人工神經網絡 @ 銀聯強積金 - MPF-ANN in BCT MPF Scheme

Portfolio management using Artifical Neural Network (ANN) in BCT MPF

銀聯強積金應用人工類神經網絡進行資產調配

Disclamier of warranties

1. The author (jchan-gi) expressly stated that nothing my repositories and webpages constitutes any advices or recommendation on investing or allocating assets on any stock or financial products.

2. No content in my study have been prepared with respect to specific personal need and investment objects of individuals. All materials reveal to the public are applicable to the author (jchan-gi) only.
3. All investors shall obtain kind advices from professional financial consultants before making any decisions. Professional financial consultants should recommend products or decisions that suit your needs and objectives.

4. The author is not licensed nor professional in the field hence this studies are not professional advice and may not be suitable for anyone except myself.

Last update: 2019/12/26.

Changelog

2019-12-26 v1 1. Using Temporal Convolution Network (TCN) instead of LSTM in future development. 2. Training and Inference has mitigated to Python file. 3. Rolling window have been adopted in training.

1. 以卷積時序網絡取代長短期記憶模型。
2. 訓練及推論已改在 Python 上進行。
3. 訓練數據改以滾動窗口方式建立。

WARNING: Please understand nature of ANN in this section before applying MPF-ANN:

警告：在應用 MPF-ANN 之前，請先於此節了解人工類神經網絡的特性

This script relies on artifical neural network (ANN) model for MPF fund allocation.
ANN is stochastic in nature. In default mode, It produces different results based on the pseudo-random seed (system time).
The mean of 20 portfolios with defined seed have been generated in order to generate replicable result. An trustworthy statistics, deflated sharpe ratio, is used for estimating overfitting of the model. The result of deflated sharpe ration reflectes the percentage of a model have position return due to less overfitting.

The model also perform walk-forward validation. A model will be created for exactly next re-balance only based on most updated data for that moment.

There is no other validation or assessment on the data.
Two validation shall give reasonable assessment on the return estimation.
However, please be reminded that backtesting is no way to be accurate in predicting future performance.

MPF-ANN 根據人工類神經網絡的預測回報作強積金基金調動。
然而，人工類神經網絡是一隨機過程。在預設的情況下，人工類神經網絡會根據系統時間設定種子，因此每次的執行結果也不盡相同。
為提高結果的可複製性，本程式以隨機方式抽取了 20 個種子以重複生成模型及計算平均值。
平均值模型將用以計算學術論文提出的過擬合夏普比率 (意譯自 deflated sharpe ratio) 來測定人工類神經網絡模型過擬合 (overfitting) 的程度。過擬合夏普比率數值代表了模型因沒有過度擬合而將有正回報的機率。
此外，此模式亦使用了前移式回測 (walk-forward validation)。每次資產調配都會基於擁有當時而言的最新資料而建立的人工類神經網結模型。

此模型只建基於上述兩項的檢測。
上述的檢測應可視為 MPF-ANN 所計算的回測結果可靠程度的參考指標。
然而，請切記過往表現不代表將來表現。

Background Introduction

背景簡介

Mandatory Provident Fund (MPF) is one of the legal pensions in Hongkong.
All Employers and some employees are required to pay at least 5% of monthly salary to MPF services provider.
MPF providers are required to provide approved funds for employees to invest, while the team could earn management fees.

However, the average annualized return of MPF is 3.1% only in from 2000-2016 (Mandatory Provident Fund Schemes Authority, 2016).
Most Hongkong employees feels they are forced to give their salary to MPF fund manager.
Indeed, the reasons of low return may due to inactive management by employees.
In this example, we will explore artifical neural network (ANN) to rise the annualized return in MPF-ANN.

在香港，強積性公積金 (強積金, Mandatory Provident Fund, MPF) 是其中一種法定退休金。
僱主及部份僱員須扣除 5% 薪金供強積金營運商。強積金營運商將每月收取管理費，並提供認可基金供僱員作投資儲蓄。但是，強積金的平均回報僅 3.1%，因而被批評回報有三份之一被基金經理蠶食。誠言，每位僱員如以主動方式管理強積金，有助提升回報而減少收費的影響。這文記錄了作者以人工類神經網絡方式提升強積金回報的探索 (即 MPF-ANN)。

BCT MPF Portfolio Generation using Temporal Convolution Network (TCN) & Relative Strength Index

銀聯積金之選基金分配的建立

作者嘗試以卷積時序網絡 (TCN, 一種新型人工類神經網絡結構) 及相對強弱指數分配強積金成份基金。TCN 的預測只考慮了基金的趨勢。相對強弱指數則可懲罰超買的強積金成份基金。

在考慮最近三至六個月回報後，程式將自動決定以 TCN 推測佈置戰術性資產部署 (tactical asset allocation, TAA) 、TCN 結果配合債券及現金的策略性資產部署 (Strategic asset allocation) 或進入避險模式。

When to manage my MPF?

