銀聯強積金應用人工神經網絡的早期試驗 - BCT
MPF-ANN Backtesting (Preliminary Trial)

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: 2019/11/22.

The algorithm have been changed significantly on 2019/08/15.

We removed the VIX condition in our algorithm.
Changes in condition of hedge are also expected.
Three major outcomes is targeted in our study:
1. ~8-10% annualized return
2. <10% annualized standard deviation (~3% monthly)
3. >1 sharpe ratio

Monthly installment results available from 2019/09/12.

Due to extreme volality and manipulated market, the Chinese Tracker fund (col = 4) have been manually excluded from calculation with effect from 2019/09/12.

The raw data feed into the neural network have modified into timestep format on 2019/11/22.

WARNING: Please read the notification in this section first:

This script uses artifical neural network (ANN) model for MPF fund allocation.
However, ANN is stochastic in nature. It produces different results based on the pseudo-random seed (system time).
30 portfolios have been generated in order to generate trustworthy statistics for better prediction.

In addition, the prediction have no additional validation, the portfolio used a kind of walk-forward validation.
If the model is poor, then the return shall be poor.

In our script, we determined 0.0013 as the initial learning rate and epochs of 22.

Introduction

Mandatory Provident Fund (MPF) is one of the legal pensions in Hong Kong.
Employees and Employers are required to pay at least 5% of monthly salary to MPF services provider.
MPF providers are required to provide designated 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).
Hence most Hongkong employees feels they are forced to give their salary to fund manager.
Indeed, the reasons of low return may due to inactive management by employees.
In this example, we will explore artifical neural network to rise the annualized return.

BCT MPF Portfolio Generation using Artifical Neural Network (ANN) Relative Strength Index

Artifical Neural Network is a stochastic black-box model. It receives input and output the result by passing weighted values in neurons.
Formal definition could be searched on internet.
To support our application, we will uses Relative Strength Index (RSI) (customized period) to refine our prices.
RSI shows whether a stock/fund overbuy (or oversell) hence overpriced (or underpriced).

In our example, we directly uses Long-Short Term Memory to train and predict the future price of MPF constitute fund prices.
The fund would be finalized by applying penalty based on the result of RSI and VIX.

When to manage?

In this example, we are going to collect daily price for all MPF fund in last day of a month.
Then, we are going to convert or reallocate the assets at the same time.
Notice that it is impossible since MPF price are uploaded 1 business day, also reallocation need at lest 2 business day to achieve.

Results

Using Top 2 Performers in LSTM ANN (Lastest rebalance date: 2019/07/31)

Annualized Return: ~11.88%
Mean Annual Return: ~11.47%
Annualized Standard Deviation: ~9.49% (StdDev(monthly return) * sqrt(12))
Sharpe Ratio (Mean Annual Return): 11.47%/9.49% = 1.2091
Sortino Ratio: 0.8054 (MAR = 0%)
Expected Shortfall: 4.46% loss (0% Risk-free rate, 95% C.I.)
Deflated Sharpe Ratio (p-value): >99.9999% (rounded to 1)

Monthly installment:
Total contribution: 337500
Latest asset value: 1048491
Mean annual return: 10.14%
Internal Rate of Return (IRR): 10.41%
Annualized Standard Deviation: 9.46%
Sharpe Ratio: 1.0722
Sortino Ratio: 0.7324 (MAR = 0%)
Expected Shortfall: 4.53% loss (0% Risk-free rate, 95% C.I.)

