如何用 R 语言实现含股票和债券的投资组合的再平衡

MatrixSpk

目录

0.1 引言

以下是使用 R 语言实现一个简单的 “股票-债券” 再平衡策略的代码示例。这里的策略基于阈值再平衡逻辑，即投资组合中的股票占比偏离目标比例超过设定阈值时触发再平衡调整。

1 策略逻辑

1.1 目标配置

设定股票（如沪深 300 指数 ETF）和债券（如国债 ETF）的目标比例，例如，股票 60%，债券 40%。

1.2 偏离阈值

当股票实际占比偏离目标比例 ±10% 时，触发再平衡（卖出超配资产，买入低配资产）。

1.3 再平衡规则

- 若股票占比 > 目标比例 + 阈值：卖出多余股票，买入债券；
- 若股票占比 < 目标比例 - 阈值：卖出部分债券，买入股票。

2 再平衡策略的 R 代码实现

2.1 安装并加载依赖包

```
# 安装相关 R 包
# install.packages(c("quantmod", "xts", "TTR"))

# 加载包
library(quantmod)   # 数据获取与处理
```

```
## 载入需要的程辑包: xts
```

```
## 载入需要的程辑包: zoo
```

```
##
## 载入程辑包: 'zoo'
```

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

```
## 载入需要的程辑包: TTR
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(xts)        # 时间序列数据
library(TTR)        # 简单指标计算
```

2.2 数据准备（模拟或获取真实数据）

```
# 生成模拟的股票（沪深 300）和债券（国债 ETF）价格数据（400 个交易日）
set.seed(123)
n_days <- 200
stock_returns <- cumsum(rnorm(n_days, mean = 0.001, sd = 0.02))
bond_returns <- cumsum(rnorm(n_days, mean = 0.0005, sd = 0.01))
stock_prices <- 100 * exp(stock_returns)
bond_prices <- 100 * exp(bond_returns)
dates <- as.Date("2024-01-01") + 0:(n_days-1)
prices <- as.xts(cbind(Stock = stock_prices, Bond = bond_prices), order.by = dates)
```

2.3 定义投资组合类和再平衡函数

```
Portfolio <- setRefClass(
  "Portfolio",
  fields = list(
    target_stock = "numeric",
    threshold = "numeric",
    initial_cash = "numeric",
    positions = "list",
    history = "list"
  ),

  methods = list(
    initialize = function(target_stock = 0.6, threshold = 0.1, initial_cash = 100000) {
      target_stock <<- target_stock
      threshold <<- threshold
      initial_cash <<- initial_cash
      positions <<- list(stock = 0, bond = 0, cash = initial_cash)
      history <<- list()
      invisible(.self)  # 使用 .self 代替 self
    },
```

```
  calculate_allocation = function(price) {
    stock_value <- positions$stock * price[, "Stock"]
    bond_value <- positions$bond * price[, "Bond"]
    total_value <- stock_value + bond_value + positions$cash
    list(
      stock_value = as.numeric(stock_value),
      bond_value = as.numeric(bond_value),
      cash = positions$cash,
      total_value = as.numeric(total_value),
      stock_ratio = as.numeric(stock_value / total_value),
      bond_ratio = as.numeric(bond_value / total_value)
    )
  },

  rebalance = function(price) {
    allocation <- calculate_allocation(price)
    stock_ratio <- allocation$stock_ratio

    if (abs(stock_ratio - target_stock) > threshold) {
      target_stock_value <- allocation$total_value * target_stock
      target_bond_value <- allocation$total_value * (1 - target_stock)
      adjust_stock <- target_stock_value - allocation$stock_value
      adjust_bond <- target_bond_value - allocation$bond_value

      # 使用 <<- 修改类字段
      positions$stock <<- positions$stock + adjust_stock / price[, "Stock"]
      positions$bond <<- positions$bond + adjust_bond / price[, "Bond"]
      positions$cash <<- positions$cash - adjust_stock - adjust_bond

      # 记录历史
      history <<- c(
        history,
        list(list(
```

```
            date = index(price),
            action = ifelse(adjust_stock > 0, " 买入股票", " 卖出股票"),
            stock_ratio = stock_ratio,
            target_ratio = target_stock,
            total_value = allocation$total_value
          ))
        )
      }
      invisible(allocation)
    },

  simulate = function(prices) {
  for (i in 1:nrow(prices)) {
    price_row <- prices[i, ]
    allocation <- calculate_allocation(price_row)
    rebalance(price_row)

    # 判断是否触发再平衡
    rebalanced_value <- if (length(history) > 0) {
      any(grepl("action", names(history[[length(history)]])))
    } else {
      FALSE
    }

    # 记录每日状态
    history <<- c(
      history,
      list(list(
        date = index(price_row),
        total_value = allocation$total_value,
        stock_ratio = allocation$stock_ratio,
        cash = allocation$cash,
        action = if (rebalanced_value) history[[length(history)]]$action else NA,
```

```
        rebalanced = rebalanced_value
      ))
    )
  }
  invisible(.self)
}
  )
)
```