何時管理及調配強積金?

作者以每月最後一日取得以前的基金價格計算每月回報及相對強弱指數。然後於同日重新配置強積金成分基金。然而，強積金大部分均有 T+2 的時延，因此實際操作將有時延誤差。

MPF-ANN Walk-forwark validation results

MPF-ANN 前移式回測結果

第一期強積金供款 First MPF Installment: 年率化回報 Annualized Return: ~10.29%
算術年均回報 Mean Annual Return: ~9.93%
年率化標準誤差 Annualized Standard Deviation: ~8.04% (StdDev(monthly return) * sqrt(12))
夏普比率 Sharpe Ratio (Mean Annual Return): 10.29%/8.04% = 1.2797
索丁諾比率 Sortino Ratio: 0.7462 (MAR = 0%)
預期損失 Expected Shortfall: 4.14% loss (0% Risk-free rate, 95% C.I.)
過擬合夏普比率 Deflated Sharpe Ratio (p-value): >99.9999% (rounded to 1)

真實強積金 (月供) Monthly MPF installment (或有誤差 maybe slightly differs):
總供款 Total contribution: 337500
最新結餘 Latest asset value: 871187.4
算術年均回報 Mean annual return: 8.70%
內部回報率 Internal Rate of Return (IRR): 8.87%
年率化標準誤差 Annualized Standard Deviation: 8.01%
夏普比率 Sharpe Ratio: 1.0855
索丁諾比率 Sortino Ratio: 0.6735 (MAR = 0%)
預期損失 Expected Shortfall: 4.21% loss (0% Risk-free rate, 95% C.I.)

Package Preparation

1. Install necessary packages

```
r = getOption("repos")
r["CRAN"] = "https://mran.revolutionanalytics.com/snapshot/2019-11-01"
options(repos = r)
install.packages("zoo")
install.packages("xts")
install.packages("fBasics")
install.packages("quantmod")
install.packages("PerformanceAnalytics")
install.packages("feather")
install.packages("dplyr")
install.packages("doParallel")
```

2. Now load necessary packages.

```
library("dplyr")
library("zoo")
library("xts")
library("fBasics")
library("quantmod")
```

```
library("PerformanceAnalytics")
library("feather")

library("doParallel")
```

Load Prices and Calculate Return

0. Parameters

```
top <- 2
RSI_Overbuy <- 0.85
RSI_Period <- 18
Min_NMMA <- 1e-6
```

1. Load the price into zoo format

```
MPF.BCT <-
  as.xts(
    read.zoo(
      "F:/OneDrive/MPF/BCT/BCT.csv",
      format = "%Y/%m/%d",
      header = TRUE,
      read = read.csv,
      na.strings = "0"
    )
  )
daily <- index(MPF.BCT)
#monthly.old <- index(MPF.BCT.returns)
#MPF.BCT.w.all.backup <- MPF.BCT.w.all
```