Recommended Parameters

Variable	Val.	Explanation
top	2	Top n Performer
RSI_Overbuy	0.85	RSI indicator (%)
RSI_Period	18	MA period for RSI (months)
Min_NMMA	0.001	Minimum Monthly Return to be consider

Detailed Workflow

Package Preparation

1. Install necessary packages

```
r = getOption("repos")
r["CRAN"] = "https://mran.revolutionanalytics.com/snapshot/2019-10-01"
options(repos = r)
```

```
install.packages("zoo")
install.packages("xts")
install.packages("fBasics")
install.packages("quantmod")
install.packages("PerformanceAnalytics")
install.packages("keras")
```

2. Now load necessary packages.

```
library("zoo")
library("xts")
library("fBasics")
library("quantmod")
library("PerformanceAnalytics")
library("keras")
library("tensorflow")


require("knitr")
opts_chunk$set(tidy.opts=list(width.cutoff=80),tidy=TRUE)
```

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

```
setwd("~")
MPF.BCT <-
  as.xts(
    read.zoo(
      "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")
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)
colnames(MPF.BCT.returns) <-
  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"
  )

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 / 2))))) {
        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)
```

Train and predict with Long Short Term Memory (LSTM) model

```
use_python("~/tensorflow_v2/bin/python")
#use_condaenv("~/anaconda3/envs/cntk-py35")
use_implementation("tensorflow")
use_backend("tensorflow")

MPF.BCT.w.all <- array(MPF.BCT.returns,
                       c(length(MPF.BCT.returns[, 1]), length(MPF.BCT.returns[1,]), 30))

MPF.BCT.w.all[, ,] <- 0

for (pass in 1:30) {
  for (col in 1:length(MPF.BCT.returns[1, ])) {
    for (row in 1:monthly.old) {
      MPF.BCT.w.all[row, col, pass] <- MPF.BCT.w.all.backup[row, col, pass]
    }
  }
}

#MPF.BCT.period <- length(MPF.BCT.w.all[(monthly.existing+1):(monthly.new),1,1])
MPF.BCT.period <- length(MPF.BCT.w.all[, 1, 1])

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]))
}

MPF.BCT.returns_normalized <- MPF.BCT.returns
MPF.BCT.returns_normalized[,] <- 0

for (col in 1:length(MPF.BCT.returns[1,])) {
  MPF.BCT.returns_normalized[, col] <-
    ((MPF.BCT.returns[, col] - min_return[col]) / (max_return[col] - min_return[col])) * 2 - 1
}

temp <- as.matrix(MPF.BCT.returns_normalized)
```

```r
colNum <- 1:length(MPF.BCT.returns_normalized[1,])
matrix <- c()

seed <- c(
  71880,
  21251,
  98689,
  65940,
  22528,
  5447,
  13014,
  49976,
  57549,
  95690,
  18466,
  46047,
  55070,
  9292,
  82205,
  88714,
  38882,
  58473,
  75294,
  66679,
  68541,
  42252,
  41907,
  37306,
  75969,
  997,
  5609,
  95359,
  75506,
  93963
)
seed.int <- as.integer(seed)

tensorflow::tf$config$optimizer$set_jit(TRUE)
tensorflow::tf$config$threading$set_intra_op_parallelism_threads(8L)
tensorflow::tf$config$threading$set_inter_op_parallelism_threads(16L)
tensorflow::tf$keras$backend$set_floatx('float32')

for (pass in 1:30) {
  learning <- 0.0013
  n_steps <- 3


  tensorflow::tf$random$set_seed(seed[pass])
  #use_session_with_seed(seed[pass], FALSE, FALSE)

  model <- keras_model_sequential()
  model %>% layer_lstm(
    units = 64,
```

```
  activation = "tanh",
  input_shape = c(60, 1),
  return_sequences = TRUE,
  dtype = 'float32'
  #kernel_initializer = initializer_glorot_normal(seed[pass])
) %>%
  layer_lstm(
    units = 16,
    activation = "tanh",
    return_sequences = TRUE,
    dtype = 'float32'
    #kernel_initializer = initializer_glorot_normal(seed[pass])
  ) %>%
  layer_lstm(units = 4,
             activation = "tanh",
             dtype = 'float32'
             #,
             #kernel_initializer = initializer_glorot_normal(seed[pass])) %>%