初始化投资组合并运行回测

```
portfolio <- Portfolio$new(target_stock = 0.6, threshold = 0.1, initial_cash = 100000)

# 初始建仓
initial_price <- prices[1, ]
initial_stock_value <- portfolio$initial_cash * portfolio$target_stock
initial_bond_value <- portfolio$initial_cash * (1 - portfolio$target_stock)
portfolio$positions$stock <- initial_stock_value / as.numeric(initial_price[, "Stock"])
portfolio$positions$bond <- initial_bond_value / as.numeric(initial_price[, "Bond"])
portfolio$positions$cash <- 0  # 初始现金用尽

# 运行模拟
portfolio$simulate(prices)

# 查看历史记录
head(portfolio$history)
```

```
## [[1]]
## [[1]]$date
## [1] "2024-01-01"
##
## [[1]]$total_value
## [1] 1e+05
##
```

```
## [[1]]$stock_ratio
## [1] 0.6
##
## [[1]]$cash
## [1] 0
##
## [[1]]$action
## [1] NA
##
## [[1]]$rebalanced
## [1] FALSE
##
##
## [[2]]
## [[2]]$date
## [1] "2024-01-02"
##
## [[2]]$total_value
## [1] 100332.9
##
## [[2]]$stock_ratio
## [1] 0.5958583
##
## [[2]]$cash
## [1] 0
##
## [[2]]$action
## [1] NA
##
## [[2]]$rebalanced
## [1] TRUE
##
##
```

```
## [[3]]
## [[3]]$date
## [1] "2024-01-03"
##
## [[3]]$total_value
## [1] 102200.5
##
## [[3]]$stock_ratio
## [1] 0.6040964
##
## [[3]]$cash
## [1] 0
##
## [[3]]$action
## [1] NA
##
## [[3]]$rebalanced
## [1] TRUE
##
##
## [[4]]
## [[4]]$date
## [1] "2024-01-04"
##
## [[4]]$total_value
## [1] 102590.2
##
## [[4]]$stock_ratio
## [1] 0.6032538
##
## [[4]]$cash
## [1] 0
##
```

```
## [[4]]$action
## [1] NA
##
## [[4]]$rebalanced
## [1] TRUE
##
##
## [[5]]
## [[5]]$date
## [1] "2024-01-05"
##
## [[5]]$total_value
## [1] 102664.5
##
## [[5]]$stock_ratio
## [1] 0.6049827
##
## [[5]]$cash
## [1] 0
##
## [[5]]$action
## [1] NA
##
## [[5]]$rebalanced
## [1] TRUE
##
##
## [[6]]
## [[6]]$date
## [1] "2024-01-06"
##
## [[6]]$total_value
## [1] 104723.7
```

```
##
## [[6]]$stock_ratio
## [1] 0.6143973
##
## [[6]]$cash
## [1] 0
##
## [[6]]$action
## [1] NA
##
## [[6]]$rebalanced
## [1] TRUE
```