2. Calculate Relative Strength Index (RSI)

3. Calculate Returns

```
MPF.BCT.AE <- monthlyReturn(as.xts(MPF.BCT$AE), type = "log")
MPF.BCT.CA <- monthlyReturn(as.xts(MPF.BCT$CA), type = "log")
MPF.BCT.CEHK <- monthlyReturn(as.xts(MPF.BCT$CEHK), type = "log")
MPF.BCT.CT <- monthlyReturn(as.xts(MPF.BCT$CT), type = "log")
MPF.BCT.EU <- monthlyReturn(as.xts(MPF.BCT$EU), type = "log")
MPF.BCT.E3 <- monthlyReturn(as.xts(MPF.BCT$E30), type = "log")
MPF.BCT.E5 <- monthlyReturn(as.xts(MPF.BCT$E50), type = "log")
MPF.BCT.E7 <- monthlyReturn(as.xts(MPF.BCT$E70), type = "log")
MPF.BCT.E9 <- monthlyReturn(as.xts(MPF.BCT$E90), type = "log")
MPF.BCT.FM <- monthlyReturn(as.xts(MPF.BCT$FM), type = "log")

MPF.BCT.GB <- monthlyReturn(as.xts(MPF.BCT$GB), type = "log")
MPF.BCT.GE <- monthlyReturn(as.xts(MPF.BCT$GE), type = "log")
MPF.BCT.GT <- monthlyReturn(as.xts(MPF.BCT$GT), type = "log")
MPF.BCT.HKB <- monthlyReturn(as.xts(MPF.BCT$HKB), type = "log")
MPF.BCT.HSIT <- monthlyReturn(as.xts(MPF.BCT$HSIT), type = "log")
MPF.BCT.MPFC <- monthlyReturn(as.xts(MPF.BCT$MPFC), type = "log")
```

```r
MPF.BCT.RMBB <- monthlyReturn(as.xts(MPF.BCT$RMBB), type = "log")
MPF.BCT.SFP <- monthlyReturn(as.xts(MPF.BCT$SFP), type = "log")
MPF.BCT.SE40 <- monthlyReturn(as.xts(MPF.BCT$SE2040), type = "log")


MPF.BCT.returns <-
  merge(
    MPF.BCT.AE,
    MPF.BCT.CA,
    MPF.BCT.CEHK,
    MPF.BCT.CT,
    MPF.BCT.EU,
    MPF.BCT.E3,
    MPF.BCT.E5,
    MPF.BCT.E7,
    MPF.BCT.E9,
    MPF.BCT.FM,
    MPF.BCT.GB,
    MPF.BCT.GE,
    MPF.BCT.GT,
    MPF.BCT.HKB,
    MPF.BCT.HSIT,
    MPF.BCT.MPFC,
    MPF.BCT.RMBB,
    MPF.BCT.SFP,
    MPF.BCT.SE40
  )

MPF.BCT.original.cost <-
  c(
    0.0182,
    0.0085,
    0.0166,
    0.0115,
    0.0165,
    0.0163,
    0.0163,
    0.0162,
    0.0152,
    0.0136,
    0.0150,
    0.0167,
    0.0100,
    0.0112,
    0.0084,
    0.0094,
    0.0126,
    0.0082,
    0.0150
  ) / 12
MPF.BCT.cs.cost <-
  c(
    0.0069,
```

```
    0.0062,
    0.0069,
    0.0060,
    0.0069,
    0.0062,
    0.0062,
    0.0062,
    0.0062,
    0.0062,
    0.0055,
    0.0069,
    0.0060,
    0.0055,
    0.0050,
    0.0040,
    0.0055,
    0.0062,
    0.0062
  ) / 12

for (col in 1:length(MPF.BCT.returns[1, ])) {
  MPF.BCT.returns[, col] <-
    MPF.BCT.returns[, col] - MPF.BCT.cs.cost[col] + MPF.BCT.original.cost[col]
}


monthly <- index(MPF.BCT.returns)
#monthly <- length(monthly)
MPF.BCT.colnames <-
  c(
    "MPF.BCT.AE",
    "MPF.BCT.CA",
    "MPF.BCT.CEHK",
    "MPF.BCT.CT",
    "MPF.BCT.EU",
    "MPF.BCT.E3",
    "MPF.BCT.E5",
    "MPF.BCT.E7",
    "MPF.BCT.E9",
    "MPF.BCT.FM",
    "MPF.BCT.GB",
    "MPF.BCT.GE",
    "MPF.BCT.GT",
    "MPF.BCT.HKB",
    "MPF.BCT.HSIT",
    "MPF.BCT.MPFC",
    "MPF.BCT.RMBB",
    "MPF.BCT.SFP",
    "MPF.BCT.SE2040"
  )

colnames(MPF.BCT.returns) <- MPF.BCT.colnames
```

```
rm(
  MPF.BCT.AE,
  MPF.BCT.CA,
  MPF.BCT.CEHK,
  MPF.BCT.CT,
  MPF.BCT.EU,
  MPF.BCT.E3,
  MPF.BCT.E5,
  MPF.BCT.E7,
  MPF.BCT.E9,
  MPF.BCT.FM,
  MPF.BCT.GB,
  MPF.BCT.GE,
  MPF.BCT.GT,
  MPF.BCT.HKB,
  MPF.BCT.HSIT,
  MPF.BCT.MPFC,
  MPF.BCT.RMBB,
  MPF.BCT.SFP,
  MPF.BCT.SE40
)
```

