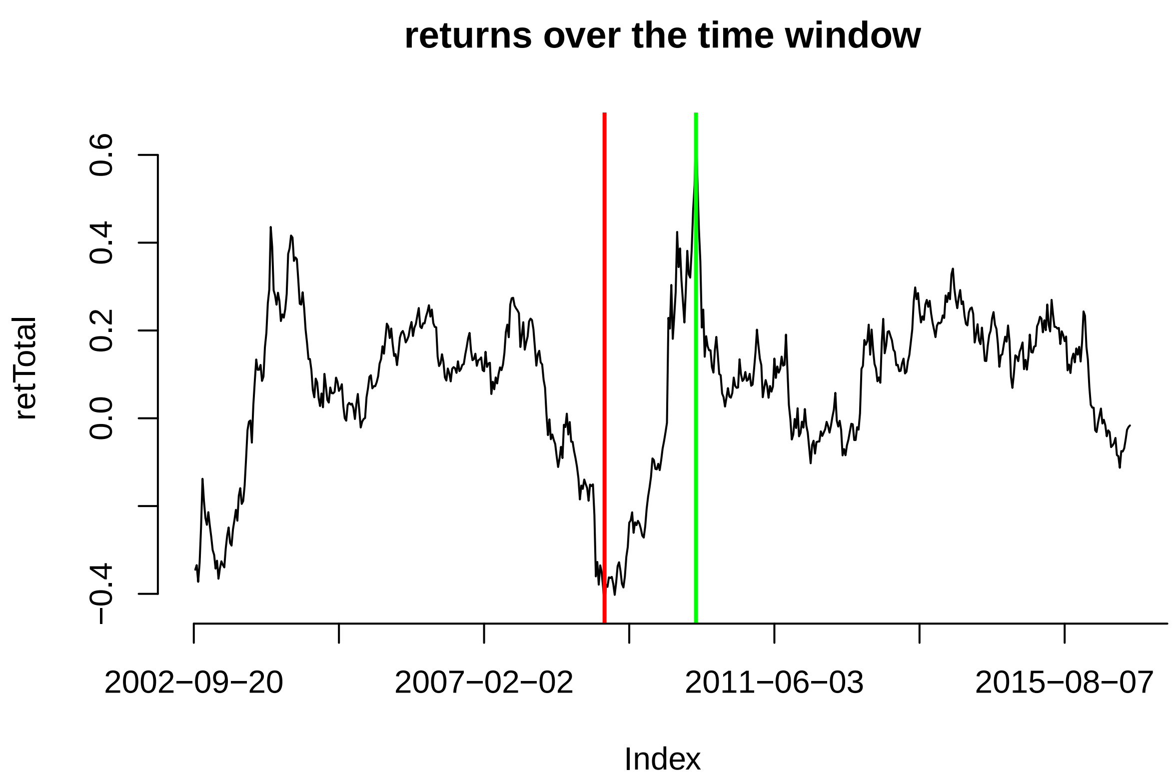
As we calculate with closing prices, we assume that the return is actually of that day (or better spoken of that week). We investment at the very beginning to the opening price, which should be rathly the closing price of the day (week) before).

```
l <- 50

retWindow <- matrix(0, nrow = nrow(ret)-l+1, ncol = ncol(ret))
rownames(retWindow) <- rownames(ret)[l:nrow(ret)]
class(rownames(retWindow)) <- "Date"

for(i in 1:nrow(retWindow)){
    retWindow[i,] <- apply(ret[i:(i+l-1),]+1, 2, function(x) prod(x)-1) # 2 -> columnwise
}

retTotal <- numeric(nrow(retWindow))
retTotal <- apply(retWindow, 1, mean) # 1 -> rowwise
names(retTotal) <- rownames(retWindow)

iMin <- which(retTotal==min(retTotal))
iMax <- which(retTotal==max(retTotal))

# dates of which the returns have been calculated
datesEvalBear <- rownames(ret)[(iMin):(iMin+l-1)]
```

```
datesEvalBull <- rownames(ret)[(iMax):(iMax+l-1)]
class(datesEvalBear) <- "Date"
class(datesEvalBull) <- "Date"
```

additional visualization of the resturns over each time window

```
plot(retTotal, type = "l", axes = FALSE, main = "returns over the time window")
abline(v = iMin, col = "red", lwd = 2)
abline(v = iMax, col = "green", lwd = 2)
axis(1, pretty(1:length(retTotal)), names(retTotal)[pretty(1:length(retTotal))+1])
axis(2)
```



last data

We also look at the most actual data.

```
datesEvalLast <- rownames(ret)[(nrow(ret)-l+1):nrow(ret)]
class(datesEvalLast) <- "Date"
```

used later for storing results. trick *deparse(substitute())* to get an error when a window is deleted.

test period

We furthermore define a test period to determine the optimal weights of the goal function. Therefore we choose the first time period before the actual evaluating time periods. From *startDateTest* up to *startEvalTime* minus *timeBefore*, to leave some space before the actual analysis starts.

```
startDateTest <- 50
timeBefore <- 50
( startEvalTime <- which(datesAll == min(c(min(datesEvalBear), min(datesEvalBull), min(datesEvalLast)))
```

```
## [1] 284
```

```
datesTest <- rownames(ret)[startDateTest:(startEvalTime-timeBefore)]
class(datesTest) <- "Date"
length(datesTest)
```

```
## [1] 185
```

cleanup

```
datesEvalNames <- c(deparse(substitute(datesEvalBear)), deparse(substitute(datesEvalBull)), deparse(subs
```

```
datesTestNames <- c(deparse(substitute(datesTest)))
```

remove variables

```
rm(l, i)
rm(retWindow, retTotal)
rm(iMin, iMax, startDateTest, startEvalTime, timeBefore)
```

Data Visualization

We visualize the data (stocks and sentix). For consistency, we first specify general parameters on how to display each index and the time periods.

Function

Put everything in one function to plot.

```
colsEvalDates <- c("red", "green", "orange")
names(colsEvalDates) <- datesEvalNames
```

Rectangle for Date periods: store as function to keep structure similar to above (and store at same Place in environment)

```
plotData <- function(x, title = "Indices"){

    # lines with data
    geomLineDataDAX <- function(x){
        parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = DAX, colour = \"DAX\"))"))
    }
    geomLineDataTEC <- function(x){
        parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = TEC, colour = \"TEC\"))"))
    }
    geomLineDataESX50 <- function(x){
        parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = ESX50, colour = \"ESX50\"))"))
    }
    geomLineDataSP5 <- function(x){
        parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = SP5, colour = \"SP5\"))"))
    }
    geomLineDataNASDAQ <- function(x){
        parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = NASDAQ, colour = \"NASDAQ\"))
    }
    geomLineDataNIKKEI <- function(x){
        parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = NIKKEI, colour = \"NIKKEI\"))
    }
    geomLineDataBUND <- function(x){
        parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = BUND, colour = \"BUND\"))"))
    }

    # rectangle for date period
    geomRectDateBear <- function(){
        parse(text = "geom_rect(aes(xmin = min(datesEvalBear), xmax = max(datesEvalBear), ymin = -Inf, y

    geomRectDateBull <- function(){
        parse(text = "geom_rect(aes(xmin = min(datesEvalBull), xmax = max(datesEvalBull), ymin = -Inf, y

    geomRectDateLast <- function(){
        parse(text = "geom_rect(aes(xmin = min(datesEvalLast), xmax = max(datesEvalLast), ymin = -Inf, y

    geomRectDateTest <- function(){
        parse(text = "geom_rect(aes(xmin = min(datesTest), xmax = max(datesTest), ymin = -Inf, ymax = In
```

```
    ggplot() +
        eval(geomLineDataDAX(x)) +
        eval(geomLineDataTEC(x)) +
        eval(geomLineDataESX50(x)) +
        eval(geomLineDataNASDAQ(x)) +
        eval(geomLineDataNIKKEI(x)) +
        eval(geomLineDataBUND(x)) +
        eval(geomRectDateLast()) +
        eval(geomRectDateBear()) +
        eval(geomRectDateBull()) +
        eval(geomRectDateTest()) +
        labs(x = "Time", y = "Value") +
        labs(title = title) +
        theme(plot.title = element_text(hjust = 0.5)) # align title in center
}

## if a special name is given, take it, otherwise take x (plot sentix by using same dataframe (adopted)
plotDataPDF <- function(x, xName = x){
    pdf(file.path(getwd(), "Plot", paste0(xName, ".pdf")), width = 10, height = 4)
    plot(plotData(x))
    dev.off()
}
```

TODO: environments in R, plug functions into environments to keep structure http://adv-r.had.co.nz/Environments.html

```
# ePlot <- new.env() # environment to store functions (doesn't work)

# ls.str(envir = ePlot)
```

probierer, funktioniert nicht (wollte alle linien auf einmal plotten)

```
# geomLineData <- function(x){
#     parse(text = paste0("eval(geomLineDataDAX(\"", x , "\")) + eval(geomLineDataTEC(\"", x , "\"))"))
# }
#
# ggplot() +
#     eval(geomLineData("retPlot")) +
#     eval(geomRectDateLast) +
#     labs(x = "Time", y = "Value")
```

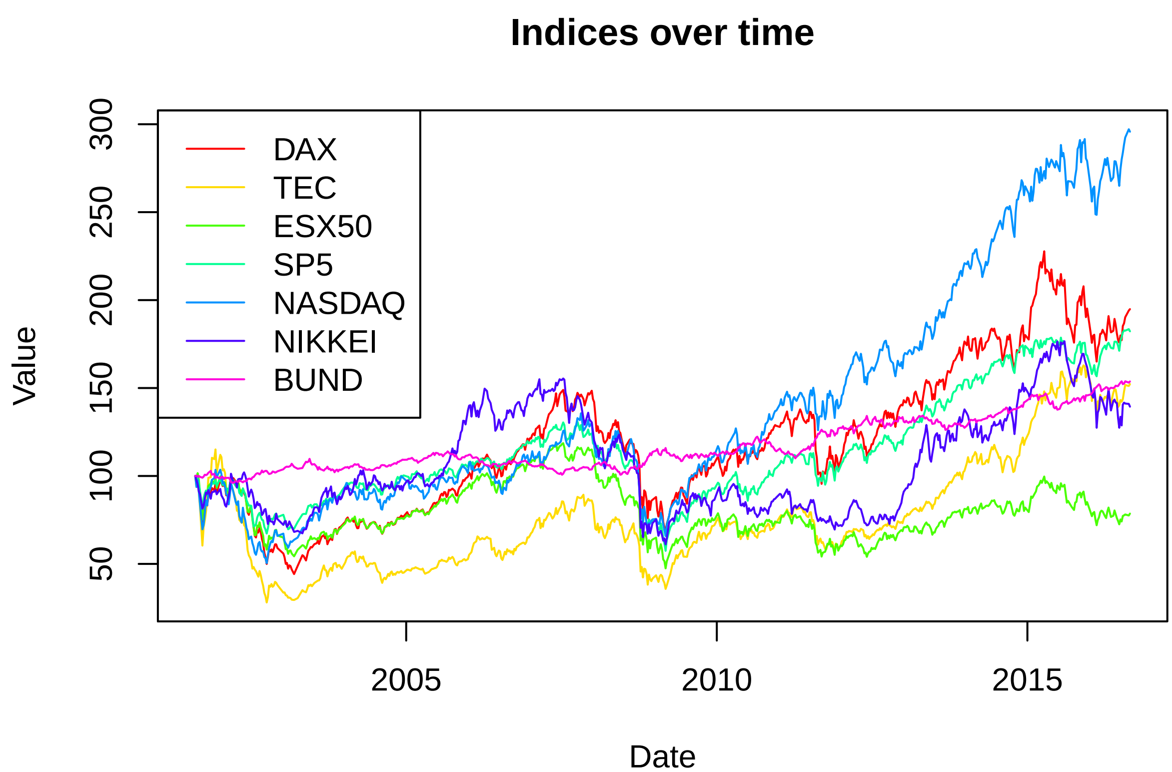
Stocks

Start of with a value of 100 for each stock and then plot the evolvment of this stock.

plot()

```
retPlot <- matrix(100, nrow = nrow(stocks), ncol = ncol(stocks)-1)
retPlot[2:nrow(stocks), ] <- 1+ret # to multiply lateron, we have to add 1
retPlot <- apply(retPlot, 2, cumprod)
rownames(retPlot) <- stocks[,1]

xNames <- rownames(retPlot)
```

```r
class(xNames) <- "Date"   # convert to date

cols <- rainbow(ncol(retPlot))
ylim <- c(min(retPlot), max(retPlot))
plot(xNames, retPlot[,1], type = "l", xlab = "Date", ylab = "Value", main = "Indices over time",
     col = cols[1], ylim = ylim)
for(sentixGroup in 2:ncol(retPlot)){
    par(new=T)
    plot(xNames, retPlot[,sentixGroup], type = "l", col = cols[sentixGroup], axes = F, xlab="", ylab=""
}
legend("topleft", legend = colnames(stocks)[2:ncol(stocks)], col = cols, lty = 1)
```




```r
rm(retPlot, xNames, ylim, sentixGroup)
```

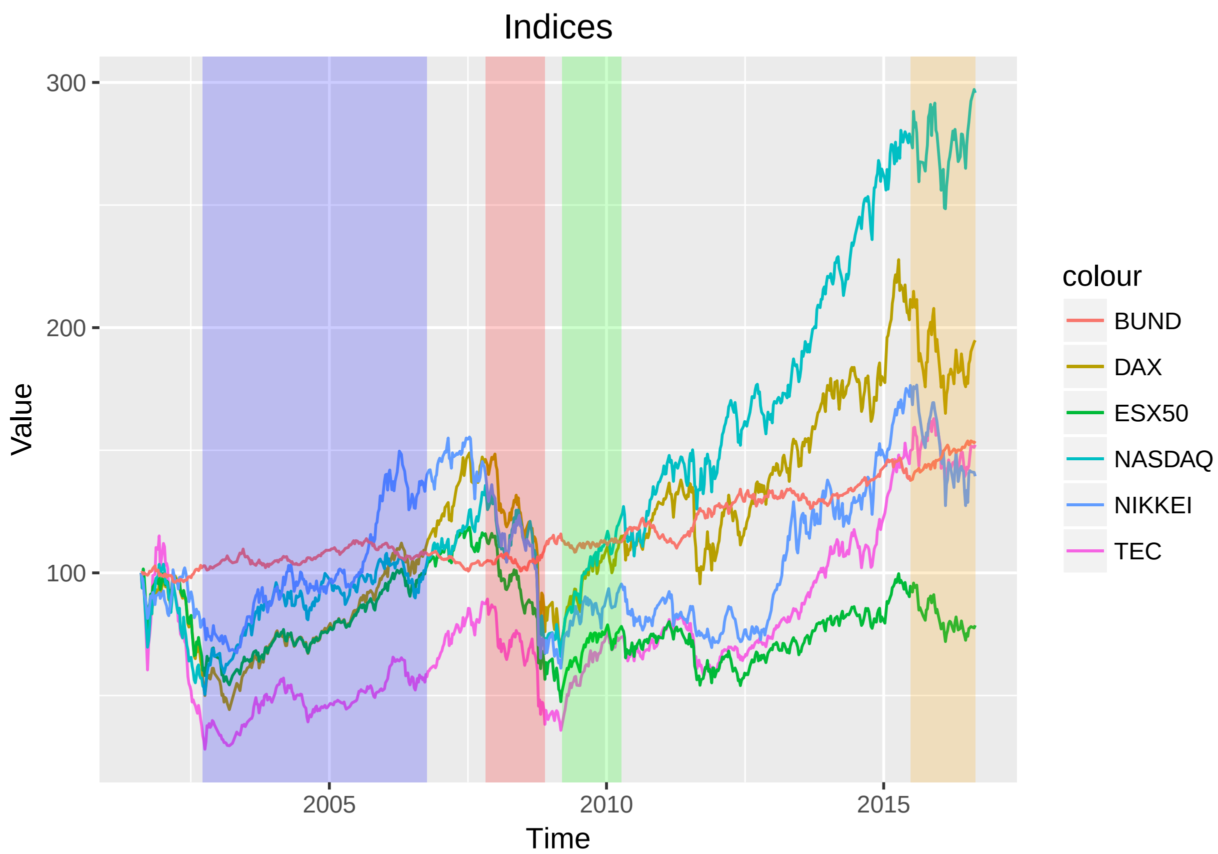
ggplot()

```r
library(ggplot2)
```

need data frame as input for ggplot

```r
retPlot <- matrix(100, nrow = nrow(stocks), ncol = ncol(stocks)-1)
retPlot[2:nrow(stocks), ] <- 1+ret # to multiply lateron, we have to add 1
retPlot <- apply(retPlot, 2, cumprod)

retPlot <- as.data.frame(retPlot)
```

```r
colnames(retPlot) <- colnames(stocks)[2:ncol(stocks)]
retPlot$Datum <- stocks[,1]
class(retPlot$Datum) <- "Date"    # convert to date

cols <- rainbow(ncol(retPlot))
ylim <- c(min(retPlot[,1:(ncol(retPlot)-1)]), max(retPlot[,1:(ncol(retPlot)-1)]))


plotData("retPlot")
```




```r
plotDataPDF("retPlot")
```

```
## pdf
##   2
```

```r
rm(retPlot, cols, ylim)
```

Dispersion

Graphs can be found in "\R-Research Project Statistics\Plot Data".

```r
lateximport <- c(paste0("\\subsection{Herfindahl}"))

for(sentixGroup in names(sDisp)){
    title <- paste("sDisp", sentixGroup)
```

```r
    sPlot <- sDisp[[sentixGroup]]
    plotDataPDF("sPlot", title)

    lateximport <- c(lateximport, paste0("\\includegraphics[width=\\textwidth]{",paste0(title, ".pdf"),
}

fileConnection <- file(file.path(getwd(), "Plot", paste0("0sentixDisp.txt")))
writeLines(lateximport, fileConnection)
close(fileConnection)

rm(sPlot, sentixGroup, lateximport, fileConnection)
```