             layer_dense(1)



             ad <- keras::optimizer_adam(lr = learning)
             model %>% compile(optimizer = ad, loss = "mean_squared_error")


             for (col in colNum) {
               X <- c()
               y <- c()

               counter <- 0
               minimum <- 4


               X <- as.matrix(na.omit(MPF.BCT.returns_normalized[, col]))
               #X_old <- length(na.omit(MPF.BCT.w.all.backup[,col,1]))
               y <-
                 as.matrix(na.omit(lag(
                   na.omit(MPF.BCT.returns_normalized[, col]), -1
                 )))

               predicted <- c()
               len <- 1

               #for (i in X_old:(length(X)-1)) {
               if (length(X) < 62) {
                 predicted <- rep(0, length(1:(length(
                   MPF.BCT.returns
                 ))))
               } else {
                 #for (i in 62:length(X) - 1) {
                 i <- length(X) - 1
```

```r
split_seq <- function(sequence_c, n_steps = 120) {
  X <- c()
  y <- c()

  for (idx in 1:length(sequence_c)) {
    end_ix <- idx + n_steps
    if (end_ix > length(sequence_c)) {
      break
    }
    seq_x <- sequence_c[idx:(end_ix - 1)]
    seq_y <- sequence_c[end_ix]
    X <- c(X, seq_x)
    y <- c(y, seq_y)
  }
  X_mat <- matrix(X, ncol = n_steps, byrow = FALSE)
  y_mat <- matrix(y, byrow = TRUE)
  list(X = X_mat, y = y_mat)
}

#X_train <- split_seq(X[1:i], 60)$X
#y_train <- split_seq(y[1:i], 60)$

X_train <- rollapply(
  X[1:(i)],
  FUN = function(x) {
    na <- rep(0, 60)
    if (length(x) == 60) {
      for (i in (61 - length(x)):60) {
        na[i] <- x[i]
      }
      na
    } else {
      for (i in length(x):1) {
        na[length(x) - i + 1] <- x[i]
      }
      rev(na)
    }
  },
  width = 60,
  partial = TRUE,
  fill = NA,
  align = "right"
)
y_train <-
  rollapply(
    y[1:(i)],
    FUN = c,
    width = 1,
    partial = TRUE,
    fill = NA,
    align = "right"
  )
```

```
#X_test <- split_seq(X[1:(i+1)], 60)$X
#y_test <- split_seq(y[1:(i+1)], 60)$y

dim(X_train) <- c(length(X_train[, 1]), 60, 1)
#dim(X_test) <- c(length(X_test[,1]),60,1)
# dim(X_train) <- c(length(X_train[,1]), 60)
# dim(X_test) <- c(length(X_test[,1]),60)


#dim(X_train) <- c(length(X_train), 60, 1)
#dim(X_test) <- c(length(X_test), 1, 1)

history <-
  model %>% fit(
    tf$cast(X_train, tf$float32),
    tf$cast(y_train, tf$float32),
    epochs = 36,
    batch_size = 128,
    verbose = 0
  )
#plot(history,metrics=c('loss'))


for (loop in 1:(length(X) - 1)) {
  X_test <- rollapply(
    X[1:(loop + 1)],
    FUN = function(x) {
      na <- rep(0, 60)
      if (length(x) == 60) {
        for (i in (61 - length(x)):60) {
          na[i] <- x[i]
        }
        na
      } else {
        for (i in length(x):1) {
          na[length(x) - i + 1] <- x[i]
        }
        rev(na)
      }
    },
    width = 60,
    partial = TRUE,
    fill = 0,
    align = "right"
  )

  dim(X_test) <- c(length(X_test[, 1]), 60, 1)
  X_test <- tail(X_test[, , 1], 1)
  dim(X_test) <- c(1, 60, 1)
  predicted[len] <-
    model %>% predict(X_test, batch_size = 128)
  len <- len + 1
```

```
                    }


                    result.length <- len - 1

                  }


                  #for (row in (monthly.existing + MPF.HSBC.period - result.length+1) :
                  #  (monthly.existing + MPF.HSBC.period)) {
                  for (row in (MPF.BCT.period - result.length + 1):(MPF.BCT.period)) {
                    MPF.BCT.w.all[row, col, pass] <-
                      predicted[(row + result.length - MPF.BCT.period)]
                  }
                }
                #keras::k_clear_session()
}
```