2.4 策略回测结果并可视化

```
# 整理历史数据
library(dplyr)
```

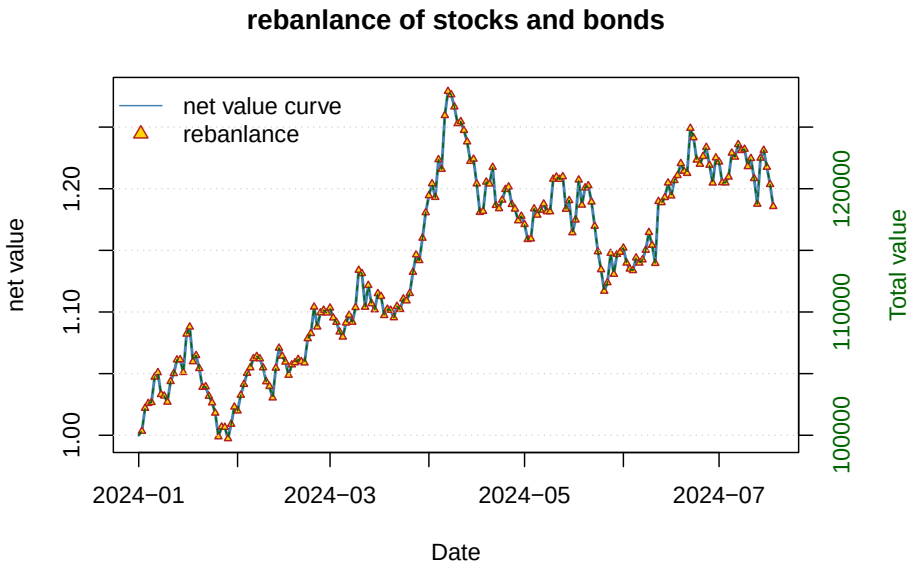
```
##
## ########################## Warning from 'xts' package ##########################
## #                                                                              #
## # The dplyr lag() function breaks how base R's lag() function is supposed to  #
## # work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
## # source() into this session won't work correctly.                            #
## #                                                                              #
## # Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
## # conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
## # dplyr from breaking base R's lag() function.                                #
## #                                                                              #
## # Code in packages is not affected. It's protected by R's namespace mechanism #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                                              #
## ################################################################################
```

```
##
## 载入程辑包：'dplyr'

## The following objects are masked from 'package:xts':
##
##     first, last

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# 合并历史数据并计算净值
history_df <- bind_rows(portfolio$history) %>%
  mutate(
    date = as.Date(date, format = "%Y-%m-%d"),  # 转换日期格式
    net_value = total_value / first(total_value)  # 基准化为初始净值
)

# 加载必要包
library(xts)
library(zoo)  # 用于时间序列处理

# 创建可复现示例数据（如果实际数据不存在）
# history_df <- structure(list(...))

# 设置图形参数
# par(family = 'STHeiti')  # 中文字体支持（Windows 用'SimHei', macOS 用'STHeiti'）
par(mar = c(5, 4, 4, 4) + 0.1)  # 调整图形边距

# 创建基础图形
```

```r
plot(
  x = history_df$date,
  y = history_df$net_value,
  type = "l",
  lwd = 2,
  col = "steelblue",
  xlab = "Date",
  ylab = "net value",
  main = "rebanlance of stocks and bonds",
  xaxt = "n"  # 禁用默认 x 轴
)

# 自定义日期坐标轴
axis.Date(
  side = 1,
  at = seq(min(history_df$date), max(history_df$date), by = "1 month"),
  format = "%Y-%m",
)

# 添加再平衡事件标记
points(
  x = history_df$date[history_df$rebalanced],
  y = history_df$net_value[history_df$rebalanced],
  pch = 24,  # 三角形标记
  col = "firebrick",
  bg = "gold",
  cex = 0.5
)

# 添加辅助网格线
grid(
  nx = NA,
  ny = NULL,  # 仅横向网格线
```

```
  col = "lightgray",
  lty = "dotted"
)

# 添加图例
legend(
  "topleft",
  legend = c("net value curve", "rebanlance"),
  col = c("steelblue", "firebrick"),
  lty = c(1, NA),
  pch = c(NA, 24),
  pt.bg = c(NA, "gold"),
  bty = "n"
)

# 可选：添加次坐标轴（例如显示原始总价值）
par(new = TRUE)
plot(
  x = history_df$date,
  y = history_df$total_value,
  type = "l",
  lty = 2,
  col = "darkgreen",
  axes = FALSE,
  xlab = "",
  ylab = ""
)
axis(4, col.axis = "darkgreen")
mtext("Total value", side = 4, line = 3, col = "darkgreen")
```



3 代码说明

3.1 数据准备：

- 使用 `rnorm` 生成模拟的股票和债券价格（可替换为真实数据，如用 `getSymbols("000300.SH", from = "2010-01-01")` 获取沪深 300 指数）。
- 数据格式为 `xts` 时间序列，方便按日期处理。

3.2 投资组合类：

- `Portfolio` 类包含目标配置、阈值、初始现金、持仓和历史记录等字段。
- `calculate_allocation` 计算当前资产比例，`rebalance` 执行再平衡逻辑，`simulate` 按日模拟交易。

3.3 再平衡逻辑：

- 每次计算股票实际占比，若偏离目标超过阈值（如 ±10%），则调整至目标比例（通过卖出/买入资产）。
- 记录每次再平衡的时间、操作和组合价值。

3.4 回测与可视化：

- 归一化净值曲线展示策略表现，红色点标记再平衡触发点，直观显示调整对组合的影响。

4 扩展建议

4.1 加入交易成本

在 rebalance 函数中添加佣金、滑点等成本（如 `cost <- abs(adjust_stock + adjust_bond) * 0.001`，假设千分之一佣金）。

4.2 处理最小交易单位：

避免买入零碎股（如 `floor(adjust_stock / price[1])` 取整）。

4.3 定期再平衡

增加按固定频率（如每年 12 月 31 日）触发再平衡的逻辑，而非仅依赖阈值。

4.4 多资产支持

扩展至股票、债券、黄金等多资产类别，通过循环遍历资产实现通用再平衡。通过以上代码，可直观理解再平衡策略在 R 中的实现逻辑，并根据需求调整参数（目标比例、阈值、资产类别等）。