And we provide summary statistics.

```r
lapply(sDisp, function(x) {base::summary(x[,-1], digits = 2)})
```

```
## $P1
##       DAX            TEC            ESX50           SP5
##  Min.   :0.39   Min.   :0.39   Min.   :0.39   Min.   :0.39
##  1st Qu.:0.55   1st Qu.:0.54   1st Qu.:0.53   1st Qu.:0.51
##  Median :0.58   Median :0.57   Median :0.56   Median :0.55
##  Mean   :0.58   Mean   :0.57   Mean   :0.56   Mean   :0.55
##  3rd Qu.:0.62   3rd Qu.:0.60   3rd Qu.:0.59   3rd Qu.:0.58
##  Max.   :0.76   Max.   :0.74   Max.   :0.75   Max.   :0.73
##      NASDAQ         NIKKEI          BUND
##  Min.   :0.42   Min.   :0.31   Min.   :0.15
##  1st Qu.:0.53   1st Qu.:0.48   1st Qu.:0.37
##  Median :0.56   Median :0.51   Median :0.41
##  Mean   :0.56   Mean   :0.51   Mean   :0.40
##  3rd Qu.:0.59   3rd Qu.:0.54   3rd Qu.:0.45
##  Max.   :0.74   Max.   :0.71   Max.   :0.57
##
## $P6
##       DAX            TEC            ESX50           SP5
##  Min.   :0.49   Min.   :0.46   Min.   :0.49   Min.   :0.47
##  1st Qu.:0.63   1st Qu.:0.62   1st Qu.:0.62   1st Qu.:0.61
##  Median :0.66   Median :0.65   Median :0.65   Median :0.64
##  Mean   :0.66   Mean   :0.65   Mean   :0.64   Mean   :0.64
##  3rd Qu.:0.69   3rd Qu.:0.68   3rd Qu.:0.68   3rd Qu.:0.67
##  Max.   :0.76   Max.   :0.75   Max.   :0.75   Max.   :0.75
##      NASDAQ         NIKKEI          BUND
##  Min.   :0.49   Min.   :0.37   Min.   :0.38
##  1st Qu.:0.62   1st Qu.:0.56   1st Qu.:0.49
##  Median :0.65   Median :0.60   Median :0.52
##  Mean   :0.65   Mean   :0.59   Mean   :0.52
##  3rd Qu.:0.68   3rd Qu.:0.62   3rd Qu.:0.55
##  Max.   :0.75   Max.   :0.71   Max.   :0.66
##
## $I1
##       DAX            TEC            ESX50           SP5
##  Min.   :0.30   Min.   :0.34   Min.   :0.30   Min.   :0.33
##  1st Qu.:0.55   1st Qu.:0.53   1st Qu.:0.53   1st Qu.:0.51
##  Median :0.59   Median :0.58   Median :0.58   Median :0.55
##  Mean   :0.59   Mean   :0.58   Mean   :0.58   Mean   :0.56
```

```
##  3rd Qu.:0.63   3rd Qu.:0.62   3rd Qu.:0.62   3rd Qu.:0.60
##  Max.   :0.85   Max.   :0.80   Max.   :0.83   Max.   :0.81
##      NASDAQ         NIKKEI          BUND
##  Min.   :0.31   Min.   :0.27   Min.   :0.29
##  1st Qu.:0.51   1st Qu.:0.46   1st Qu.:0.44
##  Median :0.56   Median :0.50   Median :0.49
##  Mean   :0.56   Mean   :0.51   Mean   :0.49
##  3rd Qu.:0.61   3rd Qu.:0.55   3rd Qu.:0.54
##  Max.   :0.79   Max.   :0.78   Max.   :0.78
##
## $I6
##       DAX            TEC           ESX50           SP5
##  Min.   :0.41   Min.   :0.40   Min.   :0.39   Min.   :0.44
##  1st Qu.:0.61   1st Qu.:0.61   1st Qu.:0.60   1st Qu.:0.59
##  Median :0.66   Median :0.65   Median :0.65   Median :0.63
##  Mean   :0.65   Mean   :0.65   Mean   :0.64   Mean   :0.63
##  3rd Qu.:0.70   3rd Qu.:0.69   3rd Qu.:0.69   3rd Qu.:0.68
##  Max.   :0.82   Max.   :0.80   Max.   :0.81   Max.   :0.77
##      NASDAQ         NIKKEI          BUND
##  Min.   :0.43   Min.   :0.36   Min.   :0.28
##  1st Qu.:0.60   1st Qu.:0.53   1st Qu.:0.49
##  Median :0.63   Median :0.58   Median :0.56
##  Mean   :0.63   Mean   :0.57   Mean   :0.55
##  3rd Qu.:0.67   3rd Qu.:0.62   3rd Qu.:0.61
##  Max.   :0.81   Max.   :0.73   Max.   :0.75
##
## $G1
##       DAX            TEC           ESX50           SP5
##  Min.   :0.39   Min.   :0.40   Min.   :0.39   Min.   :0.38
##  1st Qu.:0.55   1st Qu.:0.54   1st Qu.:0.54   1st Qu.:0.52
##  Median :0.59   Median :0.57   Median :0.57   Median :0.55
##  Mean   :0.59   Mean   :0.57   Mean   :0.57   Mean   :0.55
##  3rd Qu.:0.62   3rd Qu.:0.61   3rd Qu.:0.60   3rd Qu.:0.58
##  Max.   :0.78   Max.   :0.75   Max.   :0.76   Max.   :0.75
##      NASDAQ         NIKKEI          BUND
##  Min.   :0.42   Min.   :0.32   Min.   :0.21
##  1st Qu.:0.53   1st Qu.:0.48   1st Qu.:0.39
##  Median :0.56   Median :0.51   Median :0.43
##  Mean   :0.56   Mean   :0.51   Mean   :0.43
##  3rd Qu.:0.59   3rd Qu.:0.54   3rd Qu.:0.47
##  Max.   :0.75   Max.   :0.73   Max.   :0.59
##
## $G6
##       DAX            TEC           ESX50           SP5
##  Min.   :0.52   Min.   :0.48   Min.   :0.49   Min.   :0.49
##  1st Qu.:0.63   1st Qu.:0.62   1st Qu.:0.62   1st Qu.:0.61
##  Median :0.66   Median :0.66   Median :0.65   Median :0.64
##  Mean   :0.66   Mean   :0.65   Mean   :0.65   Mean   :0.64
##  3rd Qu.:0.69   3rd Qu.:0.68   3rd Qu.:0.68   3rd Qu.:0.67
##  Max.   :0.76   Max.   :0.75   Max.   :0.75   Max.   :0.75
##      NASDAQ         NIKKEI          BUND
##  Min.   :0.50   Min.   :0.39   Min.   :0.38
##  1st Qu.:0.62   1st Qu.:0.56   1st Qu.:0.49
##  Median :0.65   Median :0.59   Median :0.53
```

```
## Mean   :0.65   Mean   :0.59   Mean   :0.53
## 3rd Qu.:0.67   3rd Qu.:0.62   3rd Qu.:0.56
## Max.   :0.74   Max.   :0.71   Max.   :0.67
```

Herfindahl

Graphs can be found in "\R-Research Project Statistics\Plot Data".

```
lateximport <- c(paste0("\\subsection{Herfindahl}"))

for(sentixGroup in names(sHerf)){
    title <- paste("sHerf", sentixGroup)

    sPlot <- sHerf[[sentixGroup]]
    plotDataPDF("sPlot", title)

    lateximport <- c(lateximport, paste0("\\includegraphics[width=\\textwidth]{",paste0(title, ".pdf"),
}

fileConnection <- file(file.path(getwd(), "Plot", paste0("0sentixHerf.txt")))
writeLines(lateximport, fileConnection)
close(fileConnection)


rm(sPlot, sentixGroup, lateximport, fileConnection)
```

And we provide summary statistics.

```
lapply(sHerf, function(x) {base::summary(x[,-1], digits = 2)})
```

```
## $P1
##       DAX             TEC            ESX50             SP5
##  Min.   :-0.67   Min.   :-0.67   Min.   :-0.76   Min.   :-0.82
##  1st Qu.:-0.53   1st Qu.:-0.54   1st Qu.:-0.55   1st Qu.:-0.57
##  Median :-0.50   Median :-0.51   Median :-0.52   Median :-0.54
##  Mean   :-0.51   Mean   :-0.51   Mean   :-0.52   Mean   :-0.54
##  3rd Qu.:-0.48   3rd Qu.:-0.49   3rd Qu.:-0.49   3rd Qu.:-0.50
##  Max.   :-0.41   Max.   :-0.41   Max.   :-0.41   Max.   :-0.42
##     NASDAQ          NIKKEI            BUND
##  Min.   :-0.71   Min.   :-0.90   Min.   :-1.45
##  1st Qu.:-0.56   1st Qu.:-0.63   1st Qu.:-0.86
##  Median :-0.52   Median :-0.58   Median :-0.77
##  Mean   :-0.53   Mean   :-0.59   Mean   :-0.78
##  3rd Qu.:-0.49   3rd Qu.:-0.54   3rd Qu.:-0.67
##  Max.   :-0.41   Max.   :-0.42   Max.   :-0.51
##
## $P6
##       DAX             TEC            ESX50             SP5
##  Min.   :-0.61   Min.   :-0.66   Min.   :-0.63   Min.   :-0.65
##  1st Qu.:-0.47   1st Qu.:-0.47   1st Qu.:-0.47   1st Qu.:-0.48
##  Median :-0.45   Median :-0.45   Median :-0.46   Median :-0.46
##  Mean   :-0.45   Mean   :-0.46   Mean   :-0.46   Mean   :-0.47
##  3rd Qu.:-0.43   3rd Qu.:-0.44   3rd Qu.:-0.44   3rd Qu.:-0.45
##  Max.   :-0.40   Max.   :-0.41   Max.   :-0.41   Max.   :-0.41
##     NASDAQ          NIKKEI            BUND
```

```
##  Min.   :-0.61   Min.   :-0.87   Min.   :-0.71  
##  1st Qu.:-0.47   1st Qu.:-0.51   1st Qu.:-0.58  
##  Median :-0.45   Median :-0.49   Median :-0.54  
##  Mean   :-0.46   Mean   :-0.50   Mean   :-0.55  
##  3rd Qu.:-0.44   3rd Qu.:-0.47   3rd Qu.:-0.52  
##  Max.   :-0.41   Max.   :-0.42   Max.   :-0.45  
## 
## $I1
##       DAX              TEC             ESX50              SP5        
##  Min.   :-0.76   Min.   :-0.74   Min.   :-0.73   Min.   :-0.81  
##  1st Qu.:-0.53   1st Qu.:-0.56   1st Qu.:-0.54   1st Qu.:-0.58  
##  Median :-0.50   Median :-0.51   Median :-0.51   Median :-0.53  
##  Mean   :-0.50   Mean   :-0.52   Mean   :-0.51   Mean   :-0.54  
##  3rd Qu.:-0.47   3rd Qu.:-0.48   3rd Qu.:-0.47   3rd Qu.:-0.49  
##  Max.   :-0.40   Max.   :-0.40   Max.   :-0.40   Max.   :-0.40  
##      NASDAQ           NIKKEI            BUND        
##  Min.   :-0.76   Min.   :-1.10   Min.   :-1.03  
##  1st Qu.:-0.58   1st Qu.:-0.64   1st Qu.:-0.69  
##  Median :-0.53   Median :-0.58   Median :-0.61  
##  Mean   :-0.54   Mean   :-0.59   Mean   :-0.63  
##  3rd Qu.:-0.49   3rd Qu.:-0.53   3rd Qu.:-0.54  
##  Max.   :-0.40   Max.   :-0.40   Max.   :-0.42  
## 
## $I6
##       DAX              TEC             ESX50              SP5        
##  Min.   :-0.61   Min.   :-0.68   Min.   :-0.60   Min.   :-0.71  
##  1st Qu.:-0.48   1st Qu.:-0.48   1st Qu.:-0.49   1st Qu.:-0.50  
##  Median :-0.45   Median :-0.45   Median :-0.45   Median :-0.47  
##  Mean   :-0.46   Mean   :-0.46   Mean   :-0.46   Mean   :-0.47  
##  3rd Qu.:-0.43   3rd Qu.:-0.43   3rd Qu.:-0.43   3rd Qu.:-0.44  
##  Max.   :-0.40   Max.   :-0.40   Max.   :-0.40   Max.   :-0.41  
##      NASDAQ           NIKKEI            BUND        
##  Min.   :-0.65   Min.   :-0.83   Min.   :-0.97  
##  1st Qu.:-0.49   1st Qu.:-0.53   1st Qu.:-0.56  
##  Median :-0.47   Median :-0.50   Median :-0.51  
##  Mean   :-0.47   Mean   :-0.51   Mean   :-0.52  
##  3rd Qu.:-0.44   3rd Qu.:-0.48   3rd Qu.:-0.48  
##  Max.   :-0.41   Max.   :-0.41   Max.   :-0.42  
## 
## $G1
##       DAX              TEC             ESX50              SP5        
##  Min.   :-0.65   Min.   :-0.67   Min.   :-0.67   Min.   :-0.77  
##  1st Qu.:-0.53   1st Qu.:-0.54   1st Qu.:-0.54   1st Qu.:-0.57  
##  Median :-0.50   Median :-0.51   Median :-0.52   Median :-0.53  
##  Mean   :-0.50   Mean   :-0.51   Mean   :-0.52   Mean   :-0.54  
##  3rd Qu.:-0.48   3rd Qu.:-0.48   3rd Qu.:-0.49   3rd Qu.:-0.50  
##  Max.   :-0.40   Max.   :-0.41   Max.   :-0.41   Max.   :-0.41  
##      NASDAQ           NIKKEI            BUND        
##  Min.   :-0.71   Min.   :-0.94   Min.   :-1.27  
##  1st Qu.:-0.56   1st Qu.:-0.62   1st Qu.:-0.80  
##  Median :-0.52   Median :-0.58   Median :-0.72  
##  Mean   :-0.53   Mean   :-0.59   Mean   :-0.73  
##  3rd Qu.:-0.49   3rd Qu.:-0.54   3rd Qu.:-0.64  
##  Max.   :-0.41   Max.   :-0.41   Max.   :-0.49  
```

```
##
## $G6
##       DAX             TEC            ESX50            SP5
##  Min.   :-0.56   Min.   :-0.63   Min.   :-0.58   Min.   :-0.60
##  1st Qu.:-0.46   1st Qu.:-0.47   1st Qu.:-0.47   1st Qu.:-0.48
##  Median :-0.45   Median :-0.45   Median :-0.45   Median :-0.46
##  Mean   :-0.45   Mean   :-0.46   Mean   :-0.46   Mean   :-0.47
##  3rd Qu.:-0.43   3rd Qu.:-0.44   3rd Qu.:-0.44   3rd Qu.:-0.44
##  Max.   :-0.40   Max.   :-0.41   Max.   :-0.41   Max.   :-0.41
##      NASDAQ          NIKKEI           BUND
##  Min.   :-0.60   Min.   :-0.82   Min.   :-0.68
##  1st Qu.:-0.47   1st Qu.:-0.51   1st Qu.:-0.57
##  Median :-0.46   Median :-0.49   Median :-0.53
##  Mean   :-0.46   Mean   :-0.50   Mean   :-0.54
##  3rd Qu.:-0.44   3rd Qu.:-0.48   3rd Qu.:-0.51
##  Max.   :-0.41   Max.   :-0.42   Max.   :-0.44
```

Analysis

Dispersion to Returns

We first want to look on how dispersion affects returns. We hypothesize that (future) return is higher if dispersion is lower. Therefore, we look at the mean return of the q quantil of dispersion and the mean return of its *1-q* quantil. We do this for all the sentiments in comparison with all stocks.

We depict this value. row: index, column: sentiment We also depict the ranks (higher rank = higher value). Be careful as the absolute values of returns are different across indices (therefore ranking is not really justified) Comparing in each row is justified and higher value is good (should expecially be greater than 0).