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.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
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 (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 > 8 &&
    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 > 8 && hedge &&
    MPF.BCT.stock.return[row] >
    quantile(na.omit(MPF.BCT.stock.return), c(.45))  &&
    MPF.BCT.stock.return[row - 3] >
    quantile(na.omit(MPF.BCT.stock.return), c(.35))) {
  hedge <- FALSE
  up <- TRUE
}
```

```
  if (row > 8 &&  (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 <= 12 || MPF.BCT.w.sum == MPF.BCT.w[row, 16] ||
      MPF.BCT.w.sum < 1e-6 || hedge == TRUE) {
    if (row >= 24) {
      MPF.BCT.p[row, 11] <- 0.3
      MPF.BCT.p[row, 16] <- 0.7
    } 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]) < -0.07) {
    if (row >= 24) {
      MPF.BCT.p[row,] <- MPF.BCT.w[row,] / MPF.BCT.w.sum / 3
      MPF.BCT.p[row, 11] <- MPF.BCT.p[row, 11] + 0.33
      MPF.BCT.p[row, 16] <- MPF.BCT.p[row, 16] + 0.34
    } else {
      MPF.BCT.p[row,] <- MPF.BCT.w[row,] / MPF.BCT.w.sum / 3
      MPF.BCT.p[row, 16] <- MPF.BCT.p[row, 16] + 0.67
    }
  } 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
  }
}

for (pass in 1:30) {
  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[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.w.all[, , pass])

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 (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

  # 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 > 8 &&
      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
  }
```

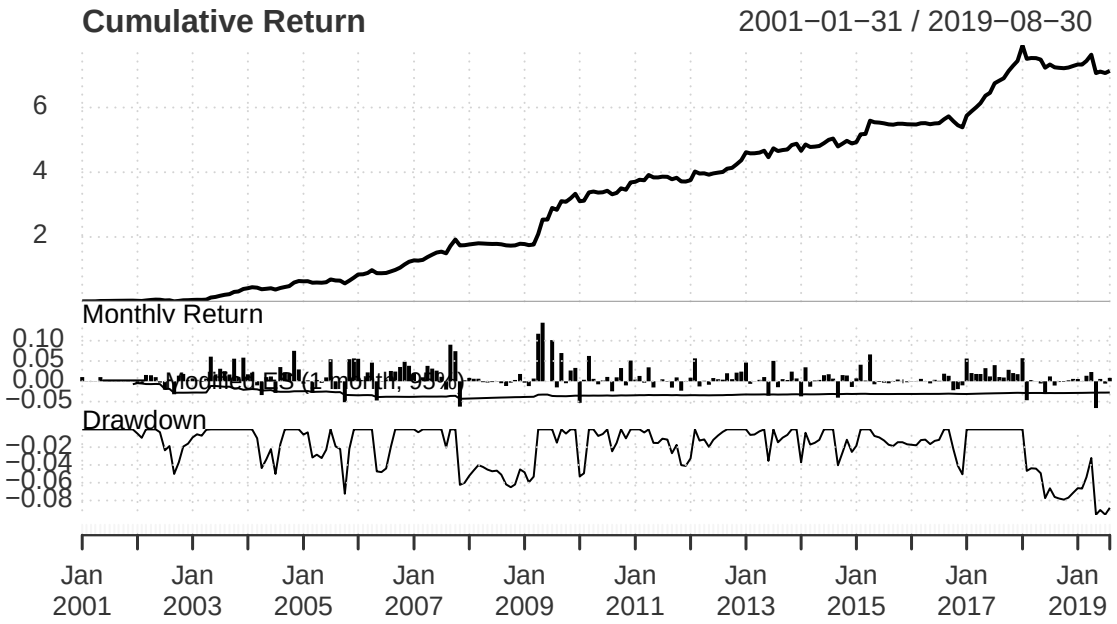
```
if (row > 8 && hedge &&
    MPF.BCT.stock.return[row] >
    quantile(na.omit(MPF.BCT.stock.return), c(.45))  &&
    MPF.BCT.stock.return[row - 3] >
    quantile(na.omit(MPF.BCT.stock.return), c(.35))) {
  hedge <- FALSE
  up <- TRUE
}