Calculate average RSI of the month, and then adjustment factor

Adjustment factor = 1 - ECDF of RSI of that month New weight = old weight * (0.05 + adjustment factor) Finally normalize it to sum(row) = 1

```
MPF.BCT.RSI.month <-
  as.xts(do.call(rbind, lapply(split(as.xts(MPF.BCT.RSI), "months"), function(x)
    colAvgs(x))), order.by = monthly)
MPF.BCT.RSI.p <- MPF.BCT.returns
MPF.BCT.RSI.p[, ] <- 0
for (col in 1:length(MPF.BCT.RSI.month[1, ])) {
  if (col != 16) {
    for (row in 1:length(MPF.BCT.RSI.month[, col])) {
      percentile <- ecdf(as.numeric(MPF.BCT.RSI.month[1:row, col]))
      if (percentile(MPF.BCT.RSI.month[row, col]) >= (RSI_Overbuy - ((length(1:row) ^
                                                                      (1 / 3)) / (length(1:row) ^ (1 / 1
        MPF.BCT.RSI.p[row, col] <- 0.4
      } else {
        MPF.BCT.RSI.p[row, col] <-
          1.4 - (percentile(MPF.BCT.RSI.month[row, col]) ^ 2)
      }
    }

  } else {
    MPF.BCT.RSI.p[, col] <- 1
  }
}
MPF.BCT.RSI.sum <-
  as.xts(rowSums(MPF.BCT.RSI.p), order.by = monthly)
for (row in 1:length(MPF.BCT.RSI.p[, col])) {
  MPF.BCT.RSI.p[row, ] = apply(MPF.BCT.RSI.p[row, ], 2, function(x)
```

```
    (x / MPF.BCT.RSI.sum[row, 1]) ^ (0.25))
}
MPF.BCT.RSI.sum <-
  as.xts(rowSums(MPF.BCT.RSI.p), order.by = monthly)
```

load python data

```
MPF.BCT.w.all <- array(MPF.BCT.returns,
                       c(length(MPF.BCT.returns[,1]),length(MPF.BCT.returns[1,]),20))
MPF.BCT.w.all[,,] <- 0

max_return <- c()
min_return <- c()

for (col in 1:length(MPF.BCT.returns[1,])) {
  max_return[col] <- max(na.omit(MPF.BCT.returns[,col]))
  min_return[col] <- min(na.omit(MPF.BCT.returns[,col]))
}

for (i in 1:20) {
  MPF.BCT.w.all[,,i] <- data.matrix(as.data.frame(read_feather(paste("F:/GitHub repo/MPF-ANN/temp/mpf_b
}
```