Let n be the number of periods considered (n=1: just this period, n=2: this and next period) and let m be the time lapse (m=0: returns starting right now, m=1: returns starting 1 one period behind).

```
q <- 0.1

compareDispRet <- function(n, m=0){
    res <- matrix(NA, nrow = ncol(ret), ncol = length(sDisp))
    rownames(res) <- colnames(ret)
    colnames(res) <- names(sDisp)

    for(d in 1:length(names(sDisp))){
        for(s in 1:ncol(ret)){
            dat <- data.frame(disp = sDisp[[d]][2:(nrow(sDisp[[d]])-n+1-m),s+1])
            for(k in 1:(nrow(ret)-n+1-m)){
                dat[k,"r"] <- prod(1+ret[(k+m):(k+m+n-1),s])-1
            }
            dat <- dat[order(dat$disp),] # ascending by default
            res[s, d] <- round( mean(dat[1:(q*nrow(dat)),"r"]) - mean(dat[((1-q)*nrow(dat)):nrow(dat),
        }
    }
    return(res)
}
```

actual dispersion to actual return, no lag

dispersion in connection with return of same period

So I1 seems to be able to predict returns, while P6 does not.

```
res <- compareDispRet(1)
res
```

```
##            P1     P6     I1     I6    G1     G6
## DAX     0.018  0.001  0.034 -0.004 0.027  0.001
## TEC     0.000 -0.002  0.018  0.011 0.006  0.007
## ESX50   0.012 -0.004  0.029  0.000 0.023 -0.001
## SP5     0.001 -0.001  0.017  0.004 0.011 -0.001
## NASDAQ  0.002 -0.003  0.018  0.003 0.011  0.002
## NIKKEI  0.005  0.014  0.021  0.019 0.013  0.020
## BUND    0.000  0.001 -0.001  0.003 0.000  0.001
```

```
matrix(rank(res), ncol = ncol(res), dimnames = list(rownames(res), colnames(res)))
```

```
##           P1   P6   I1   I6   G1   G6
```

```
## DAX    34.0 15.0 42.0  1.5 40.0 15.0
## TEC    10.5  4.0 34.0 27.0 24.0 25.0
## ESX50  29.0  1.5 41.0 10.5 39.0  6.5
## SP5    15.0  6.5 32.0 22.0 27.0  6.5
## NASDAQ 18.5  3.0 34.0 20.5 27.0 18.5
## NIKKEI 23.0 31.0 38.0 36.0 30.0 37.0
## BUND   10.5 15.0  6.5 20.5 10.5 15.0
```

actual dispersion to actual return, lag of 1

dispersion in connection with return of same period

```r
res <- compareDispRet(1, 1)
res
```

```
##            P1     P6     I1     I6     G1     G6
## DAX    -0.008  0.001  0.001  0.012 -0.010  0.002
## TEC    -0.021 -0.007  0.002  0.012 -0.014 -0.003
## ESX50  -0.003 -0.004 -0.002  0.004 -0.008 -0.002
## SP5    -0.007 -0.008 -0.001 -0.001 -0.003 -0.004
## NASDAQ  0.002 -0.007  0.006  0.012  0.004 -0.004
## NIKKEI  0.000  0.006  0.003  0.009 -0.003  0.012
## BUND    0.000 -0.001  0.002 -0.002  0.001 -0.002
```

```r
matrix(rank(res), ncol = ncol(res), dimnames = list(rownames(res), colnames(res)))
```

```
##          P1   P6   I1   I6   G1   G6
## DAX     5.0 27.0 27.0 40.5  3.0 30.5
## TEC     1.0  8.0 30.5 40.5  2.0 14.5
## ESX50  14.5 11.0 18.5 34.5  5.0 18.5
## SP5     8.0  5.0 22.0 22.0 14.5 11.0
## NASDAQ 30.5  8.0 36.5 40.5 34.5 11.0
## NIKKEI 24.5 36.5 33.0 38.0 14.5 40.5
## BUND   24.5 22.0 30.5 18.5 27.0 18.5
```

actual dispersion with future return (n=3), no lag

dispersion of one period with return over next n periods (this period up to n-1 period).

```r
res <- compareDispRet(3)
res
```

```
##            P1     P6    I1    I6     G1     G6
## DAX     0.007  0.000 0.028 0.008  0.006  0.004
## TEC    -0.031 -0.008 0.013 0.037 -0.018  0.003
## ESX50   0.001 -0.012 0.025 0.000 -0.001 -0.005
## SP5    -0.010 -0.016 0.014 0.004  0.001 -0.009
## NASDAQ -0.002 -0.013 0.019 0.018  0.006  0.002
## NIKKEI -0.002  0.020 0.022 0.031  0.008  0.035
## BUND   -0.002  0.001 0.001 0.002 -0.002  0.001
```

```r
matrix(rank(res), ncol = ncol(res), dimnames = list(rownames(res), colnames(res)))
```

```
##          P1   P6 I1   I6   G1   G6
## DAX    29.0 15.5 39 30.5 27.5 25.5
## TEC     1.0  8.0 32 42.0  2.0 24.0
```

```
## ESX50  19.0  5.0 38 15.5 14.0  9.0
## SP5     6.0  3.0 33 25.5 19.0  7.0
## NASDAQ 11.5  4.0 35 34.0 27.5 22.5
## NIKKEI 11.5 36.0 37 40.0 30.5 41.0
## BUND   11.5 19.0 19 22.5 11.5 19.0
```

actual dispersion with future return (n=6), no lag

dispersion of one period with return over next n periods (this period up to n-1 period).

```r
res <- compareDispRet(6)
res
```

```
##            P1     P6     I1     I6     G1     G6
## DAX     0.001  0.000  0.010 -0.012 -0.004  0.001
## TEC    -0.034 -0.027 -0.008  0.066 -0.034  0.001
## ESX50  -0.010 -0.010  0.003  0.004 -0.015  0.006
## SP5    -0.010 -0.030  0.002  0.009 -0.007 -0.025
## NASDAQ  0.000 -0.030 -0.002  0.038  0.001 -0.014
## NIKKEI  0.000  0.026  0.021  0.059  0.004  0.051
## BUND   -0.006  0.001  0.001  0.006 -0.004  0.002
```

```r
matrix(rank(res), ncol = ncol(res), dimnames = list(rownames(res), colnames(res)))
```

```
##          P1   P6   I1   I6   G1   G6
## DAX    24.5 20.0 36.0  9.0 16.5 24.5
## TEC     1.5  5.0 13.0 42.0  1.5 24.5
## ESX50  11.0 11.0 30.0 31.5  7.0 33.5
## SP5    11.0  3.5 28.5 35.0 14.0  6.0
## NASDAQ 20.0  3.5 18.0 39.0 24.5  8.0
## NIKKEI 20.0 38.0 37.0 41.0 31.5 40.0
## BUND   15.0 24.5 24.5 33.5 16.5 28.5
```

```r
rm(q, res, compareDispRet)
```

Optimization of Portfolios

classic portfolio optimization

First of all, we do a classic portfolio optimization. We start of with a mean variance diagram.

notation

Let $x = (x_1, ..., x_p)^T$ represent the portfolio (x_i is percentage of available capital invested in security i). Therefore it holds $\sum_{i=1}^{p} x_i = 1$. Note, that short selling is allowed.

Let $R = (R_1, ..., R_p)^T$ represent the annual returns (R_i is return of security i). And let $\mu = (\mu_1, ..., \mu_p)^T$ represent the expected returns ($\mu_i = \mathrm{E}[R_i] > 0$).

Furthermore $C = (c_{ij})_{i,j \in \{1,...,p\}}$ denotes the (annual) covariance matrix ($c_{ij} = \mathrm{Cov}(R_i, R_j)$).

Then we have Return $R(x)$ of portfolio x given by $R(x) = \sum_{i=1}^{p} x_i R_i = x^T R$.

The expected return $\mu(x)$ of portfolio x is given by $\mu(x) = \mathrm{E}[R(x)] = \sum_{i=1}^{p} x_i \mu_i = x^T \mu$.

The Variance $\sigma^2(x)$ of portfolio x is given by $\sigma^2(x) = \mathrm{Var}(R(x)) = \mathrm{E}[(R(x) - \mathrm{E}(R(x)))^2] = x^T C x$.

We therefore annualize the returns and the variance.

```
anRet <- (1+ret)^52-1
anMu <- (1+mu)^52-1
anC <- C*52
```

We furthermore exclude riskless asset (assume BUND to be risk free)

```
retRisky <- ret[,-7]
colnames(retRisky)
```

```
## [1] "DAX"    "TEC"    "ESX50"  "SP5"    "NASDAQ" "NIKKEI"
```

```
muRisky <- colMeans(retRisky)
CRisky <- cov(retRisky)

anRetRisky <- (1+retRisky)^52-1
anMuRisky <- (1+muRisky)^52-1
anCRisky <- CRisky*52
```

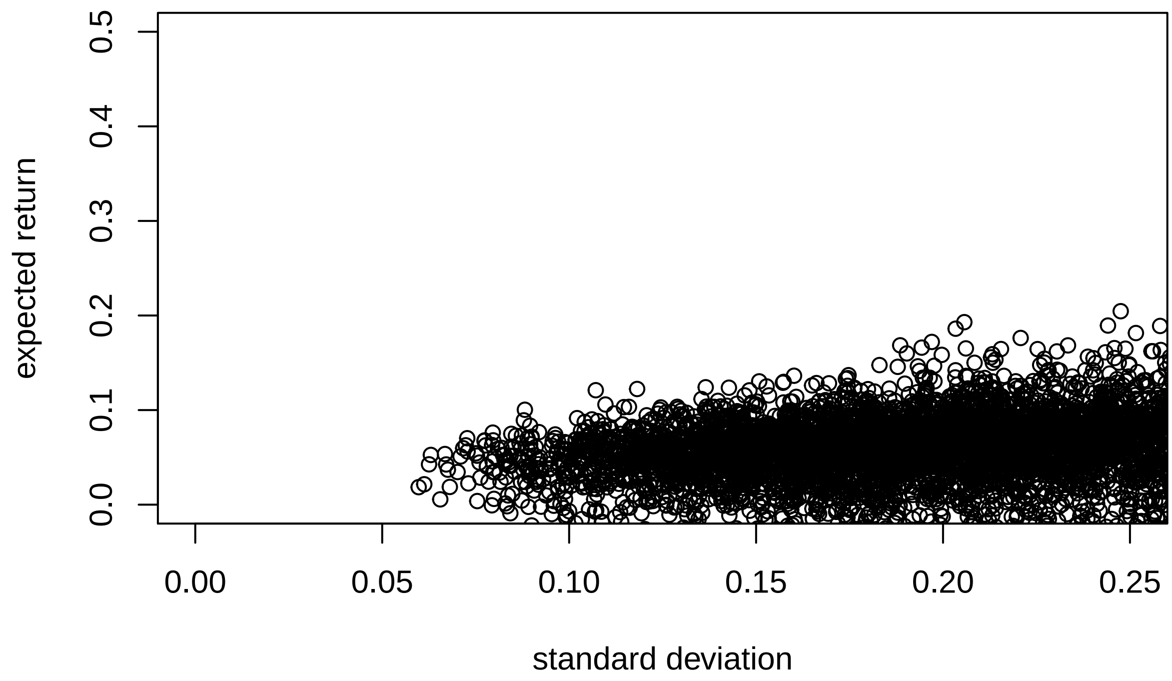
mean variance diagram

We plot K random portfolios.

with riskfree asset

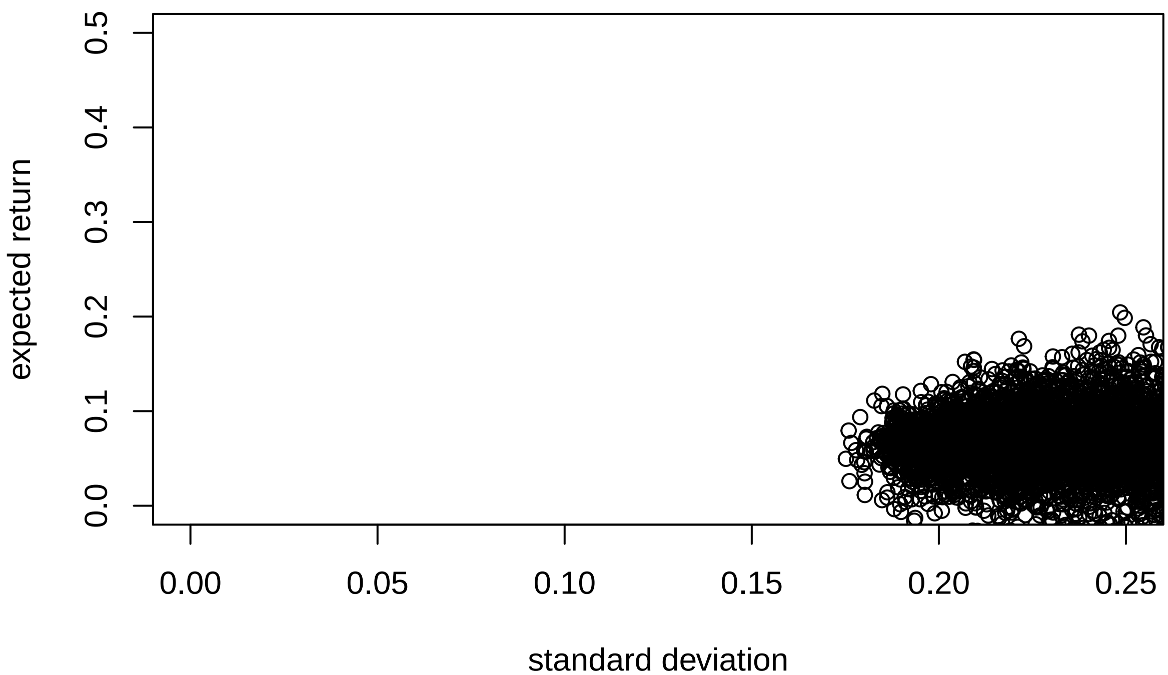
```
set.seed(1)
K <- 10000

mvRandom <- matrix(0, ncol = 2, nrow = K)
for(i in 1:nrow(mvRandom)){
    x <- rnorm(ncol(ret))
    x <- x/sum(x) # normalize

    mvRandom[i, 1] <- sum(x*anMu)
```

```
    mvRandom[i, 2] <- sqrt((x%*%anC)%*%x)
}

plot(mvRandom[,2], mvRandom[,1],
     xlab = "standard deviation", ylab = "expected return",
     xlim = c(0, 0.25), ylim = c(0, 0.5))
```



without risk free asset

```
set.seed(1)
K <- 10000

mvRandom <- matrix(0, ncol = 2, nrow = K)
for(i in 1:nrow(mvRandom)){
    x <- rnorm(ncol(retRisky))
    x <- x/sum(x) # normalize

    mvRandom[i, 1] <- sum(x*anMuRisky)
    mvRandom[i, 2] <- sqrt((x%*%anCRisky)%*%x)
}

plot(mvRandom[,2], mvRandom[,1],
     xlab = "standard deviation", ylab = "expected return",
     xlim = c(0, 0.25), ylim = c(0, 0.5))
```



efficiency

We can use theorem 2.2. of Portfolio Analysis (slide 40). But be careful as C is close to singular.

efficiency line by formula d)

```
det(anC)
```

```
## [1] 1.05804e-13
```

```
det(anCRisky)
```

```
## [1] 3.151767e-11
```

without risk free asset

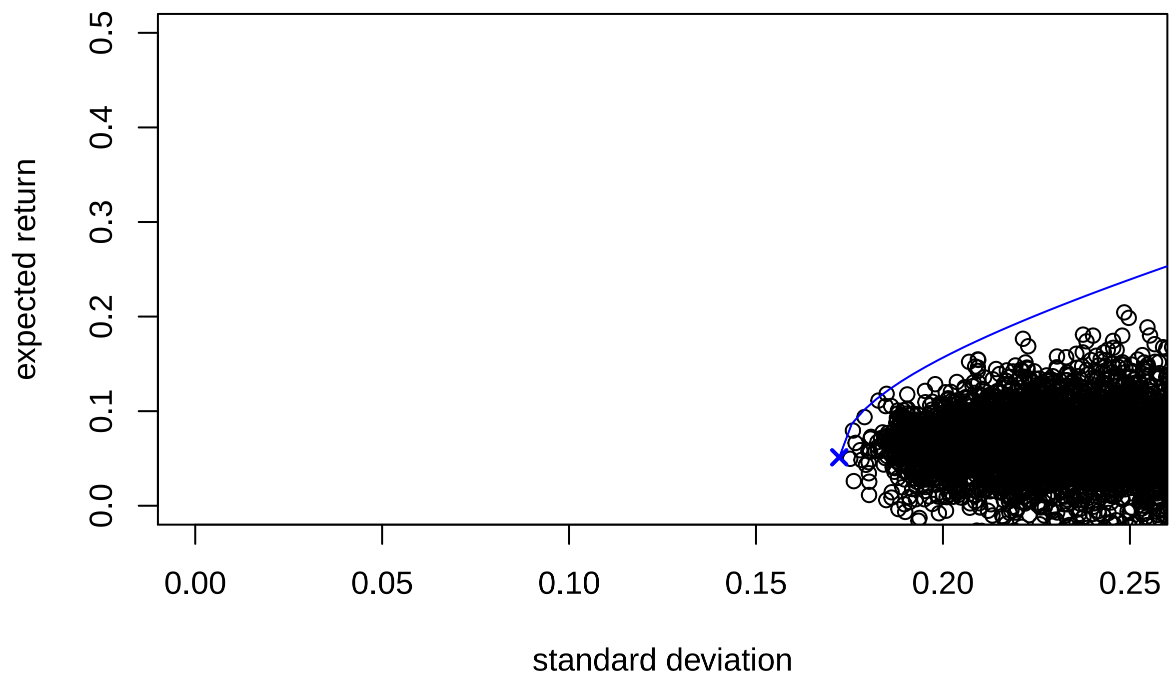
```
anCRisky1 <- solve(anCRisky)
anCRisky %*% anCRisky1
```

```
##                  DAX           TEC         ESX50           SP5
## DAX    1.000000e+00 -5.551115e-17 -2.775558e-16 -4.163336e-16
## TEC    2.126771e-15  1.000000e+00  1.498801e-15 -1.276756e-15
## ESX50  1.491862e-15  4.163336e-16  1.000000e+00 -2.220446e-16
## SP5    1.261144e-15  3.608225e-16  1.665335e-16  1.000000e+00
## NASDAQ 1.065120e-15  3.191891e-16 -3.330669e-16  6.383782e-16
## NIKKEI 8.326673e-16 -2.220446e-16 -5.551115e-16 -8.326673e-16
##               NASDAQ        NIKKEI
```

```
## DAX    -2.359224e-16 -2.220446e-16
## TEC     7.077672e-16 -2.220446e-16
## ESX50  -2.359224e-16 -2.220446e-16
## SP5    -1.734723e-16 -1.110223e-16
## NASDAQ  1.000000e+00  0.000000e+00
## NIKKEI  8.049117e-16  1.000000e+00
```

```r
a <- sum(anCRisky1 %*% anMuRisky)
b <- c((anMuRisky %*% anCRisky1) %*% anMuRisky)
c <- sum(anCRisky1)
d <- b*c - a^2
```

```r
set.seed(1)
K <- 10000

mvRandom <- matrix(0, ncol = 2, nrow = K)
for(i in 1:nrow(mvRandom)){
    x <- rnorm(ncol(retRisky))
    x <- x/sum(x) # normalize

    mvRandom[i, 1] <- sum(x*anMuRisky)
    mvRandom[i, 2] <- sqrt((x%*%anCRisky)%*%x)
}

plot(mvRandom[,2], mvRandom[,1],
     xlab = "standard deviation", ylab = "expected return",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

k <- 100
elWithout <- matrix(0, ncol = 2, nrow = k)
elWithout[,2] <- seq(sqrt(1/c), 0.5, length.out = k)
for(i in 1:nrow(elWithout)){
    elWithout[i,1] <- a/c + sqrt(d/c*(elWithout[i,2]^2 - 1/c))
}
par(new=T)
plot(elWithout[,2], elWithout[,1], type = "l", col = "blue",
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

par(new=T)
plot(sqrt(1/c), a/c,
     col = "blue", pch = 4, lwd = 2,
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))
```




```r
(xMVPwithoutRF <- 1/c*rowSums(anCRisky1))
```

```
##         DAX         TEC       ESX50         SP5      NASDAQ      NIKKEI
## -0.16044087 -0.09906128 -0.09838768  1.31668249 -0.21422886  0.25543620
```

```r
c(a/c, xMVPwithoutRF %*% anMuRisky)
```

```
## [1] 0.0512193 0.0512193
```

```r
c(sqrt(1/c), sqrt( (xMVPwithoutRF%*%anCRisky)) %*% xMVPwithoutRF)
```

