Vodafone Big Data Lab – Financial and Risk Analytics course
Course Project "Portfolio Selection Strategies"

Dr. Oleksandr Romanko

May 27, 2023

Due: Saturday, June 17, 2023, not later than 18:00.

Use Python for this Course Project.

You should hand in:

- Your report in the form of Jupyter notebook.

Where to hand in: Online via Canvas portal, both your code and report as Jupyter notebook.

Introduction

The purpose of this course project is to compare computational investment strategies based on minimizing portfolio variance and maximizing Sharpe ratio. You will need to take into account the effect of trading costs.

We start with the classical Markowitz model of portfolio optimization. We are trading only stocks $S_1, \ldots, S_n$, and the total number of stocks is $n = 20$. At holding period t the expected (daily) return of stock i is μ_i^t, the standard deviation of the return is σ_i^t. The correlation between the returns of stocks $i \neq j$ is ρ_{ij}^t. To abbreviate notation we introduce the return vector $\mu^t = (\mu_1^t, \ldots, \mu_n^t)^T$ and the covariance matrix $Q^t = [\rho_{ij}^t \sigma_i^t \sigma_j^t]_{i,j=1,\ldots,n}$. At time period t we estimate μ^t and Q^t from the period $t-1$ time series. There are 12 holding periods and each holding period lasts 2 months (2 years in total). As a result, you can re-balance your portfolio at most 12 times.

Each transaction has a cost composed of a variable portion only. The variable fee is due to the difference between the selling and bidding price of a stock, and is 0.5% of the traded volume. This means that if a stock is quoted at 30 dollars and the investor buys 10 shares, then they are going to pay $0.005 \cdot 30 \cdot 10 = 1.50$ dollars on top of the 300 dollars the 10 shares would cost otherwise. If they sell 20 shares at the same 30 dollar price then they get 600 dollars, minus the $0.005 \cdot 600 = 3$ dollar transaction cost. Your may not need to account for transaction fees in your optimization, but you need to take them into account when computing portfolio value.

The goal of your optimization effort is to create a tool that allows the user to make regular decisions about re-balancing their portfolio and compare different investment strategies. The user wants to consider the total return, the risk and Sharpe ratio. They may want to minimize/maximize any of these components, while limiting one or more of the other components. The basic building block is a decision made at the first trading day of each 2-month holding period: given a current portfolio, the market prices on that day, and the estimates of the mean and covariance of the daily returns, make a decision about what to buy and sell according to a strategy. You are proposed to test four strategies:

1. "Buy and hold" strategy;
2. "Equally weighted" (also known as "$1/n$") portfolio strategy;
3. "Minimum variance" portfolio strategy;

4. "Maximum Sharpe ratio" portfolio strategy.

To test your bi-monthly decisions, you are given daily closing prices of 20 stocks over a two-year period. Obviously, you cannot use the actual future prices when making a decision, only the price at that day, the return and covariance estimates. These estimates were derived from the actual data and are quite accurate. Once you are done with trades for a current holding period, you should move on to the next period, use new prices (and new estimates), and make another decision. You also need to track the value of your portfolio on each day, so that you have it handy at the first trading day of a new period. You re-balance your portfolio on the first trading day of each 2-month holding period (up to 12 re-balances during 2 years).

The stocks for the assignment are from the US stock markets and are quoted in US dollars. Quotes are given on trading days, which are the days for which a price is given. The data is from 2019 and 2020.

The initial portfolio is as follows: "IBM" = 1045 shares, "BK" = 20000 shares.

The value of the initial portfolio is about one million dollars. The initial cash account is zero USD. The cash account must be nonnegative at all times. Your optimization algorithm may suggest buying or selling a non-integer number of shares, which is not allowed. You need to round the number of shares bought or sold to integer values and subtract transaction fees. If the value of your portfolio after re-balancing plus the transaction fees are smaller than the portfolio value before re-balancing, the remaining funds are accumulated in the cash account. Cash account does not pay any interest, but cash funds should be used toward stock purchases when portfolio is re-balanced next time.

An estimate of the daily returns of the stocks (for trading days) and the covariances of the returns are computed for you. These estimates are based on the previous two-month data and are updated every second month. You can only use the current estimate for the calculations and are not allowed to change or update the estimates in any way.

Note that for buying and selling asset shares, you need to optimize over asset weights. Current weight of asset i in a portfolio is $w_i = \frac{v_i \cdot x_i}{V}$, where V is the current portfolio value, v_i is current price of asset i and x_i is the number of units (shares) of asset i in your portfolio. As initial portfolio value is 1000020.59 USD, price of IBM stock at the first trading day of holding period 1 is 104.46 USD, and we hold 1045 shares, then $w_{10}^1 = \frac{v_{10}^1 \cdot x_{10}^1}{V^1} = \frac{104.46 \cdot 1045}{1000020.59} = 0.109$.