Calculate the weight according to predicted return

```
MPF.BCT.w <- MPF.BCT.returns
MPF.BCT.w[, ] <- 0

MPF.BCT.w <- rowAvgs(MPF.BCT.w.all, dims = 2)
MPF.BCT.w[MPF.BCT.w == 0] <- NA
for (col in 1:length(MPF.BCT.w[1, ])) {
  MPF.BCT.w[, col] <-
    ((MPF.BCT.w[, col] + 1) / 2 * (max_return[col] - min_return[col]) + min_return[col])
}



MPF.portf.weight <- MPF.BCT.returns
MPF.portf.weight[,] <- NA
MPF.portf.weight.all <- MPF.BCT.returns
MPF.portf.weight.all[, ] <- NA
colnames(MPF.portf.weight) <-
  c(
    "MPF.BCT.AE",
    "MPF.BCT.CA",
    "MPF.BCT.CEHK",
    "MPF.BCT.CT",
    "MPF.BCT.EU",
    "MPF.BCT.E3",
    "MPF.BCT.E5",
```

```
    "MPF.BCT.E7",
    "MPF.BCT.E9",
    "MPF.BCT.FM",
    "MPF.BCT.GB",
    "MPF.BCT.GE",
    "MPF.BCT.GT",
    "MPF.BCT.HKB",
    "MPF.BCT.HSIT",
    "MPF.BCT.MPFC",
    "MPF.BCT.RMBB",
    "MPF.BCT.SFP",
    "MPF.BCT.SE2040"
  )


MPF.BCT.stock.return <-
  as.xts(rowSums(MPF.BCT.RSI.month), order.by = monthly)
MPF.BCT.stock.return[] <- NA
MPF.portf.return <-
  as.xts(rowSums(MPF.BCT.RSI.month), order.by = monthly)
MPF.portf.return[] <- NA


MPF.BCT.returns.mat <- as.matrix(MPF.BCT.returns)

MPF.BCT.p <- as.matrix(MPF.BCT.returns)
MPF.BCT.p[,] <- 0
SR.all <- c()


hedge <- FALSE
hedge_once <- FALSE
up <- TRUE
round_percent <- function(x) {
  x <- x * 100
  result <- floor(x)    # Find integer bits
  remain <- x - result
  rsum <- sum(result)   # Find out how much we are missing
  i <- 1
  if (rsum < 100) {
    o <- order(remain, decreasing = TRUE)
    while (rsum < 100) {
      if (i > length(remain))
        i <- 1
      idx <- o[i]
      if (result[idx] == 0) {
        i <- i + 1
        next
      }
      result[idx] <- result[idx] + 1
      rsum <- sum(result)
      i <- i + 1
    }
```

```
  }
  result <- result / 100
  return(result)
}


for (row in 1:length(MPF.BCT.w[, 1])) {
  MPF.BCT.stock.mean <- 0
  i <- 0

  for (col in 1:length(MPF.BCT.w[1,])) {
    MPF.BCT.w[row, col] <-
      na.fill((MPF.BCT.w[row, col]) * MPF.BCT.RSI.p[row, col], 0)

    if (col != 2 &&
        col != 6 &&
        col != 7 &&
        col != 11 && col != 14 && col != 16 && col != 17 && col != 18) {
      if (!is.na(MPF.BCT.returns.mat[row, col]) &&
          MPF.BCT.returns.mat[row, col] != 0) {
        MPF.BCT.stock.mean <-
          MPF.BCT.stock.mean + MPF.BCT.returns.mat[row, col]
        i <- i + 1
      }

      if (col == 3 || col == 4 || MPF.BCT.w[row, col] < 1e-6) {
        MPF.BCT.w[row, col] <- 0
      }
    } else {
      if (MPF.BCT.w[row, col] < 0) {
        MPF.BCT.w[row, col] <- 0
      }
    }
  }

  MPF.BCT.stock.return[row] <- MPF.BCT.stock.mean / i

  # Retain two most increasing fund
  last <- length(MPF.BCT.w[1,]) - top
  order <- order(MPF.BCT.w[row,])
  for (col in order[1:last]) {
    MPF.BCT.w[row, col] <- 0
  }



  if (row > 12 &&
      MPF.BCT.stock.return[row] >
      quantile(na.omit(MPF.BCT.stock.return), c(.35))  &&
      MPF.BCT.stock.return[(row - 3)] >
      quantile(na.omit(MPF.BCT.stock.return), c(.45))) {
    up <- FALSE
  }
```

```
if (row > 12 && hedge &&