if (row > 8 &&  (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 <= 12 || MPF.BCT.w.sum == MPF.BCT.w.i[row, 16] ||
    MPF.BCT.w.sum < 1e-6 || hedge == TRUE) {
  if (row >= 24) {
    MPF.BCT.p[row, 11] <- 0.3
    MPF.BCT.p[row, 16] <- 0.7
  } 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]) < -0.07) {
  if (row >= 24) {
    MPF.BCT.p[row,] <- MPF.BCT.w.i[row,] / MPF.BCT.w.sum / 3
    MPF.BCT.p[row, 11] <- MPF.BCT.p[row, 11] + 0.33
    MPF.BCT.p[row, 16] <- MPF.BCT.p[row, 16] + 0.34
  } else {
    MPF.BCT.p[row,] <- MPF.BCT.w.i[row,] / MPF.BCT.w.sum / 3
    MPF.BCT.p[row, 16] <- MPF.BCT.p[row, 16] + 0.67
  }
} else {
  MPF.BCT.p[row,] <- MPF.BCT.w[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.fm), 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(12))
}
```

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         0.1188269
```

```
mean.annual.return
```

```
## [1] 0.1147017
```

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

```
##                           portfolio.returns
## Sharpe Ratio (annualized)          1.252589
```

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

```
##                                   portfolio.returns
## Sharpe Ratio (mean annual return)          1.209104
```

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

```
##                                portfolio.returns
```

```
## Annualized Standard Deviation        0.09486504
```

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

```
##                           portfolio.returns
## Sortino Ratio (MAR = 0%)         0.8053901
```

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

```
##    portfolio.returns
## ES       -0.04456358
```

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

```
##            MPF.BCT.AE MPF.BCT.CA MPF.BCT.CEHK MPF.BCT.CT MPF.BCT.EU
## 2019-08-30          0          0             0          0          0
##            MPF.BCT.E3 MPF.BCT.E5 MPF.BCT.E7 MPF.BCT.E9 MPF.BCT.FM
## 2019-08-30          0          0          0          0          0
##            MPF.BCT.GB MPF.BCT.GE MPF.BCT.GT MPF.BCT.HKB MPF.BCT.HSIT
## 2019-08-30        0.3          0          0           0            0
##            MPF.BCT.MPFC MPF.BCT.RMBB MPF.BCT.SFP MPF.BCT.SE2040
## 2019-08-30          0.7            0           0               0
```

```
### Deflated Sharpe Ratio
SR_zero <-
  sqrt((StdDev(SR.all)) ^ 2 / 12) * ((1 - 0.57721) * qnorm(1 - 1 / 31) +
                                       (0.57721) * qnorm(1 - (1 / (31 * 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

```
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] 1048491
```

```
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] 0.1014057
```

```
IRR.root$root
```

```
## [1] 0.1040667
```

```
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) 1.072172
```

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

```
##                                       [,1]
## Annualized Standard Deviation 0.09457973
```

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

```
##           [,1]
## ES -0.04532327
```

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

```
##                                [,1]
## Sortino Ratio (MAR = 0%) 0.7323655
```