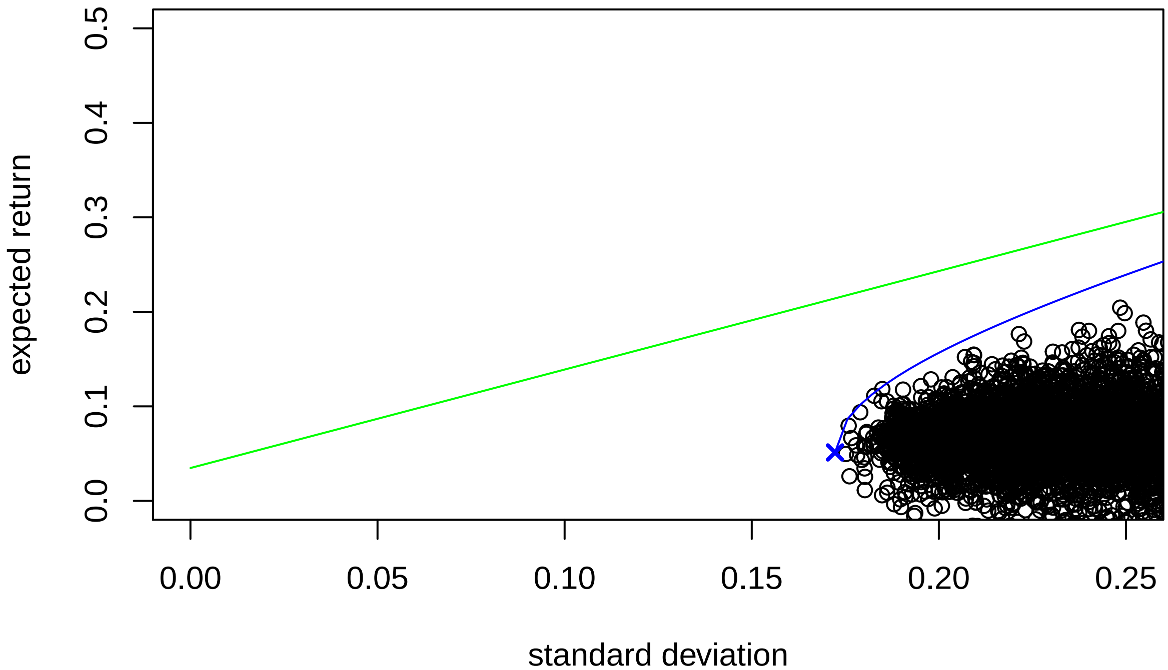
```
## [1] 0.1722548 0.1722548
```

with risk free asset

assume BUND to be risk free

```r
r <- anMu[7]

set.seed(1)
K <- 10000

mvRandom <- matrix(0, ncol = 2, nrow = K)
for(i in 1:nrow(mvRandom)){
    x <- rnorm(ncol(retRisky))
    x <- x/sum(x) # normalize

    mvRandom[i, 1] <- sum(x*anMuRisky)
```

```
    mvRandom[i, 2] <- sqrt((x%*%anCRisky)%*%x)
}

plot(mvRandom[,2], mvRandom[,1],
     xlab = "standard deviation", ylab = "expected return",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

k <- 100
elWithout <- matrix(0, ncol = 2, nrow = k)
elWithout[,2] <- seq(sqrt(1/c), 0.5, length.out = k)
for(i in 1:nrow(elWithout)){
    elWithout[i,1] <- a/c + sqrt(d/c*(elWithout[i,2]^2 - 1/c))
}
par(new=T)
plot(elWithout[,2], elWithout[,1], type = "l", col = "blue",
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

par(new=T)
plot(sqrt(1/c), a/c,
     col = "blue", pch = 4, lwd = 2,
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

elWith <- matrix(0, ncol = 2, nrow = k)
elWith[,2] <- seq(0, 0.5, length.out = k)
for(i in 1:nrow(elWith)){
    elWith[i,1] <- r + elWith[i,2]*sqrt(c*r^2 - 2*a*r + b)
}
par(new=T)
plot(elWith[,2], elWith[,1], type = "l", col = "green",
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))
```




```
(xMarket <- 1/(a-c*r)*anCRisky1%*%(anMuRisky-r))
```

```
##                [,1]
## DAX      20.1538293
## TEC      -1.3669675
## ESX50   -24.0025806
## SP5       0.4176436
## NASDAQ    5.0341532
## NIKKEI    0.7639219
```

```
unname((b-a*r)/(a-c*r))
```

```
## [1] 1.991479
```

```
unname((c*r^2 - 2*a*r + b)/(a-c*r)^2)
```

```
## [1] 3.52626
```

cleanup

```
rm(a, anCRisky1, b, c, d, elWith, elWithout, i, k, K, mvRandom, r, x)
rm(anC, anCRisky, anRet, anRetRisky, CRisky, xMarket, anMuRisky, muRisky, retRisky, xMVPwithoutRF)
```

with sentiment

find optimal weights for goal function (grid search)

IDEE: one could also look at just the previous n dates to calculate the average annual quantities.

general setup

We use several packages for the optimization.

```
library(Rdonlp2)
```

Setup Grid. Take care that weights sum up to 1, each weight is at least *wmin* and at most *wmax*.

```
stepsPerWeight <- 19
wmin <- 0.05
wmax <- 0.95
weights <- seq(wmin, wmax, length.out = stepsPerWeight)
grid <- expand.grid(w1 = weights, w2 = weights, w3 = weights )
grid <- grid[abs(rowSums(grid) - 1.0) < 0.0001,]
rownames(grid) <- 1:nrow(grid)

nrow(grid)
```

```
## [1] 171
```

```
rm(stepsPerWeight, wmin, wmax, weights)
```

With this setup, we have 171 combinations of weights.

Overview of what data we use.

```
# Return
targetRpa <- 0.06 ## targeted return of 6 % p.a.

# Volatility
targetVolpa <- 0.04 ## % p.a.

# Dispersion
targetDisp <- 0.58 ## found as it looks promising and reachable in the analysis



IneqA <- matrix(1, nrow = 1, ncol = ncol(ret)) # to take care of investments
```

dispersion direct min

We handle dispersion like return in the first place. Therefore we have the following objective functions:

1. return $\max\left(w_1 \cdot \frac{x^T\mu}{\mu_{target}}\right)$
2. volatility $\min\left(w_2 \cdot \frac{\sqrt{x^TCx}}{\sigma_{\text{target}}}\right))$
3. dispersion $\min\left(w_3 \cdot \frac{x^T\text{d}}{\text{d}_{\text{target}}}\right)$

where d denotes the annualized dispersion of each index.

We will minimize the following objective function. Be aware that maximizing something equals minimizing its negative. Furthermore *anDOpt* denotes the annualized dispersion of the indizes. We divide by the target values to have the different components of the objective function comparable (in units of the corresponding target value). We denote *Opt* to be the (newly calculated) data.

```
hDispersionDirectMin <- function(x){
    y <- numeric(3)
    y[1] <- -1.0 * w[1] * drop(crossprod(x, anMuOpt)) / targetRpa
    y[2] <- w[2] * drop(sqrt(t(x) %*% anCOpt %*% x)) * sqrt(12) / targetVolpa
    y[3] <- w[3] * drop(crossprod(x, anDOpt)) / targetDisp
    return(sum(y))
}
```

constant portfolio weights over time window

NOTE: We keep structure, as we might change lateron the test window to come up with different weights for the different time periods (one test window to get weights for bear market, another test window to get weights for bull market, . . .) NOTE: Then also adopt for dopar

First, we fix the weights x_i of each security at the beginning of (at the date before) the time window and keep them constant over time.

We store our results in the following data structure (levels of list), while having in mind that we might create a ternary plot lateron (therefore weights inside).

time window -> dispersion (sentixDataNames) -> weights of goal function -> weights of assets

We store the solution (the weights of assets), the objective value and the time needed for the computation (in seconds).

Work in parallel.

```
library(foreach)
library(parallel) # detectCores()
library(doSNOW)
```

We save with saveRDS() to be able to import and compare different results.

```
cores <- detectCores()

if(Sys.getenv("USERNAME") == "Stefan"){
    cl <- makeCluster(cores - 1)
} else if(Sys.getenv("USERNAME") == "gloggest"){
    cl <- makeCluster(cores) # use server fully
} else
    stop("Who are you???")


xDispConstTest <- list()

registerDoSNOW(cl)
xDispConstTest <- foreach(t = datesTestNames, .export = c(datesTestNames), .packages = c("Rdonlp2")) %do
    L <- list()
    timeInd <- which(datesAll == min(get(t)))-1 ## one day before start of time window

    retOpt <- ret[1:timeInd,]
    anMuOpt <- (1+colMeans(retOpt))^52-1
    anCOpt <- cov(retOpt)*52

    for(sentixGroup in names(sDisp)){
```

```
        anDOpt <- colMeans(sDisp[[sentixGroup]][1:timeInd,-1])

        for(weightInd in 1:nrow(grid)){
            w <- unlist(grid[weightInd,])

            erg <- donlp2NLP(start = rep(1/ncol(retOpt), ncol(retOpt)), fun = hDispersionDirectMin,
                      par.lower = rep(0, ncol(retOpt)), ineqA = IneqA,
                      ineqA.lower = 1.0, ineqA.upper = 1.0)
            L[[sentixGroup]][[paste(w, collapse = "-")]] <- list(x = erg$solution, obj = erg$objective,
                                                   time = as.numeric(erg$elapsed))
        }
    }
    L
}
stopCluster(cl)

names(xDispConstTest) <- datesTestNames

saveRDS(xDispConstTest, file = file.path(getwd(), "Optimization", paste0("EDispersionMinConstantTest_",
```

We now add the returns and the variance over the test time window to lateron calculate the Sharpe Ratio and determine the best weights for each sentiment group.

The function takes the calculated weighted of the assets as inputs and outputs (in the same data structure) the portfolio weights, its return and its variance over the test time window.

```
xDispConstTest <- readRDS(file.path(getwd(), "Optimization", "EDispersionMinConstantTest_gloggest2017-08
```

```
calcEvalTestConst <- function(dat){
    res <- list()
    for(t in names(dat)){
        retTest <- ret[get(t),]
        muTest <- apply((1+retTest), 2, function(x) {prod(x)-1}) # total return (over whole period)
        sigmaTest <- cov(ret) # variance (over whole period)
        rf <- muTest["BUND"]

        res[[t]] <- lapply(dat[[t]], function(x) {
            lapply(x, function(y){
                list(r <- (crossprod(y$x, muTest)-rf), sd <- sqrt(y$x %*% sigmaTest %*% y$x),
                     sr = r/sd, fweight = y$obj)
            })
        })
    }
    return(res)
}
temp <- calcEvalTestConst(xDispConstTest)
```

PROBLEM: from *t=1* to *t=50*, we have negative returns of the stocks, therefore, we invest fully in BUND, if we put enough weight on return. This is not, what we want => go directly to different portfolio weights over (test) time window

```
rm(temp, cl, calcEvalTestConst, xDispConstTest)
```

```
## Warning in rm(temp, cl, calcEvalTestConst, xDispConstTest): Objekt 'cl'
## nicht gefunden
```

different portfolio weights over time window

We evaluate an optimal portfolio at each date within our time period and assume that we can redistribute our wealth at no cost.

We use a moving time window of k dates before the actual date to determine mean and variance and therefore to determine the portfolio. Furthermore, we just use the actual dispersion.

We move the parallelization further inside to be sure that we make use of parallelization (might just have one test window).

End result has the following structure: time window -> dispersion (sentixDataNames) -> weights of goal function -> dates in time window -> weights of assets

Last weight for penultimate date of time window (hold until last date).

determine portfolio in test time window (for each gridpoint)

```
k <- 50

cores <- detectCores()

if(Sys.getenv("USERNAME") == "Stefan"){
    cl <- makeCluster(cores - 1)
} else if(Sys.getenv("USERNAME") == "gloggest"){
    cl <- makeCluster(cores) # use server fully
} else
    stop("Who are you???")


xDispVarTest <- list()

for(t in datesTestNames){

    xDispVarTest[[t]] <- foreach(i = names(sDisp), .export = c(datesTestNames), .packages = c("Rdonlp2")
        L <- list()

        for(weightInd in 1:nrow(grid)){
            w <- unlist(grid[weightInd,])

            mat <- matrix(NA, nrow = (length(get(t))-1), ncol = ncol(ret))
            colnames(mat) <- colnames(ret)
            rownames(mat) <- get(t)[1:(length(get(t))-1)]
            obj <- numeric(length(get(t))-1)
            tim <- numeric(length(get(t))-1)

            # first separate to then use the previous solution as starting point for next solution
            ### -------------------
            j <- 1
            tInd <- which(datesAll == get(t)[j])
            retOpt <- ret[(tInd-k+1):tInd,]
            anMuOpt <- (1+colMeans(retOpt))^52-1
            anCOpt <- cov(retOpt)*52
            anDOpt <-  as.numeric(sDisp[[i]][tInd,-1])

            erg <- donlp2NLP(start = rep(1/ncol(retOpt), ncol(retOpt)), fun = hDispersionDirectMin,
                             par.lower = rep(0, ncol(retOpt)), ineqA = IneqA,
```

```
                              ineqA.lower = 1.0, ineqA.upper = 1.0)
            mat[1,] <- erg$solution
            obj[1] <- erg$objective
            tim[1] <- as.numeric(erg$elapsed)
            ### -------------------

            for(j in 2:(length(get(t))-1)){
                tInd <- which(datesAll == get(t)[j])
                retOpt <- ret[(tInd-k+1):tInd,]
                anMuOpt <- (1+colMeans(retOpt))^52-1
                anCOpt <- cov(retOpt)*52
                anDOpt <- as.numeric(sDisp[[i]][tInd,-1])

                erg <- donlp2NLP(start = mat[j-1,], fun = hDispersionDirectMin,
                                  par.lower = rep(0, ncol(retOpt)), ineqA = IneqA,
                                  ineqA.lower = 1.0, ineqA.upper = 1.0)
                mat[j,] <- erg$solution
                obj[j] <- erg$objective
                tim[j] <- as.numeric(erg$elapsed)
            }

            L[[paste(w, collapse = "-")]] <- list(x = mat, obj = obj, time = tim)
            print(weightInd/nrow(grid))
        }
        L
    }
    names(xDispVarTest[[t]]) <- names(sDisp)
}

saveRDS(xDispVarTest, file = file.path(getwd(), "Optimization", paste0("EDispersionMinVaryingTest_", Sy

stopCluster(cl)
```

```
xDispVarTest <- readRDS(file.path(getwd(), "Optimization", "EDispersionMinVaryingTest_gloggest2017-08-2
```

determine optimal goal weights (optimal grid point)

We have the portfolio weights for each date (start) in "dat$datesTest$P1$'0.9-0.05-0.05'$x". We hold this for one period, therefore we calculate the portfolio return at each time step.

The datastructure is the following

dat -> datesTest -> P1 -> 0.9-0.05-0.05 -> x

We want the datastrucute

dat -> datesTest -> P1 -> 0.9-0.05-0.05 -> r, sd, sr, fweight

with

r: return (overall) sd: standard deviation (overall) sr: sharpe ratio fweight: mean of goal function

For the calculation of r, we procede as: $\mu = \mathrm{E[R]} = \sum_{t=1}^{T} \mathrm{R}_t$ and for sd: $\mathrm{sd(R)} = \sqrt{\mathrm{Var(R)}}$

In *ret* we have how much return was done at the current date (row), so we have to make a one period shift (hold portfolio up to next period and the return we make is given as the next stored return).

```
calcTestVar <- function(dat){
    res <- list()
```

```
    for(timeWindowName in names(dat)){
        timeWindow <- get(timeWindowName)
        retTimeWindow <- ret[timeWindow,]
        retTimeWindow <- retTimeWindow[-1,]
        colnames(retTimeWindow) <- colnames(ret)

        rf <- mean(retTimeWindow[,"BUND"])

        for(sentixGroup in names(dat[[timeWindowName]])){

            for(goalWeight in names(dat[[timeWindowName]][[sentixGroup]])){
                R <- rowSums(dat[[timeWindowName]][[sentixGroup]][[goalWeight]]$x * retTimeWindow)

                r <- mean(R)
                sd <- sd(R)

                anR <- (1+r)^52-1
                anSd <- sqrt((sd^2)*52)

                fweight = mean(dat[[timeWindowName]][[sentixGroup]][[goalWeight]]$obj)

                res[[timeWindowName]][[sentixGroup]][[goalWeight]] <- list(r = r, sd = sd, sr = r/sd,
                                                                          anR = anR, anSd = anSd, a
                                                                          fweight = fweight)
            }
        }
    }
    return(res)
}
```

```
xDispVarTestCalc <- calcTestVar(xDispVarTest)
```

now, determine optimal weight, for each *sentixGroup* and *timeWindow*, optimal meaning maximum sharpe ratio