    MPF.BCT.stock.return[(row)]>
    quantile(na.omit(MPF.BCT.stock.return), c(.55))  &&
    MPF.BCT.stock.return[(row - 3)] >
    quantile(na.omit(MPF.BCT.stock.return), c(.35))) {
  hedge <- FALSE
  up <- TRUE
}

if (row > 12 &&  (MPF.BCT.stock.return[row] < 0 &&
                  MPF.BCT.stock.return[row - 1] >
                  quantile(na.omit(MPF.BCT.stock.return), c(.75)))) {
    hedge <- TRUE
}



MPF.BCT.w.sum <- sum(MPF.BCT.w[row,])


if (row <= 15 || MPF.BCT.w.sum == MPF.BCT.w[row, 16] ||
    MPF.BCT.w.sum < 1e-6 || hedge == TRUE || hedge_once == TRUE) {
  if (row >= 24) {
    MPF.BCT.p[row, 11] <- 0.35
    MPF.BCT.p[row, 16] <- 0.65
  } else {
    MPF.BCT.p[row, 16] <- 1
  }
} else if (length(which(MPF.BCT.w[row,] != 0)) == 1 ||
           min(MPF.BCT.stock.return[(row - 3):row]) < -0.065){
  if (row >= 24) {
    MPF.BCT.p[row,] <- MPF.BCT.w[row,] / MPF.BCT.w.sum / 10 * 4
    MPF.BCT.p[row, 11] <- MPF.BCT.p[row, 11] + 0.45
    MPF.BCT.p[row, 16] <- MPF.BCT.p[row, 16] + 0.15
  } else {
    MPF.BCT.p[row,] <- MPF.BCT.w[row,] / MPF.BCT.w.sum / 10 * 4
    MPF.BCT.p[row, 16] <- MPF.BCT.p[row, 16] + 0.6
  }
} else {
  MPF.BCT.p[row,] <- MPF.BCT.w[row,] / MPF.BCT.w.sum
}

MPF.portf.weight[row,] <- round_percent(MPF.BCT.p[row,])
portf.rebal.fm <-
  Return.portfolio(
    MPF.BCT.returns,
    weight = MPF.portf.weight,
    geometric = TRUE,
    rebalance_on = "months"
  )

MPF.portf.return[row] <-
  tail(na.omit(portf.rebal.fm), 1)
```

```
  MPF.portf.drawdown <- Drawdowns(MPF.portf.return,
                                  geometric = TRUE)
  if (tail(na.omit(MPF.portf.drawdown), 1) < -0.065 &&
      up == FALSE) {
    hedge = TRUE
  }
}

cl <- makeCluster(24)
registerDoParallel(cl)
SR.all <- foreach(pass=1:20, .combine="c", .packages=c("zoo", "xts", "PerformanceAnalytics")) %dopar% {
# for (pass in 1:20) {

  MPF.BCT.w.i <- MPF.BCT.w
  MPF.BCT.w.i[,] <- MPF.BCT.w.all[, ,pass]
  MPF.BCT.w.i[MPF.BCT.w.i == 0] <- NA
  for (col in 1:length(MPF.BCT.w.i[1, ])) {
    MPF.BCT.w.i[, col] <-
      ((MPF.BCT.w.i[, col] + 1) / 2 * (max_return[col] - min_return[col]) + min_return[col])
  }

  MPF.BCT.returns.mat <- as.matrix(MPF.BCT.returns)

  for (row in 1:length(MPF.BCT.w.i[, 1])) {
    MPF.BCT.stock.mean <- 0
    i <- 0

    for (col in 1:length(MPF.BCT.w.i[1,])) {
      MPF.BCT.w.i[row, col] <-
        na.fill((MPF.BCT.w.i[row, col]) * MPF.BCT.RSI.p[row, col], 0)

      if (col != 2 &&
          col != 6 &&
          col != 7 &&
          col != 11 && col != 14 && col != 16 && col != 17 && col != 18) {
        if (!is.na(MPF.BCT.returns.mat[row, col]) &&
            MPF.BCT.returns.mat[row, col] != 0) {
          MPF.BCT.stock.mean <-
            MPF.BCT.stock.mean + MPF.BCT.returns.mat[row, col]
          i <- i + 1
        }

        if (col == 3 || col == 4 || MPF.BCT.w.i[row, col] < 1e-6) {
          MPF.BCT.w.i[row, col] <- 0
        }
      } else {
        if (MPF.BCT.w.i[row, col] < 0) {
          MPF.BCT.w.i[row, col] <- 0
        }
      }
    }

    MPF.BCT.stock.return[row] <- MPF.BCT.stock.mean / i
```