Now, we also want to visualize the data. We use a ternary plot for this.

```
library(ggtern)
```

```
## --
## Consider donating at: http://ggtern.com
## Even small amounts (say $10-50) are very much appreciated!
## Remember to cite, run citation(package = 'ggtern') for further info.
## --

##
## Attaching package: 'ggtern'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, aes, annotate, calc_element, ggplot, ggplot_build,
##     ggplot_gtable, ggplotGrob, ggsave, layer_data, theme,
##     theme_bw, theme_classic, theme_dark, theme_gray, theme_light,
##     theme_linedraw, theme_minimal, theme_void
```

```
extractWeightsWithValue <- function(dat, value){
    ret <- list()
```

```
    for(timeWindowName in names(dat)){
        for(sentixGroup in names(dat[[timeWindowName]])){
            df <- data.frame(w = names(dat[[timeWindowName]][[sentixGroup]])[1], value = dat[[timeWindo
            df$w <- as.character(df$w)
            df$w1 <- as.numeric(unlist(strsplit(df$w, "-"))[1])
            df$w2 <- as.numeric(unlist(strsplit(df$w, "-"))[2])
            df$w3 <- as.numeric(unlist(strsplit(df$w, "-"))[3])

            for(weightsName in names(dat[[timeWindowName]][[sentixGroup]])[2:length(names(dat[[timeWind
                df <- rbind(df, c(weightsName, dat[[timeWindowName]][[sentixGroup]][[weightsName]][[valu
            }

            ret[[timeWindowName]][[sentixGroup]] <- data.frame(w1 = as.numeric(df[,"w1"]),
                                                               w2 = as.numeric(df[,"w2"]),
                                                               w3 = as.numeric(df[,"w3"]),
                                                               value = as.numeric(df[,"value"]
        }
    }
    return(ret)
}

srWeightsAn <- extractWeightsWithValue(xDispVarTestCalc, "anSR")
```

```
terntheme <- function(){
    theme_rgbg() +
        theme(legend.position = c(0, 1),
              legend.justification = c(0, 1),
              plot.margin=unit(c(0, 2,0, 2), "cm"),
              tern.panel.background = element_rect(fill = "lightskyblue1"))
}
```

note: “Calling ‘structure(NULL, *)’ is deprecated, as NULL cannot have attributes. Consider ‘structure(list(), *)’ instead.” is just a message, not an error. it is due to strict checks in the newer R-version https://stat.ethz.ch/pipermail/r-devel/2016-December/073554.html

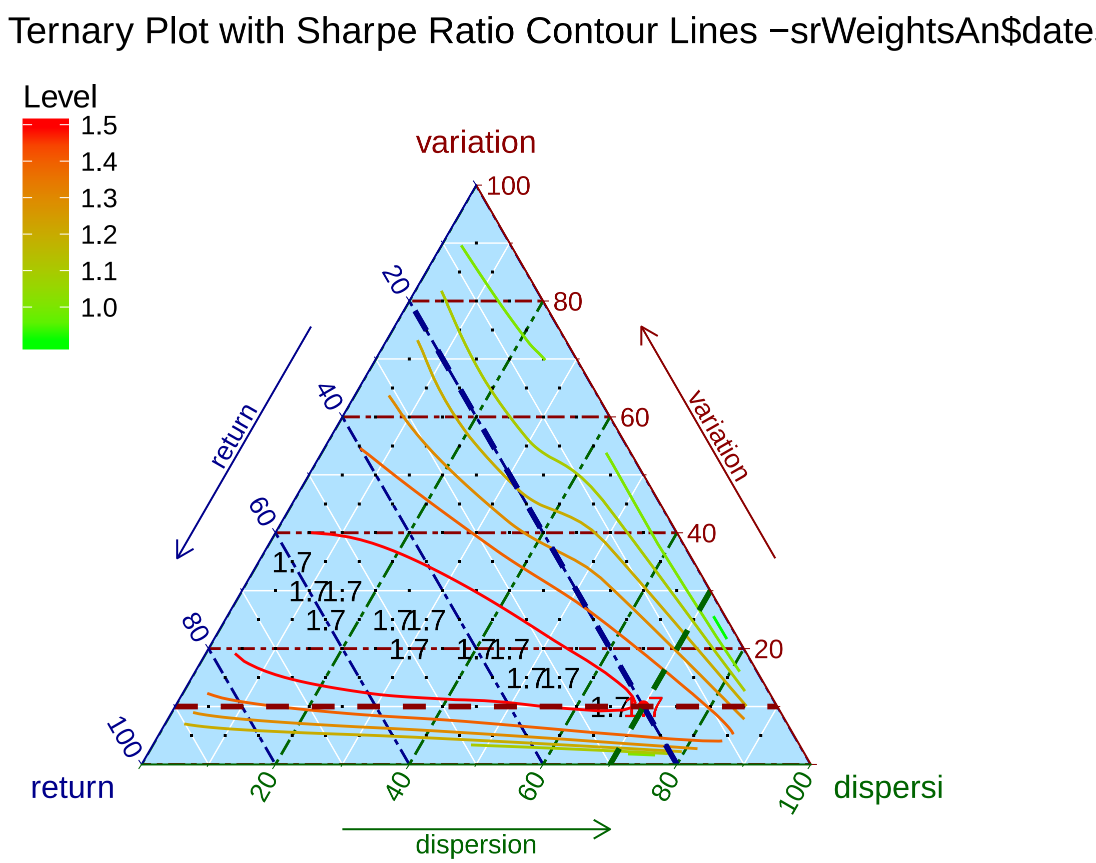
..level.. is calculated and then used inside the function

```
plotTernary <- function(dat){
    wmax <- dat[which.max(dat$value), c("w1", "w2", "w3", "value")]

    ggtern(dat, aes(x=w1, y=w2, z=w3, value=value)) +
        geom_point(shape=".")+
        geom_text(aes(x=w1, y=w2, label=round(value,1)), data = dat[dat$value>(dat[(dat$w1==wmax$w1) &
        geom_interpolate_tern(aes(value=value, color = ..level..)) +
        geom_point(aes(x=w1, y=w2), dat = wmax, color = "red") +
        geom_text(aes(x=w1, y=w2, label=round(dat[(dat$w1==wmax$w1) & (dat$w2==wmax$w2),"value"],1)), da
        terntheme() +
        Lline(Lintercept =  wmax$w1, colour = theme_rgbg()$tern.axis.line.L$colour, linetype = 2, lwd=1)
        Tline(Tintercept = wmax$w2, colour = theme_rgbg()$tern.axis.line.T$colour, linetype = 2, lwd=1)
        Rline(Rintercept = wmax$w3, color = theme_rgbg()$tern.axis.line.R$colour, linetype = 2, lwd=1) +
        scale_color_gradient(low = "green", high = "red") +
        labs(x = "return", y = "variation", z = "dispersion",
             title = paste0("Ternary Plot with Sharpe Ratio Contour Lines -", deparse(substitute(dat)),
             color = "Level")
}
```

```r
lateximport <- c(paste0("\\subsection{Ternary}"))

for(t in names(srWeightsAn$datesTest)){
    plot(plotTernary(srWeightsAn$datesTest[[t]]))

    title <- paste0("Ternary-Weights ",t, ".pdf")
    pdf(file.path(getwd(), "Plot", title), width = 10, height = 10)
    plot(plotTernary(srWeightsAn$datesTest[[t]]))
    dev.off()

    lateximport <- c(lateximport, paste0("\\includegraphics[height=0.45\\textheight]{",title,"}\\linebre
}
```

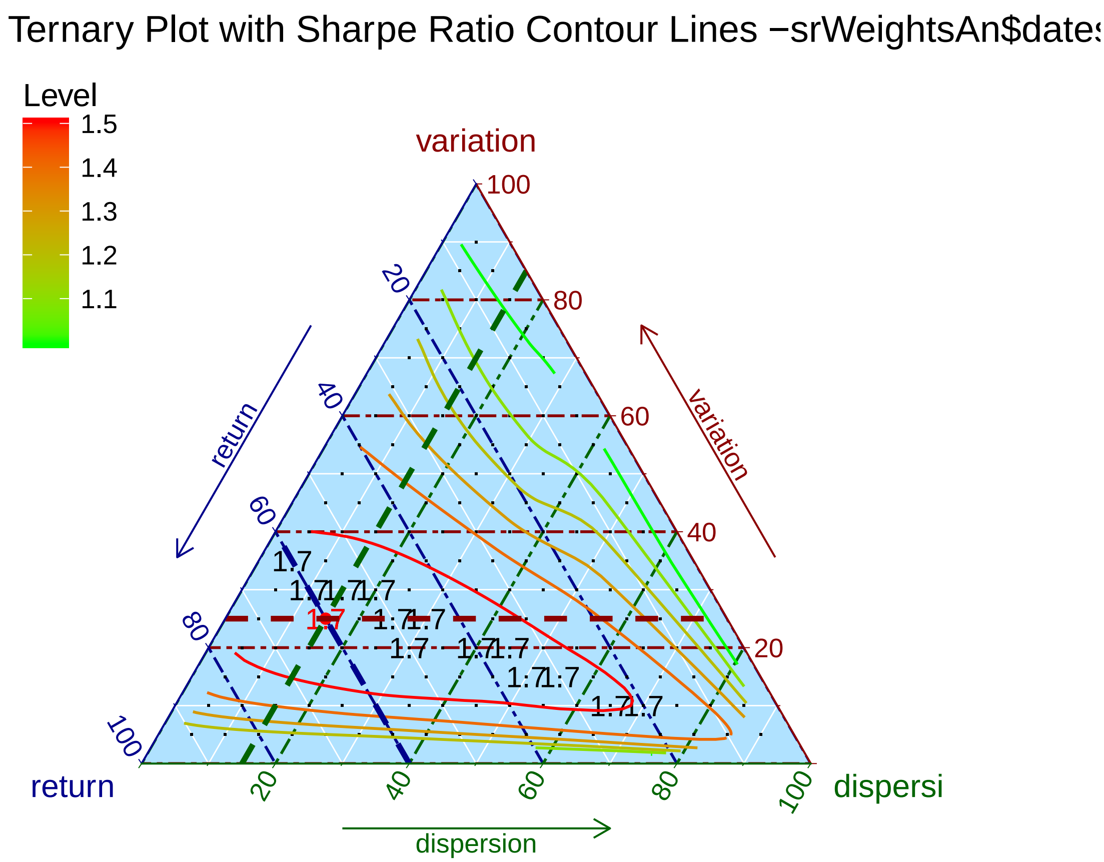
```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```



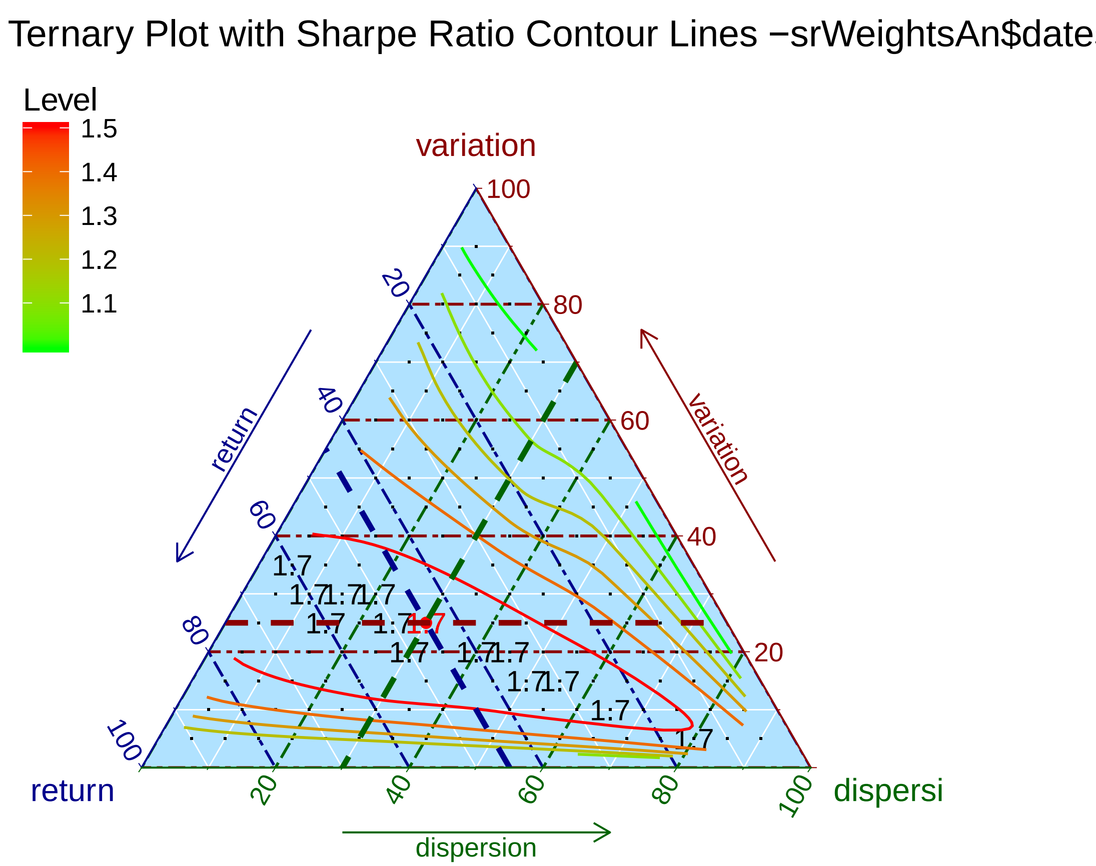

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```



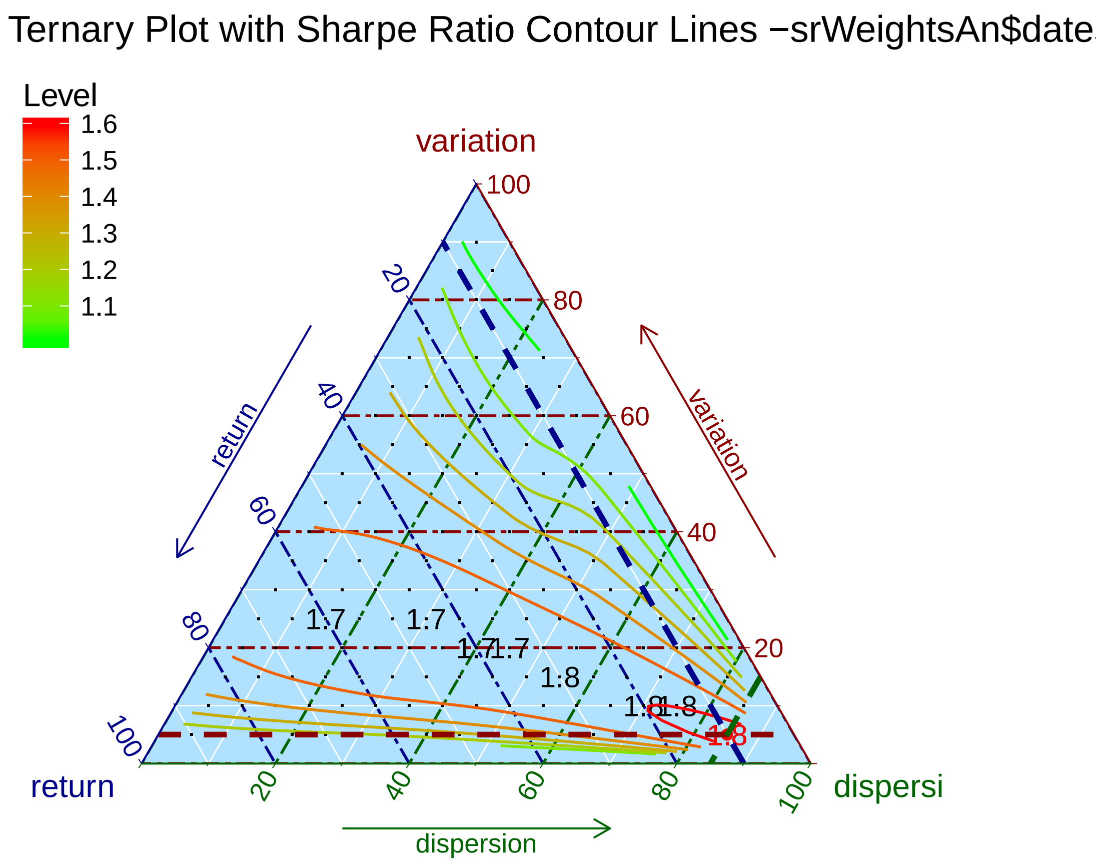

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```



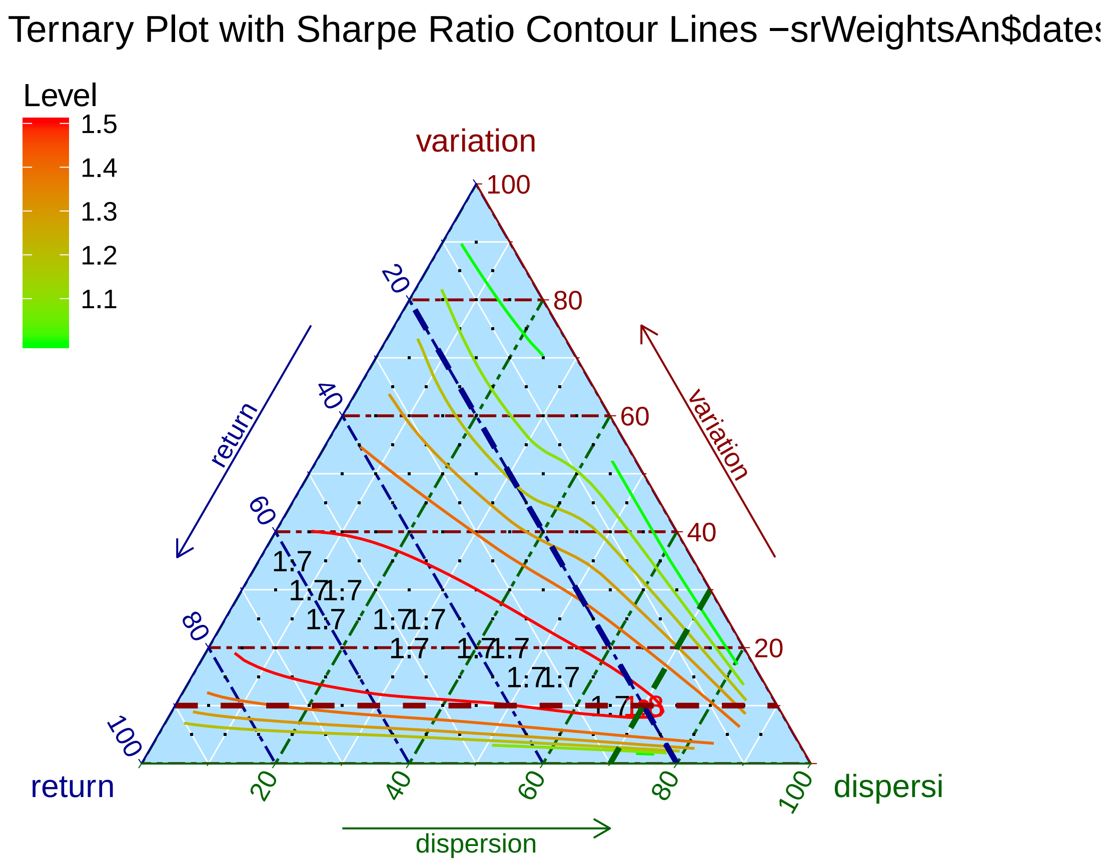

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```



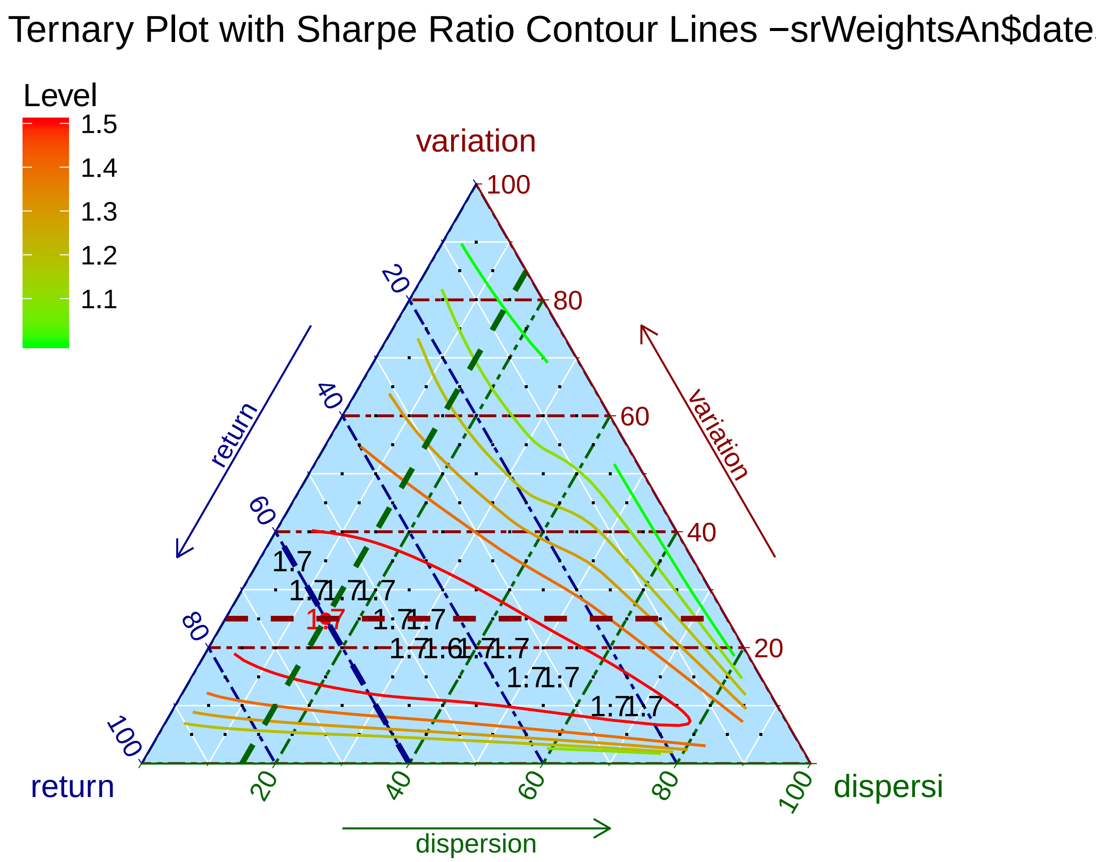

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```




```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```




```
fileConnection <- file(file.path(getwd(), "Plot", paste0("0Ternary.txt")))
writeLines(lateximport, fileConnection)
close(fileConnection)
```

now, determine optimal weight, for each *sentixGroup* and *timeWindow*, optimal meaning maximum sharpe ratio

```
wOptSRAn <- list()
for(timeWindowName in names(srWeightsAn)){
    for(sentixGroup in names(srWeightsAn[[timeWindowName]])){
        df <- srWeightsAn[[timeWindowName]][[sentixGroup]]

        wOptSRAn[[timeWindowName]][[sentixGroup]] <- df[which.max(df$value), c("w1", "w2", "w3")]
    }
}
wOptSRAn$datesTest
```

```
## $P1
##      w1  w2  w3
## 158 0.2 0.1 0.7
##
## $P6
##     w1   w2   w3
## 40 0.6 0.25 0.15
##
## $I1
##      w1   w2  w3
```

```
## 85 0.45 0.25 0.3
##
## $I6
##      w1   w2   w3
## 169 0.1 0.05 0.85
##
## $G1
##      w1  w2  w3
## 158 0.2 0.1 0.7
##
## $G6
##     w1   w2   w3
## 40 0.6 0.25 0.15
```

```
rm(xDispVarTest, xDispVarTestCalc, calcTestVar)
rm(plotTernary, terntheme, fileConnection, lateximport)
```

find optimal portfolio

dispersion direct min

Now, we want to determine the portfolio weights over the time Windows. We use the data structure:

time window -> dispersion (sentixGroup) -> weights of assets

NOTE: change the weights of the test data, if different test time windows are used to get weights for different eval time weights

We use a moving time window of k dates before the actual date to determine mean and variance and therefore to determine the portfolio. Furthermore, we just use the actual dispersion.

```
k <- 50

cores <- detectCores()

if(Sys.getenv("USERNAME") == "Stefan"){
    cl <- makeCluster(cores - 1)
} else if(Sys.getenv("USERNAME") == "gloggest"){
    cl <- makeCluster(cores) # use server fully
} else
    stop("Who are you???")


xDispVarEval <- list()

registerDoSNOW(cl)
for(t in datesEvalNames){
    # timeInd <- datesAll[which(datesAll == min(get(t)))-1] ## one day before start of time window ### 
    
    xDispVarEval[[t]] <- foreach(sentixGroup = names(sDisp), .export = c(datesEvalNames), .packages = c
        L <- list()


        w <- unlist(wOptSRAn$datesTest[[sentixGroup]])

        mat <- matrix(NA, nrow = (length(get(t))-1), ncol = ncol(ret))
```

```r
        colnames(mat) <- colnames(ret)
        rownames(mat) <- get(t)[1:(length(get(t))-1)]
        obj <- numeric(length(get(t))-1)
        tim <- numeric(length(get(t))-1)

        # first separate to then use the previous solution as starting point for next solution
        ### -------------------
        j <- 1
        tInd <- which(datesAll == get(t)[j])
        retOpt <- ret[(tInd-k+1):tInd,]
        anMuOpt <- (1+colMeans(retOpt))^52-1
        anCOpt <- cov(retOpt)*52
        anDOpt <-  as.numeric(sDisp[[sentixGroup]][tInd,-1])

        erg <- donlp2NLP(start = rep(1/ncol(retOpt), ncol(retOpt)), fun = hDispersionDirectMin,
                         par.lower = rep(0, ncol(retOpt)), ineqA = IneqA,
                         ineqA.lower = 1.0, ineqA.upper = 1.0)
        mat[1,] <- erg$solution
        obj[1] <- erg$objective
        tim[1] <- as.numeric(erg$elapsed)
        ### -------------------

        for(j in 2:(length(get(t))-1)){
            tInd <- which(datesAll == get(t)[j])
            retOpt <- ret[(tInd-k+1):tInd,]
            anMuOpt <- (1+colMeans(retOpt))^52-1
            anCOpt <- cov(retOpt)*52
            anDOpt <- as.numeric(sDisp[[sentixGroup]][tInd,-1])

            erg <- donlp2NLP(start = mat[j-1,], fun = hDispersionDirectMin,
                             par.lower = rep(0, ncol(retOpt)), ineqA = IneqA,
                             ineqA.lower = 1.0, ineqA.upper = 1.0)
            mat[j,] <- erg$solution
            obj[j] <- erg$objective
            tim[j] <- as.numeric(erg$elapsed)
        }

        list(x = mat, obj = obj, time = tim)
    }
    names(xDispVarEval[[t]]) <- names(sDisp)
}

stopCluster(cl)

# names(xDispVarTest) <- datesTestNames

saveRDS(xDispVarEval, file = file.path(getwd(), "Optimization", paste0("EDispersionMinVaryingEval_", Sy
```

```r
xDispVarEval <- readRDS(file.path(getwd(), "Optimization", "EDispersionMinVaryingEval_Stefan2017-08-29-
```

cleanup

```
rm(grid, IneqA)

detach("package:doSNOW", unload = T)
unloadNamespace("doParallel")
detach("package:parallel", unload = T)
```

```
## Warning: 'parallel' namespace cannot be unloaded:
##   namespace 'parallel' is imported by 'Rsolnp' so cannot be unloaded
```

```
detach("package:foreach", unload = T)

detach("package:ggtern", unload = T)
```

without sentiment (classic)

constant portfolio

We also do some classical portfolio optimization, namely

1.	tangency portfolio	fPortfolio	highest return/risk ratio on the efficient frontier (market portfolio)
2.	minimum variance	fPortfolio	portfolio with minimal risk on the efficient frontier
3.	rp	cccp	risk parity solution of long-only portfolio
4.	PGMV	FRAPO (Pfaff)	global minimum variance (via correlation)
5.	PMD	FRAPO (Pfaff)	most diversivied portfolio (long-only)
6.	ew	own	equal weight

safe results in *xClassicConst* in an anolous manner to above

time window -> portfolio optimizing -> weights of assets

Be aware that the portfolios work with time series and therefore some typecasting is necessary.

```
library(fPortfolio)
library(FRAPO)
```

```
xClassicConst <- list()

# convert rownames back to date format (character!)
t <- rownames(ret)
class(t) <- "Date"
rdatTimeSource <- timeSeries(ret, charvec = as.character(t))

# equal weights to start with (maybe)
ew <- rep(1/ncol(ret), ncol(ret))

for(t in datesEvalNames){
    timeInd <- datesAll[which(datesAll == min(get(t)))-1] ## one day before start of time window

    rdatTime <- window(rdatTimeSource, start = start(rdatTimeSource), end = timeInd) # note: first day

    ans <- tangencyPortfolio(rdatTime)
    xClassicConst[[t]][["tanPort"]] <- getWeights(ans)

    ans <- minvariancePortfolio(rdatTime)
```

```r
    xClassicConst[[t]][["mVaPort"]] <- getWeights(ans)

    C <- cov(rdatTime)
    ans <- rp(ew, C, ew, optctrl = ctrl(trace = FALSE))
    xClassicConst[[t]][["rp"]] <- c(getx(ans))

    ans <- PGMV(rdatTime, optctrl = ctrl(trace = FALSE))
    xClassicConst[[t]][["PGMV"]] <- Weights(ans) / 100

    ans <- PMD(rdatTime, optctrl = ctrl(trace = FALSE))
    xClassicConst[[t]][["PMD"]] <- Weights(ans) / 100

    xClassicConst[[t]][["ew"]] <- ew
}
```

```r
rm(rdatTime, rdatTimeSource, t, ew, ans)
```

different portfolio weights over time window

IDEA: look at portfolio-rollingPortfolios {fPortfolio}

manually rolling

safe results in *xClassicVar* in an anolous manner to above

time window -> portfolio (classic) -> weights of assets

NOTE: change the weights of the test data, if different test time windows are used to get weights for different eval time weights

We use a moving time window of k dates before the actual date to determine mean and variance and therefore to determine the portfolio. Furthermore, we just use the actual dispersion.

```r
k <- 50

xClassicVar <- list()

# convert rownames back to date format (character!)
t <- rownames(ret)
class(t) <- "Date"
rdatTimeSource <- timeSeries(ret, charvec = as.character(t))

# equal weights to start with (maybe)
ew <- rep(1/ncol(ret), ncol(ret))

for(timeWindowName in datesEvalNames){
    datesEvalNow <- get(timeWindowName)

    mat <- matrix(NA, nrow = (length(datesEvalNow)-1), ncol = ncol(rdatTimeSource))
    colnames(mat) <- colnames(ret)
    rownames(mat) <- datesEvalNow[1:(length(datesEvalNow)-1)]

    xClassicVar[[timeWindowName]][["tanPort"]]$x <- mat
    xClassicVar[[timeWindowName]][["mVaPort"]]$x <- mat
    xClassicVar[[timeWindowName]][["rp"]]$x <- mat
    xClassicVar[[timeWindowName]][["PGMV"]]$x <- mat
```

```
    xClassicVar[[timeWindowName]][["PMD"]]$x <- mat
    xClassicVar[[timeWindowName]][["ew"]]$x <- mat

    for(d in 1:(length(datesEvalNow)-1)){ # last date no portfolio weights

        timeEndInd <- which(datesAll == datesEvalNow[d]) ## one day before start of time window => NO, w
        timeEnd <- datesAll[timeEndInd]
        timeStart <- datesAll[timeEndInd-k+1]

        rdatTime <- timeSeries::window(rdatTimeSource, start = timeStart, end = timeEnd) # note: first

        ans <- tangencyPortfolio(rdatTime)
        xClassicVar[[timeWindowName]][["tanPort"]]$x[d,] <- getWeights(ans)

        ans <- minvariancePortfolio(rdatTime)
        xClassicVar[[timeWindowName]][["mVaPort"]]$x[d,] <- getWeights(ans)

        C <- cov(rdatTime)
        ans <- rp(ew, C, ew, optctrl = ctrl(trace = FALSE))
        xClassicVar[[timeWindowName]][["rp"]]$x[d,] <- c(getx(ans))

        ans <- PGMV(rdatTime, optctrl = ctrl(trace = FALSE))
        xClassicVar[[timeWindowName]][["PGMV"]]$x[d,] <- Weights(ans) / 100

        ans <- PMD(rdatTime, optctrl = ctrl(trace = FALSE))
        xClassicVar[[timeWindowName]][["PMD"]]$x[d,] <- Weights(ans) / 100

        xClassicVar[[timeWindowName]][["ew"]]$x[d,] <- ew
    }
}
```

```
rm(t, timeEnd, timeEndInd, timeStart, timeWindowName, k, ew, rdatTime, rdatTimeSource, ans)
```

different portfolio weights over time window, just risky assets

IDEA: look at portfolio-rollingPortfolios {fPortfolio}

manually rolling

safe results in *xClassicVar* in an anolous manner to above

time window -> portfolio (classic) -> weights of assets

NOTE: change the weights of the test data, if different test time windows are used to get weights for different eval time weights

We use a moving time window of k dates before the actual date to determine mean and variance and therefore to determine the portfolio. Furthermore, we just use the actual dispersion.

we exclude the risk free asset *BUND* of the analysis

TODO: risk free rate mit nuller in BUND

```
k <- 50

xClassicVarNoRf <- list()
```

```
# convert rownames back to date format (character!)
t <- rownames(ret)
class(t) <- "Date"
rdatTimeSource <- timeSeries(ret, charvec = as.character(t))

# equal weights to start with (maybe)
ew <- rep(1/(ncol(ret)-1), (ncol(ret)-1))

for(timeWindowName in datesEvalNames){
    datesEvalNow <- get(timeWindowName)

    mat <- matrix(NA, nrow = (length(datesEvalNow)-1), ncol = ncol(rdatTimeSource))
    colnames(mat) <- colnames(ret)[1:ncol(rdatTimeSource)]
    rownames(mat) <- datesEvalNow[1:(length(datesEvalNow)-1)]

    xClassicVarNoRf[[timeWindowName]][["tanPort"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["mVaPort"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["rp"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["PGMV"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["PMD"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["ew"]]$x <- mat

    for(d in 1:(length(datesEvalNow)-1)){ # last date no portfolio weights

        timeEndInd <- which(datesAll == datesEvalNow[d]) ## one day before start of time window => NO, 
        timeEnd <- datesAll[timeEndInd]
        timeStart <- datesAll[timeEndInd-k+1]

        rdatTime <- timeSeries::window(rdatTimeSource, start = timeStart, end = timeEnd) # note: first
        rf <- mean(rdatTime[,"BUND"])
        rdatTime <- rdatTime[,setdiff(names(rdatTime), "BUND")] # reduce to all but BUND

        portfolio <- portfolioSpec()
        setRiskFreeRate(portfolio) <- rf

        ans <- tangencyPortfolio(rdatTime, spec = portfolio)
        xClassicVarNoRf[[timeWindowName]][["tanPort"]]$x[d,] <- c(getWeights(ans), 0)

        ans <- minvariancePortfolio(rdatTime, spec = portfolio)
        xClassicVarNoRf[[timeWindowName]][["mVaPort"]]$x[d,] <- c(getWeights(ans), 0)

        C <- cov(rdatTime)
        ans <- rp(ew, C, ew, optctrl = ctrl(trace = FALSE))
        xClassicVarNoRf[[timeWindowName]][["rp"]]$x[d,] <- c(getx(ans), 0)

        ans <- PGMV(rdatTime, optctrl = ctrl(trace = FALSE))
        xClassicVarNoRf[[timeWindowName]][["PGMV"]]$x[d,] <- c(Weights(ans) / 100, 0)

        ans <- PMD(rdatTime, optctrl = ctrl(trace = FALSE))
        xClassicVarNoRf[[timeWindowName]][["PMD"]]$x[d,] <- c(Weights(ans) / 100, 0)

        xClassicVarNoRf[[timeWindowName]][["ew"]]$x[d,] <- c(ew, 0)
    }
```

```
}
```

```
rm(t, timeEnd, timeEndInd, timeStart, timeWindowName, k, ew, rdatTime, rdatTimeSource, ans)
```

detach

```
detach("package:FRAPO", unload = T)
detach("package:fPortfolio", unload = T)
detach("package:fAssets", unload = T)

unloadNamespace("fCopulae")
unloadNamespace("fMultivar")
detach("package:fBasics", unload = T) # need to unload "fCopulae" and "fMultivar" first, somehow "detac

detach("package:timeSeries", unload = T)
```

Visualization

Functions

Evaluation of Varying Portfolio

We want to visualize the evolvement of a portfolio over each time window.