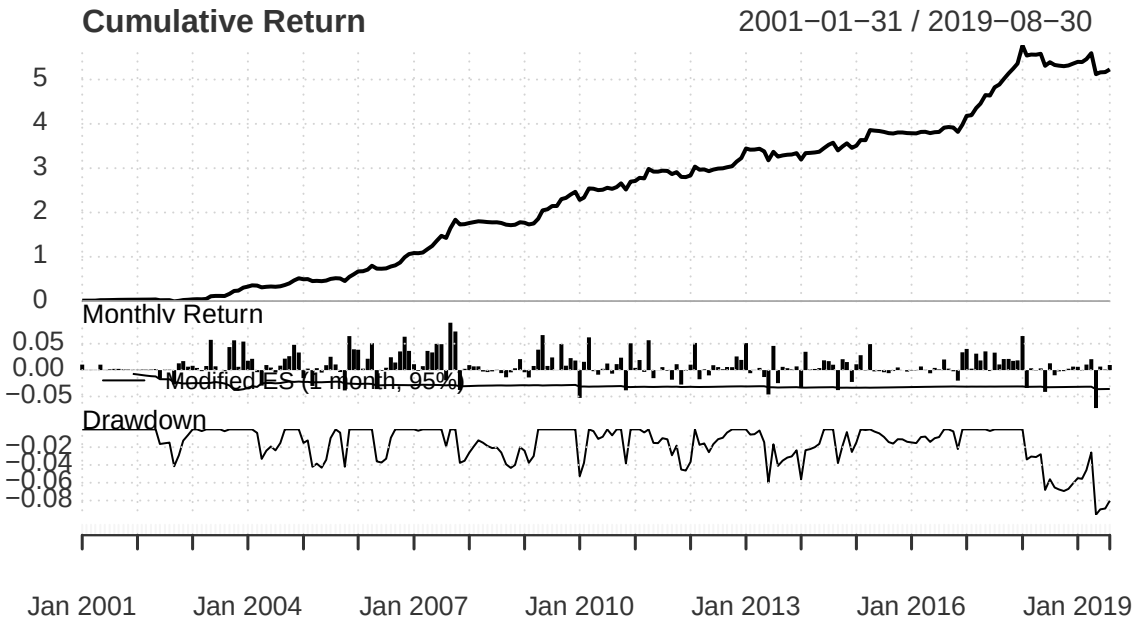
```r
# Retain two most increasing fund
last <- length(MPF.BCT.w.i[1,]) - top
order <- order(MPF.BCT.w.i[row,])
for (col in order[1:last]) {
  MPF.BCT.w.i[row, col] <- 0
}

#print("segment 1")

if (row > 12 &&
    MPF.BCT.stock.return[row] >
    quantile(na.omit(MPF.BCT.stock.return), c(.35))  &&
    MPF.BCT.stock.return[(row - 3)] >
    quantile(na.omit(MPF.BCT.stock.return), c(.45))) {
  up <- FALSE
}

if (row > 12 && hedge &&
    MPF.BCT.stock.return[(row)]>
    quantile(na.omit(MPF.BCT.stock.return), c(.55))  &&
    MPF.BCT.stock.return[(row - 3)] >
    quantile(na.omit(MPF.BCT.stock.return), c(.35))) {
  hedge <- FALSE
  up <- TRUE
}

if (row > 12 &&  (MPF.BCT.stock.return[row] < 0 &&
                  MPF.BCT.stock.return[row - 1] >
                  quantile(na.omit(MPF.BCT.stock.return), c(.75)))) {
    hedge <- TRUE
}

MPF.BCT.w.sum <- sum(MPF.BCT.w.i[row,])
MPF.BCT.p[row, ] <- 0

if (row <= 15 || MPF.BCT.w.sum == MPF.BCT.w.i[row, 16] ||
    MPF.BCT.w.sum < 1e-6 || hedge == TRUE || hedge_once == TRUE) {
  if (row >= 24) {
    MPF.BCT.p[row, 11] <- 0.35
    MPF.BCT.p[row, 16] <- 0.65
  } else {
    MPF.BCT.p[row, 16] <- 1
  }
} else if (length(which(MPF.BCT.w.i[row,] != 0)) == 1 ||
           min(MPF.BCT.stock.return[(row - 3):row]) < -0.065){
  if (row >= 24) {
    MPF.BCT.p[row,] <- MPF.BCT.w.i[row,] / MPF.BCT.w.sum / 10 * 4
    MPF.BCT.p[row, 11] <- MPF.BCT.p[row, 11] + 0.45
    MPF.BCT.p[row, 16] <- MPF.BCT.p[row, 16] + 0.15
  } else {
    MPF.BCT.p[row,] <- MPF.BCT.w.i[row,] / MPF.BCT.w.sum / 10 * 4
    MPF.BCT.p[row, 16] <- MPF.BCT.p[row, 16] + 0.6
  }
```

```
    } else {
      MPF.BCT.p[row,] <- MPF.BCT.w.i[row,] / MPF.BCT.w.sum
    }

    MPF.portf.weight.all[row, ] <-
      round_percent(MPF.BCT.p[row,])

    portf.rebal.i <-
      Return.portfolio(
        MPF.BCT.returns,
        weight = MPF.portf.weight.all,
        geometric = TRUE,
        rebalance_on = "months"
      )

    MPF.portf.return[row] <- tail(na.omit(portf.rebal.i), 1)
    MPF.portf.drawdown <- Drawdowns(MPF.portf.return,
                                    geometric = TRUE)
    if (row > 12 && tail(na.omit(MPF.portf.drawdown), 1) < -0.065 &&
        up == FALSE) {
      hedge = TRUE
    }
  }
  # SR.all[pass] <- Return.annualized(portf.rebal.i, geometric = TRUE) / (StdDev(portf.rebal.i) * sqrt(
  SR <- Return.annualized(portf.rebal.i, geometric = TRUE) / (StdDev(portf.rebal.i) * sqrt(12))
  SR
}

registerDoSEQ()
stopCluster(cl)
```