Be aware of the index shifting: retPlot[j-1, i] take wealth of previous day retOverTime[j-1,] take return of today (j is one step ahead)

Remove numbering of x-axis by *xaxt='n'*.

Generate *retPortSentixVarying*, the returns of portfolios with varying portfolio weights using sentix as third factor with optimal weights. It has the following structure:

time window -> dispersion (sentixGroup) -> return of Portfolio, sharpe ratio

x: portfolio weights R: return of portfolio on each date r: mean return of portfolio over whole time window sd: standard deviation of return of portfolio over whole time window sr: sharpe ratio (weekly) anR: annualized return of portfolio over whole time window anSd: annualized standard deiation anSR: sharpe ratio (annual) fweight: mean of goal function value turnover: turnover of weights (how much of portfolio has to be changed) in percent

Calculation of turnover: NOT USEFUL (not comparable as portfolios evolve differently): we fix portfolio weights in t, hold these weights to t+1 (while portfolio raises to (1+ret)*price_t), and may change weightsin t+1 -> amount changed is (change in weights) * (value of index in t+1) USEFUL: we calculate the percentage points of weights that change in each time step. Divide by 2 as a percentage point is taken from one part of the portfolio and given to another part (so counted twice) -> get amount of portfolio that changes

Use an adoption of *calcTestVar()*

```
calcEvalVarClassic <- function(dat){
    res <- list()
    for(timeWindowName in names(dat)){
        timeWindow <- get(timeWindowName)
        retTimeWindow <- ret[timeWindow,]
        retTimeWindow <- retTimeWindow[-1,]
        colnames(retTimeWindow) <- colnames(ret)

        rf <- mean(retTimeWindow[,"BUND"])

        for(portfolioName in names(dat[[timeWindowName]])){


            R <- rowSums(dat[[timeWindowName]][[portfolioName]]$x * retTimeWindow)

            turnover <- c(0, rowSums(abs(diff(dat[[timeWindowName]][[portfolioName]]$x)))/2) # start of

            r <- mean(R)
            sd <- sd(R)

            anR <- (1+r)^52-1
            anSd <- sqrt((sd^2)*52)


            res[[timeWindowName]][[portfolioName]] <- list(x = dat[[timeWindowName]][[portfolioName]]$x
```

```
                                                                    R = R, r = r, sd = sd, sr = r/sd,
                                                                    anR = anR, anSd = anSd, anSR = anR/anSd,
                                                                    turnover = turnover)

        }
    }
    return(res)
}
```

difference in function is that fweight is not there for classic portfolios

```
calcEvalVarSentix <- function(dat){
    res <- calcEvalVarClassic(dat)
    for(timeWindowName in names(dat)){
        for(portfolioName in names(dat[[timeWindowName]])){
            fweight = mean(dat[[timeWindowName]][[portfolioName]]$obj)
            res[[timeWindowName]][[portfolioName]]$fweight <- fweight
        }
    }
    return(res)
}
```

plot performance

We now optimize the plotting for ggplot(). (DOESN'T WORK)

Therefore our dataframe to plot should have the following structure: date: Date value: worth of Portfolio portfolio: Portfolio (SentixGroup)

first in separate list, then in one dataframe

NOTE: returns occur one date later as stated here (in the data)

There has been an issue with *date*. It is getted as character and we need to transform it to integer and then back to date to store date as a numeric value (formatted as a date) and then used as x-axis

```
plotPortfolio <- function(data, timeWindowName){
    datWork <- data[[timeWindowName]]
    timeWindow <- get(timeWindowName)

    colBackground <- colsEvalDates[timeWindowName]

    retPlot <- data.frame(date = as.integer(as.Date(get(timeWindowName))))# date is read as character,
    class(retPlot$date) <- "Date"

    for(s in names(datWork)){
        ret <- cumprod(1+datWork[[s]]$R)
        retPlot[[s]] <- c(100, 100*ret)
    }


    ggplot(retPlot, aes(x=date))+
        geom_line(aes(y=retPlot[,2], color = colnames(retPlot)[2]))


    plotCommand <- paste0(text = "ggplot(retPlot, aes(x=date))+")
```

```
    for (i in 2:(ncol(retPlot)-1)){
        plotCommand <- paste0(plotCommand, "geom_line(aes(y=retPlot[,",i,"], color = colnames(retPlot)[
    }
    plotCommand <- paste0(plotCommand, "geom_line(aes(y=retPlot[,",ncol(retPlot),"], color = colnames(re

    eval(parse(text = plotCommand))+
        labs(title = paste("Time:", timeWindowName),
             y = "Value",
             x = "Date") +
        scale_color_discrete(name = "Index")+
        theme(panel.background = element_rect(fill = alpha(colBackground, 0.2)))
}
```

```
plotPortfolioComplete <- function(dat, fileName){
    lateximport <- c(paste0("\\subsection{",fileName,"}"))

    for(d in datesEvalNames){
        plotPortfolio(dat, d)

        title <- paste0(fileName, "-", d, ".pdf")
        pdf(file.path(getwd(), "Plot", title), width = 10, height = 4)
        plot(plotPortfolio(dat, d))
        dev.off()

        lateximport <- c(lateximport, paste0("\\includegraphics[width=\\textwidth]{",title,"}"))
    }

    fileConnection <- file(file.path(getwd(), "Plot", paste0("0",fileName,".txt")))
    writeLines(lateximport, fileConnection)
    close(fileConnection)
}
```

change of weights

```
plotWeightsLines <- function(datName, d, s){
    dat <- datName[[d]][[s]]$x
    dat <- as.data.frame(dat)
    dat$date <- as.Date(rownames(dat))
    plotCommand <- paste0("ggplot(dat, aes(x=date)) +")

    for(i in 1:(ncol(dat)-2)){
        plotCommand <- paste0(plotCommand, "geom_line(aes(y=dat[,",i,"], color = colnames(dat)[", i, "]
    }
    plotCommand <- paste0(plotCommand, "geom_line(aes(y=dat[,",ncol(dat)-1,"], color = colnames(dat)[",

    eval(parse(text = plotCommand))+
        labs(title = paste("Time:", d),
             subtitle = paste("Portfolio:", s),
             y = "Weight",
             x = "Date") +
        scale_color_discrete(name = "Index")
}
```

```
plotWeightsLinesComplete <- function(dat, fileName){
    lateximport <- c(paste0("\\subsection{",fileName,"}"))

    for(d in datesEvalNames){
        lateximport <- c(lateximport, paste0("\\subsubsection{", fileName, " - ", d, "}"))

        for(s in names(dat[[d]])){
            # plotWeightsLines(dat, d, s)

            title <- paste0(fileName, "-", d,"-", s, ".pdf")
            pdf(file.path(getwd(), "Plot", title), width = 10, height = 4)
            plot(plotWeightsLines(dat, d, s))
            dev.off()

            lateximport <- c(lateximport, paste0("\\includegraphics[width=\\textwidth]{",title,"}"))
        }
    }

    fileConnection <- file(file.path(getwd(), "Plot", paste0("0",fileName,".txt")))
    writeLines(lateximport, fileConnection)
    close(fileConnection)
}
```

TODO: change of weights with turnover

TODO: include turnover as bar plot with second y-axis to visualize how much a portfolio has to be changed.

```
plotWeightsLines <- function(datName, d, s){
    dat <- datName[[d]][[s]]$x
    dat <- as.data.frame(dat)
    dat$date <- as.Date(rownames(dat))
    dat$turnover <- datName[[d]][[s]]$turnover

    colBackground <- colsEvalDates[d]

    plotCommand <- paste0("ggplot(dat, aes(x=date)) +")
    for(i in 1:(ncol(dat)-3)){
        plotCommand <- paste0(plotCommand, "geom_line(aes(y=dat[,",i,"], color = colnames(dat)[", i, "]
    }
    plotCommand <- paste0(plotCommand, "geom_line(aes(y=dat[,",ncol(dat)-2,"], color = colnames(dat)[",

    eval(parse(text = plotCommand))+
        ylim(0, 1)+
        geom_bar(aes(y=dat$turnover, colour = "Turnover"), stat = "identity")+
        scale_y_continuous(sec.axis = sec_axis(~., name = "Turnover"))+
        labs(title = paste("Time:", d),
             subtitle = paste("Portfolio:", s),
             y = "Weight ",
             x = "Date") +
        scale_color_discrete(name = "Index") +
        theme(panel.background = element_rect(fill = alpha(colBackground, 0.2)))
}
```

```r
plotWeightsLinesComplete <- function(dat, fileName){
    lateximport <- c(paste0("\\subsection{",fileName,"}"))

    for(d in datesEvalNames){
        lateximport <- c(lateximport, paste0("\\subsubsection{", fileName, " - ", d, "}"))

        for(s in names(dat[[d]])){
            # plotWeightsLines(dat, d, s)

            title <- paste0(fileName, "-", d,"-", s, ".pdf")
            pdf(file.path(getwd(), "Plot", title), width = 10, height = 4)
            plot(plotWeightsLines(dat, d, s))
            dev.off()

            lateximport <- c(lateximport, paste0("\\includegraphics[width=\\textwidth]{",title,"}"))
        }
    }

    fileConnection <- file(file.path(getwd(), "Plot", paste0("0",fileName,".txt")))
    writeLines(lateximport, fileConnection)
    close(fileConnection)
}
```

summary statistics

print the summary (in matrix to pass it on to LaTeX-Table lateron)

```r
summaryClassic <- function(datName, d, roundTo = 2){
    dat <- datName[[d]]

    mat <- matrix(NA, nrow = 3, ncol = length(dat))
    rownames(mat) <- c("Mean Return (an)", "Volatility (an)", "Sharpe Ratio (an)")
    colnames(mat) <- names(dat)

    for(sInd in 1:length(dat)){
        mat[1,sInd] <- round(dat[[sInd]]$anR, roundTo)
        mat[2,sInd] <- round(dat[[sInd]]$anSd, roundTo)
        mat[3,sInd] <- round(dat[[sInd]]$anSR, roundTo)
    }
    return(mat)
}
```

```r
# install.packages("xtable")
library(xtable)
```

```
##
## Attaching package: 'xtable'

## The following object is masked from 'package:timeSeries':
##
##     align

## The following object is masked from 'package:timeDate':
##
##     align
```

```
summaryClassicComplete <- function(dat, fileName, roundTo = 2){
    lateximport <- c(paste0("\\subsection{",fileName,"}"))

    for(d in datesEvalNames){
        lateximport <- c(lateximport, paste0("\\subsubsection{", d, "}"))
        lateximport <- c(lateximport, print(xtable(summaryClassic(dat, d, roundTo))))
        print(summaryClassic(dat, d, roundTo))
    }

    lateximport <- c(lateximport, "\\clearpage")
    fileConnection <- file(file.path(getwd(), "Plot", paste0("0",fileName,".txt")))
    writeLines(lateximport, fileConnection)
    close(fileConnection)
}
```

whole analysis in one command

```
wholeAnalysis <- function(dat, fileName){
    retDat <- calcEvalVarClassic(dat)

    # weights
    plotWeightsLinesComplete(retDat, paste0("Weights-", fileName))

    # performance of portfolio
    plotPortfolioComplete(retDat, paste0("Performance-", fileName))

    # summary statistics
    summaryClassicComplete(retDat, paste0("Summary-", fileName))
}
```

Classic Optimization

Constant weights over time window

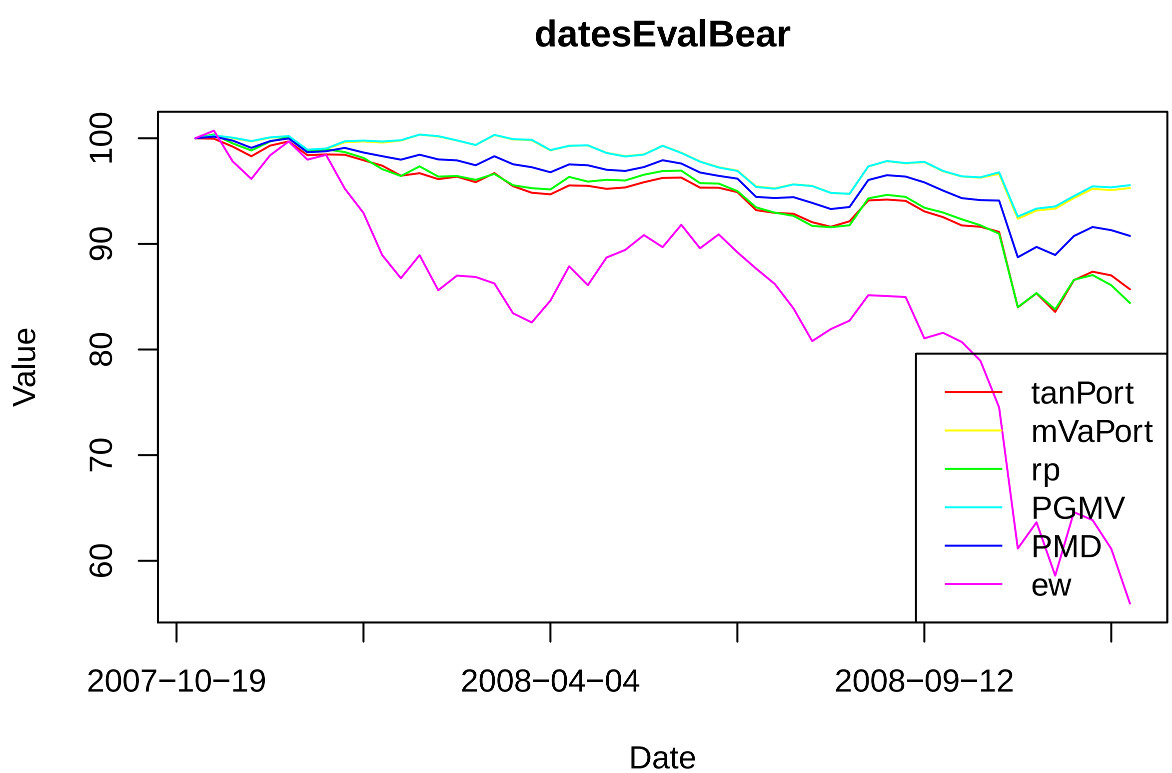
We want to visualize the evolvement of a portfolio over each time window.

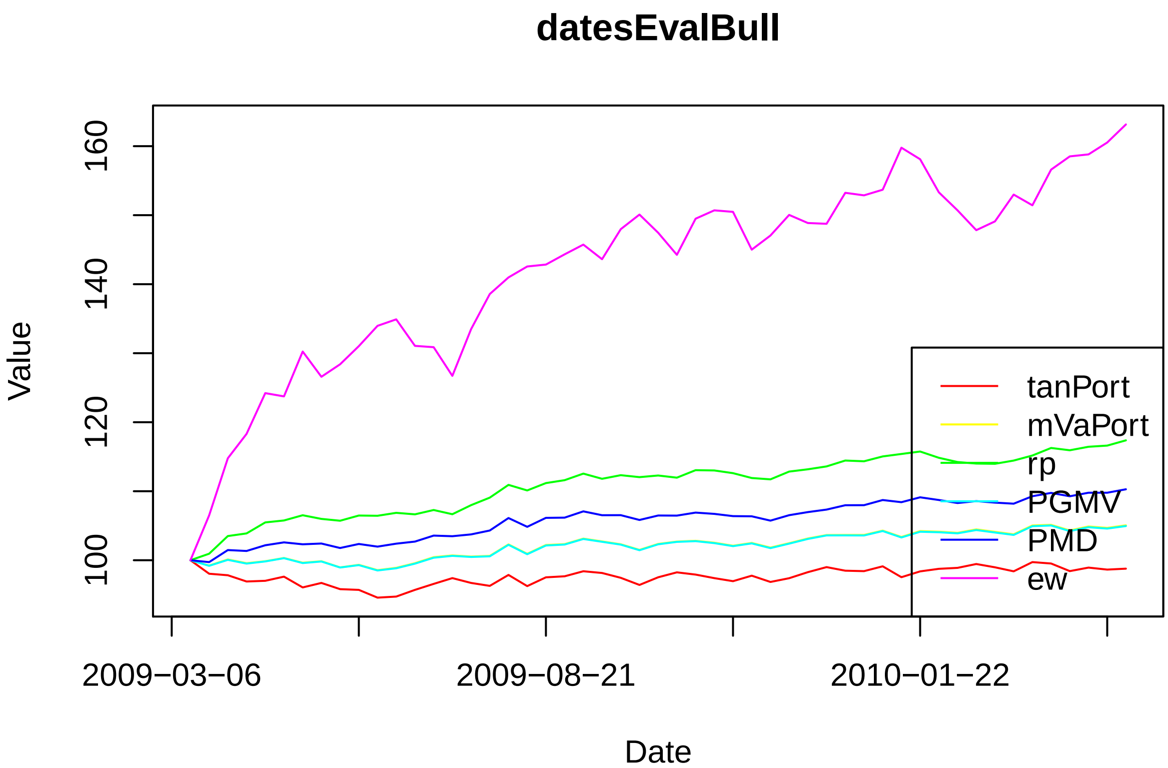
Be aware of the index shifting: retPlot[j-1, i] take wealth of previous day retOverTime[j-1,] take return of today (j is one step ahead)

Remove numbering of x-axis by *xaxt='n'*.