Performance Analysis

```
portf.rebal.fm <- Return.portfolio(
  MPF.BCT.returns,
  weight = MPF.portf.weight,
  geometric = TRUE,
  rebalance_on = "months"
)
mean.annual.return <-
  mean(do.call(rbind, lapply(split(portf.rebal.fm, "years"), function(x)
    colMeans(x))) * 12)
charts.PerformanceSummary(
  portf.rebal.fm,
  methods = "ModifiedES",
  geometric = TRUE,
  p = .95,
  main = "BCT MPF Scheme First Contribution Performance"
)
```

BCT MPF Scheme First Contribution Performance




```
portf.rebal.fm.sharpe <-
  Return.annualized(portf.rebal.fm, geometric = TRUE) / (StdDev(portf.rebal.fm) * sqrt(12))
portf.rebal.fm.mean.sharpe <-
  mean.annual.return / (StdDev(portf.rebal.fm) * sqrt(12))
rownames(portf.rebal.fm.sharpe) <- "Sharpe Ratio (annualized)"
rownames(portf.rebal.fm.mean.sharpe) <-
  "Sharpe Ratio (mean annual return)"
colnames(portf.rebal.fm.mean.sharpe) <- "portfolio.returns"
Return.annualized(portf.rebal.fm, geometric = TRUE)
```

```
##                   portfolio.returns
## Annualized Return         0.1029425
```

```
mean.annual.return
```

```
## [1] 0.09931035
```

```
portf.rebal.fm.sharpe
```

```
##                            portfolio.returns
## Sharpe Ratio (annualized)          1.279658
```

```
portf.rebal.fm.mean.sharpe
```

```
##                                   portfolio.returns
## Sharpe Ratio (mean annual return)          1.234508
```

```
StdDev.annualized(portf.rebal.fm)
```

```
##                                portfolio.returns
## Annualized Standard Deviation        0.08044531
```

```
SortinoRatio(portf.rebal.fm)
```

```
##                           portfolio.returns
## Sortino Ratio (MAR = 0%)          0.746187
```

```
ES(portf.rebal.fm, method = "historical")
```

```
##    portfolio.returns
## ES       -0.04144657
```

```
tail(MPF.portf.weight, n = 1)
```

```
##            MPF.BCT.AE MPF.BCT.CA MPF.BCT.CEHK MPF.BCT.CT MPF.BCT.EU MPF.BCT.E3 MPF.BCT.E5 MPF.BCT.E7
## 2019-08-30          0          0            0          0          0          0          0          0
##            MPF.BCT.HKB MPF.BCT.HSIT MPF.BCT.MPFC MPF.BCT.RMBB MPF.BCT.SFP MPF.BCT.SE2040
## 2019-08-30           0            0         0.65            0           0              0
```

```
### Deflated Sharpe Ratio
SR_zero <-
  sqrt((StdDev(SR.all)) ^ 2 / 12) * ((1 - 0.57721) * qnorm(1 - 1 / 21) +
                                       (0.57721) * qnorm(1 - (1 / (21 * 2.71828))))
DSR <-
  pnorm(((portf.rebal.fm.sharpe / sqrt(12) - SR_zero) * sqrt(length(MPF.BCT.returns[, 1]))) /
          sqrt(1 - skewness(portf.rebal.fm) * portf.rebal.fm.sharpe +
                 ((kurtosis(portf.rebal.fm) - 1) / 4) * (portf.rebal.fm.sharpe) ^ 2)
  )

rownames(DSR) <- "Deflated Sharpe Ratio"
DSR
```

```
##                       portfolio.returns
## Deflated Sharpe Ratio                 1
```

Monthly Installment

```r
MPF.BCT.units <- MPF.BCT.returns
MPF.BCT.units[, ] <- 0

MPF.monthly.asset <- MPF.BCT.returns
MPF.monthly.asset[, ] <- 0

MPF.monthly.returns <-
  as.xts(rowSums(MPF.BCT.returns), order.by = monthly)
MPF.monthly.returns[] <- 0

MPF.time <- 0:length(MPF.BCT.returns[, 1]) / 12
MPF.pay <- -1500 + 0 * MPF.time

for (row in 1:length(MPF.BCT.returns[, 1])) {
  this.price <- as.matrix(MPF.BCT[monthly[row]])
  MPF.BCT.units[row, ] <- this.price

  if (row == 1) {
    last.value <- 1500
    this.value <- as.numeric((1500 / MPF.BCT[1, 16]) * this.price[16])
    MPF.monthly.returns[row] <- log(this.value / last.value)
    MPF.monthly.asset[row,] <-
      (this.value + 1500) / this.price * MPF.portf.weight[row, ]
    last.price <- this.price
  } else {
    last.value <-
      as.numeric(sum(na.fill(last.price * MPF.monthly.asset[row - 1, ], 0)))
    this.value <-
      as.numeric(sum(na.fill(this.price * MPF.monthly.asset[row - 1, ], 0)))
    MPF.monthly.returns[row] <- log(this.value / last.value)
    MPF.monthly.asset[row,] <-
      (this.value + 1500) / this.price * MPF.portf.weight[row, ]
    last.price <- this.price
  }
}


total.asset.value <- sum(MPF.monthly.asset[row, ] * this.price)
total.contribution <- 1500 * length(MPF.BCT.returns[, 1])

MPF.pay[row + 1] <- total.asset.value
IRR.f <- function(r)
  sum(MPF.pay * exp(-r * MPF.time))
IRR.root <- uniroot(IRR.f, c(0, 1))


total.asset.value
```

```
## [1] 871187.4
```

```r
total.contribution
```

```
## [1] 337500
```

```
mean.monthly.annual.return <-
  mean(do.call(rbind, lapply(split(MPF.monthly.returns, "years"), function(x)
    colMeans(x))) * 12)
mean.monthly.annual.return
```

```
## [1] 0.08704892
```

```
IRR.root$root
```

```
## [1] 0.08870753
```

```
stddev.monthly <- (StdDev(MPF.monthly.returns) * sqrt(12))
monthly.installment.sharpe.ratio <-
  mean.monthly.annual.return / stddev.monthly
rownames(monthly.installment.sharpe.ratio) <-
  "Sharpe Ratio (mean annual return)"
monthly.installment.sharpe.ratio
```

```
##                                       [,1]
## Sharpe Ratio (mean annual return) 1.085536
```

```
StdDev.annualized(MPF.monthly.returns)
```

```
##                                     [,1]
## Annualized Standard Deviation 0.08018983
```

```
ES(MPF.monthly.returns, method = "historical")
```

```
##           [,1]
## ES -0.04210941
```

```
SortinoRatio(MPF.monthly.returns)
```

```
##                                [,1]
## Sortino Ratio (MAR = 0%) 0.6735455
```