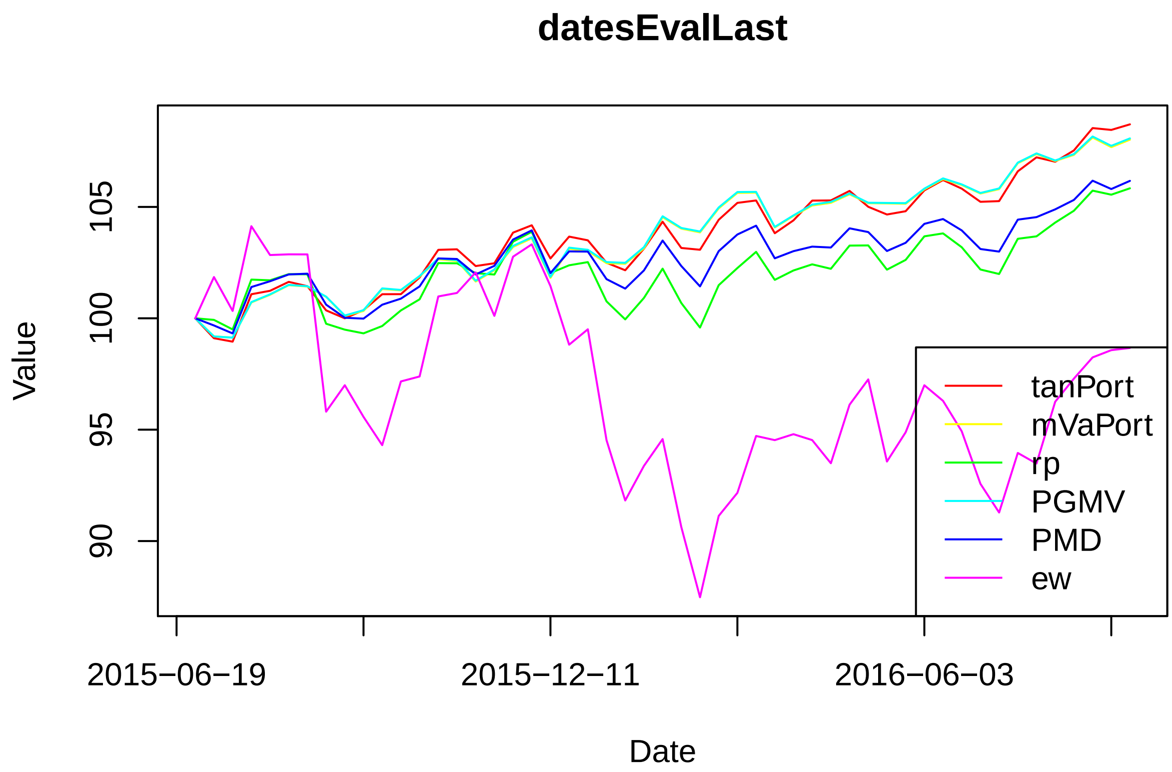
```
for(d in datesEvalNames){
    cols <- rainbow(length(xClassicConst[[d]]))
    retOverTime <- 1+ret[get(d),]
    retPlotDates <- get(d)
    retPlotDates <- c(datesAll[which(datesAll==min(retPlotDates))-1], retPlotDates)
    retPlot <- data.frame(Datum = retPlotDates)

    for(i in names(xClassicConst[[d]])){
        retPlot[1,i] <- 100
        for(j in 2:nrow(retPlot)){
            retPlot[j,i] <- retPlot[j-1,i]*crossprod(xClassicConst[[d]][[i]], retOverTime[j-1,])
```

```
        }
    }

    ylim = c(min(retPlot[,-1]), max(retPlot[,-1]))
    plot(retPlot[,2], type = "l", ylim = ylim, col = cols[1], main = d, xlab = "Date", ylab = "Value", :
    for(i in 3:ncol(retPlot)){
        par(new=T)
        plot(retPlot[,i], type = "l", ylim = ylim, axes = F, xlab = "", ylab = "", col = cols[i-1])
    }
    axis(1, at = c(0, 10, 20, 30, 40, 50), labels = retPlot[c(0, 10, 20, 30, 40, 50)+1,1])
    legend("bottomright", legend = names(xClassicConst[[d]]), col = cols, lty = 1)

    pdf(file.path(getwd(), "Plot", paste0("Performance-ClassicConst-", d, ".pdf")), width = 10, height :
    plot(retPlot[,2], type = "l", ylim = ylim, col = cols[1], main = d, xlab = "Date", ylab = "Value", :
    for(i in 3:ncol(retPlot)){
        par(new=T)
        plot(retPlot[,i], type = "l", ylim = ylim, axes = F, xlab = "", ylab = "", col = cols[i-1])
    }
    axis(1, at = c(0, 10, 20, 30, 40, 50), labels = retPlot[c(0, 10, 20, 30, 40, 50)+1,1])
    legend("bottomright", legend = names(xClassicConst[[d]]), col = cols, lty = 1)
    dev.off()
}
```





datesEvalBull
Value
160
140
120
100
2009−03−06
2009−08−21
2010−01−22
Date
tanPort
mVaPort
rp
PGMV
PMD
ew



Varying of portfolio weights

```
wholeAnalysis(xClassicVar, "Classic")
```

```
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:10:36 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrr}
##   \hline
##  & tanPort & mVaPort & rp & PGMV & PMD & ew \\
##   \hline
## Mean Return (an) & -0.34 & -0.08 & -0.17 & -0.08 & -0.12 & -0.44 \\
##   Volatility (an) & 0.29 & 0.08 & 0.11 & 0.08 & 0.09 & 0.29 \\
##   Sharpe Ratio (an) & -1.19 & -1.00 & -1.54 & -0.99 & -1.37 & -1.51 \\
##    \hline
## \end{tabular}
## \end{table}
##                    tanPort mVaPort    rp  PGMV   PMD    ew
## Mean Return (an)    -0.34   -0.08 -0.17 -0.08 -0.12 -0.44
## Volatility (an)      0.29    0.08  0.11  0.08  0.09  0.29
## Sharpe Ratio (an)   -1.19   -1.00 -1.54 -0.99 -1.37 -1.51
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:10:37 2017
## \begin{table}[ht]
```

```
## \centering
## \begin{tabular}{rrrrrrr}
##   \hline
##  & tanPort & mVaPort & rp & PGMV & PMD & ew \\
##   \hline
## Mean Return (an) & 0.03 & 0.07 & 0.16 & 0.07 & 0.11 & 0.60 \\
##   Volatility (an) & 0.06 & 0.05 & 0.04 & 0.05 & 0.04 & 0.18 \\
##   Sharpe Ratio (an) & 0.61 & 1.48 & 3.50 & 1.51 & 2.75 & 3.35 \\
##    \hline
## \end{tabular}
## \end{table}
##                   tanPort mVaPort   rp PGMV  PMD   ew
## Mean Return (an)     0.03    0.07 0.16 0.07 0.11 0.60
## Volatility (an)      0.06    0.05 0.04 0.05 0.04 0.18
## Sharpe Ratio (an)    0.61    1.48 3.50 1.51 2.75 3.35
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:10:37 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrr}
##   \hline
##  & tanPort & mVaPort & rp & PGMV & PMD & ew \\
##   \hline
## Mean Return (an) & 0.01 & 0.10 & 0.04 & 0.10 & 0.07 & -0.02 \\
##   Volatility (an) & 0.08 & 0.05 & 0.09 & 0.05 & 0.06 & 0.17 \\
##   Sharpe Ratio (an) & 0.15 & 2.10 & 0.42 & 2.07 & 1.04 & -0.11 \\
##    \hline
## \end{tabular}
## \end{table}
##                   tanPort mVaPort   rp PGMV  PMD    ew
## Mean Return (an)     0.01    0.10 0.04 0.10 0.07 -0.02
## Volatility (an)      0.08    0.05 0.09 0.05 0.06  0.17
## Sharpe Ratio (an)    0.15    2.10 0.42 2.07 1.04 -0.11
```

Varying of portfolio weights no risk free asset

```r
wholeAnalysis(xClassicVarNoRf, "Classic-No-Risk-Free")
```

```
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:11:01 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrr}
##   \hline
##  & tanPort & mVaPort & rp & PGMV & PMD & ew \\
##   \hline
## Mean Return (an) & -0.61 & -0.52 & -0.50 & -0.52 & -0.52 & -0.50 \\
##   Volatility (an) & 0.41 & 0.34 & 0.34 & 0.34 & 0.35 & 0.34 \\
##   Sharpe Ratio (an) & -1.50 & -1.52 & -1.45 & -1.52 & -1.50 & -1.44 \\
##    \hline
## \end{tabular}
## \end{table}
##                   tanPort mVaPort    rp  PGMV   PMD    ew
```

```
## Mean Return (an)    -0.61   -0.52 -0.50 -0.52 -0.52 -0.50
## Volatility (an)      0.41    0.34  0.34  0.34  0.35  0.34
## Sharpe Ratio (an)   -1.50   -1.52 -1.45 -1.52 -1.50 -1.44
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:11:01 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrr}
##   \hline
##  & tanPort & mVaPort & rp & PGMV & PMD & ew \\
##   \hline
## Mean Return (an) & 0.84 & 0.80 & 0.72 & 0.80 & 0.75 & 0.73 \\
##   Volatility (an) & 0.23 & 0.20 & 0.21 & 0.20 & 0.21 & 0.21 \\
##   Sharpe Ratio (an) & 3.60 & 3.94 & 3.41 & 3.93 & 3.56 & 3.38 \\
##    \hline
## \end{tabular}
## \end{table}
##                   tanPort mVaPort   rp PGMV  PMD   ew
## Mean Return (an)     0.84    0.80 0.72 0.80 0.75 0.73
## Volatility (an)      0.23    0.20 0.21 0.20 0.21 0.21
## Sharpe Ratio (an)    3.60    3.94 3.41 3.93 3.56 3.38
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:11:01 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrr}
##   \hline
##  & tanPort & mVaPort & rp & PGMV & PMD & ew \\
##   \hline
## Mean Return (an) & -0.03 & 0.01 & -0.03 & 0.01 & -0.05 & -0.04 \\
##   Volatility (an) & 0.21 & 0.15 & 0.19 & 0.15 & 0.20 & 0.20 \\
##   Sharpe Ratio (an) & -0.16 & 0.08 & -0.17 & 0.08 & -0.25 & -0.20 \\
##    \hline
## \end{tabular}
## \end{table}
##                   tanPort mVaPort    rp PGMV   PMD    ew
## Mean Return (an)    -0.03    0.01 -0.03 0.01 -0.05 -0.04
## Volatility (an)      0.21    0.15  0.19 0.15  0.20  0.20
## Sharpe Ratio (an)   -0.16    0.08 -0.17 0.08 -0.25 -0.20
```

Sentix Optimization

```
wholeAnalysis(xDispVarEval, "Sentix")
```

```
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:11:26 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrr}
##   \hline
##  & P1 & P6 & I1 & I6 & G1 & G6 \\
##   \hline
## Mean Return (an) & 0.05 & 0.07 & 0.04 & 0.06 & 0.05 & 0.07 \\
```

```
##   Volatility (an) & 0.07 & 0.07 & 0.06 & 0.07 & 0.07 & 0.07 \\
##   Sharpe Ratio (an) & 0.79 & 0.89 & 0.59 & 0.84 & 0.78 & 0.90 \\
##    \hline
## \end{tabular}
## \end{table}
##                     P1   P6   I1   I6   G1   G6
## Mean Return (an)  0.05 0.07 0.04 0.06 0.05 0.07
## Volatility (an)   0.07 0.07 0.06 0.07 0.07 0.07
## Sharpe Ratio (an) 0.79 0.89 0.59 0.84 0.78 0.90
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:11:26 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrr}
##   \hline
##  & P1 & P6 & I1 & I6 & G1 & G6 \\
##   \hline
## Mean Return (an) & 0.11 & 0.16 & 0.11 & 0.09 & 0.11 & 0.16 \\
##   Volatility (an) & 0.06 & 0.10 & 0.05 & 0.05 & 0.06 & 0.10 \\
##   Sharpe Ratio (an) & 1.89 & 1.59 & 2.01 & 1.71 & 1.85 & 1.59 \\
##    \hline
## \end{tabular}
## \end{table}
##                     P1   P6   I1   I6   G1   G6
## Mean Return (an)  0.11 0.16 0.11 0.09 0.11 0.16
## Volatility (an)   0.06 0.10 0.05 0.05 0.06 0.10
## Sharpe Ratio (an) 1.89 1.59 2.01 1.71 1.85 1.59
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:11:26 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrr}
##   \hline
##  & P1 & P6 & I1 & I6 & G1 & G6 \\
##   \hline
## Mean Return (an) & 0.12 & 0.13 & 0.12 & 0.12 & 0.12 & 0.13 \\
##   Volatility (an) & 0.06 & 0.11 & 0.06 & 0.06 & 0.06 & 0.11 \\
##   Sharpe Ratio (an) & 1.84 & 1.25 & 1.81 & 2.06 & 1.83 & 1.25 \\
##    \hline
## \end{tabular}
## \end{table}
##                     P1   P6   I1   I6   G1   G6
## Mean Return (an)  0.12 0.13 0.12 0.12 0.12 0.13
## Volatility (an)   0.06 0.11 0.06 0.06 0.06 0.11
## Sharpe Ratio (an) 1.84 1.25 1.81 2.06 1.83 1.25
```

All together

sentix with classic portfolio with varying weights

Performance

```
retPortClassicVarying <- calcEvalVarClassic(xClassicVar)
retPortSentixVarying <- calcEvalVarClassic(xDispVarEval)

retAllVarying <- retPortClassicVarying
for(timeWindowName in names(retAllVarying)){
    retAllVarying[[timeWindowName]] <- append(retAllVarying[[timeWindowName]], retPortSentixVarying[[ti
}
```

```
plotPortfolioComplete(retAllVarying, "Performance-All")
```

Summary Statistics

```
summaryClassicComplete(retAllVarying, "SummaryAll")
```

```
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:11:32 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrrrrrrrr}
##   \hline
##  & tanPort & mVaPort & rp & PGMV & PMD & ew & P1 & P6 & I1 & I6 & G1 & G6 \\
##   \hline
## Mean Return (an) & -0.34 & -0.08 & -0.17 & -0.08 & -0.12 & -0.44 & 0.05 & 0.07 & 0.04 & 0.06 & 0.05 &
##   Volatility (an) & 0.29 & 0.08 & 0.11 & 0.08 & 0.09 & 0.29 & 0.07 & 0.07 & 0.06 & 0.07 & 0.07 & 0.0
##   Sharpe Ratio (an) & -1.19 & -1.00 & -1.54 & -0.99 & -1.37 & -1.51 & 0.79 & 0.89 & 0.59 & 0.84 & 0.
##    \hline
## \end{tabular}
## \end{table}
##                    tanPort mVaPort    rp  PGMV   PMD    ew   P1   P6   I1
## Mean Return (an)     -0.34   -0.08 -0.17 -0.08 -0.12 -0.44 0.05 0.07 0.04
## Volatility (an)       0.29    0.08  0.11  0.08  0.09  0.29 0.07 0.07 0.06
## Sharpe Ratio (an)    -1.19   -1.00 -1.54 -0.99 -1.37 -1.51 0.79 0.89 0.59
##                      I6   G1   G6
## Mean Return (an)   0.06 0.05 0.07
## Volatility (an)    0.07 0.07 0.07
## Sharpe Ratio (an)  0.84 0.78 0.90
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:11:32 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrrrrrrrr}
##   \hline
##  & tanPort & mVaPort & rp & PGMV & PMD & ew & P1 & P6 & I1 & I6 & G1 & G6 \\
##   \hline
## Mean Return (an) & 0.03 & 0.07 & 0.16 & 0.07 & 0.11 & 0.60 & 0.11 & 0.16 & 0.11 & 0.09 & 0.11 & 0.16
##   Volatility (an) & 0.06 & 0.05 & 0.04 & 0.05 & 0.04 & 0.18 & 0.06 & 0.10 & 0.05 & 0.05 & 0.06 & 0.1
##   Sharpe Ratio (an) & 0.61 & 1.48 & 3.50 & 1.51 & 2.75 & 3.35 & 1.89 & 1.59 & 2.01 & 1.71 & 1.85 & 1
##    \hline
## \end{tabular}
## \end{table}
##                    tanPort mVaPort   rp PGMV  PMD   ew   P1   P6   I1   I6
```

```
## Mean Return (an)     0.03    0.07 0.16 0.07 0.11 0.60 0.11 0.16 0.11 0.09
## Volatility (an)      0.06    0.05 0.04 0.05 0.04 0.18 0.06 0.10 0.05 0.05
## Sharpe Ratio (an)    0.61    1.48 3.50 1.51 2.75 3.35 1.89 1.59 2.01 1.71
##                     G1   G6
## Mean Return (an)  0.11 0.16
## Volatility (an)   0.06 0.10
## Sharpe Ratio (an) 1.85 1.59
## % latex table generated in R 3.4.1 by xtable 1.8-2 package
## % Sun Sep 03 18:11:32 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrrrrrrrr}
##   \hline
##  & tanPort & mVaPort & rp & PGMV & PMD & ew & P1 & P6 & I1 & I6 & G1 & G6 \\
##   \hline
## Mean Return (an) & 0.01 & 0.10 & 0.04 & 0.10 & 0.07 & -0.02 & 0.12 & 0.13 & 0.12 & 0.12 & 0.12 & 0.13
##   Volatility (an) & 0.08 & 0.05 & 0.09 & 0.05 & 0.06 & 0.17 & 0.06 & 0.11 & 0.06 & 0.06 & 0.06 & 0.11
##   Sharpe Ratio (an) & 0.15 & 2.10 & 0.42 & 2.07 & 1.04 & -0.11 & 1.84 & 1.25 & 1.81 & 2.06 & 1.83 & 1
##    \hline
## \end{tabular}
## \end{table}
##                   tanPort mVaPort   rp PGMV  PMD    ew   P1   P6   I1   I6
## Mean Return (an)     0.01    0.10 0.04 0.10 0.07 -0.02 0.12 0.13 0.12 0.12
## Volatility (an)      0.08    0.05 0.09 0.05 0.06  0.17 0.06 0.11 0.06 0.06
## Sharpe Ratio (an)    0.15    2.10 0.42 2.07 1.04 -0.11 1.84 1.25 1.81 2.06
##                     G1   G6
## Mean Return (an)  0.12 0.13
## Volatility (an)   0.06 0.11
## Sharpe Ratio (an) 1.83 1.25
```

```r
rm(retPortClassicVarying, retPortSentixVarying, retAllVarying)
```

cleanup

```r
rm(calcEvalVarClassic, calcEvalVarSentix, plotPortfolio, plotPortfolioComplete, plotWeightsLines, plotWe

detach("package:xtable", unload = T)
```