Questions

1. **(50 %) Implement investment strategies in Python:**

   You need to test four portfolio re-balancing strategies:

   1. "Buy and hold" strategy: it is the simplest strategy where you hold initial portfolio for the entire investment horizon of 2 years. The strategy is already implemented in the function `strat_buy_and_hold`.
   2. "Equally weighted" (also known as "$1/n$") portfolio strategy: asset weights are selected as $w_i^t = 1/n$, where n is the number of assets. You may need to re-balance your portfolio in each period as the number of shares x_i^t changes even when $w_i^t = 1/n$

stays the same in each period. The strategy should be implemented in the function `strat_equally_weighted`.

3. "Minimum variance" portfolio strategy: compute minimum variance portfolio for each period and re-balance accordingly. The strategy should be implemented in the function `strat_min_variance`.
4. "Maximum Sharpe ratio" portfolio strategy: compute a portfolio that maximizes Sharpe ratio for each period and re-balance accordingly. The strategy should be implemented in the function `strat_max_Sharpe`.

Design and implement a rounding procedure, so that you always trade (buy or sell) an integer number of shares.

Design and implement a validation procedure in your code to test that each of your strategies is feasible (you have enough budget to re-balance portfolio, you correctly compute transaction costs, funds in your cash account are non-negative).

There is a file **Portfolio_Selection.ipynb** on the course web-page. You are required to complete the code in the file.

Your Python code should use CVXPY module for solving optimization problems. Make commenting of your code for important logics.

2. **(10 %) Analyze your results for years 2019 and 2020:**

   - Produce the following output for the 12 periods (years 2019 and 2020):

```
Period 1: start date 01/02/2019, end date 02/28/2019
  Strategy "Buy and Hold", value begin = $ 1000020.59, value end = $ 1121819.46
  Strategy "Equally Weighted Portfolio", value begin = ... , value end = ...
  ...

Period 12: start date 11/02/2020, end date 12/31/2020
  Strategy "Buy and Hold", value begin = $ 812956.16, value end = $ 973877.89
  Strategy "Equally Weighted Portfolio", value begin = ... , value end = ...
```

   - Plot one chart in Python that illustrates the daily value of your portfolio (for each trading strategy) over the years 2019 and 2020 using daily prices provided. Include the chart in your report.
   - Compare your trading strategies and discuss their performance relative to each other. Which strategy would you select for managing your own portfolio and why?

3. **(20 %) Discuss possible improvements to your trading strategies:**

   - Design and implement your own portfolio selection strategy in addition to the four other strategies. Discuss if your own strategy can achieve better results.

4. **(20 %) Test your trading strategies for years 2008 and 2009:**

   - Download daily closing prices for the same twenty stocks for years 2008 and 2009. If necessary, transform data into a format that your code can read.
   - Test five strategies that you have implemented for the 12 periods (years 2008 and 2009). Use the same initial portfolio that you are given. Produce the same output for your strategies as in Question 2.
   - Plot one chart in Python that illustrates the daily value of your portfolio (for each of the five trading strategies) over the years 2008 and 2009 using daily prices that you have downloaded. Include the chart in your report.

- Compare and discuss relative performance of your five trading strategies during 2019-2020 and 2008-2009 time periods. Which strategy would you select for managing your own portfolio that works during normal economic periods and during crisis economic periods and why?

Python File to be Completed

```
# Import libraries
import pandas as pd
import numpy as np
import math

# Complete the following functions
def strat_buy_and_hold(x_init, cash_init, mu, Q, cur_prices):
   x_optimal = x_init
   cash_optimal = cash_init
   return x_optimal, cash_optimal

def strat_equally_weighted(x_init, cash_init, mu, Q, cur_prices):

    return x_optimal, cash_optimal

def strat_min_variance(x_init, cash_init, mu, Q, cur_prices):

   return x_optimal, cash_optimal

def strat_max_Sharpe(x_init, cash_init, mu, Q, cur_prices):

   return x_optimal, cash_optimal

def strat_your_strategy(x_init, cash_init, mu, Q, cur_prices):

   return x_optimal, cash_optimal

# Input file
input_file_prices = 'ts2019_2020.csv'

# Read data into a dataframe
df = pd.read_csv(input_file_prices)

# Convert dates into array [year month day]
def convert_date_to_array(datestr):
    temp = [int(x) for x in datestr.split('/')]
    return [temp[-1], temp[0], temp[1]]

dates_array = np.array(list(df['Date'].apply(convert_date_to_array)))
data_prices = df.iloc[:, 1:].to_numpy()
dates = np.array(df['Date'])
# Find the number of trading days in Nov-Dec 2018 and
# compute expected return and covariance matrix for period 1
day_ind_start0 = 0
day_ind_end0 = len(np.where(dates_array[:,0]==2018)[0])
cur_returns0 = data_prices[day_ind_start0+1:day_ind_end0,:] / data_prices[day_ind_start0:day_ind_end0-1,:] - 1
mu = np.mean(cur_returns0, axis = 0)
Q = np.cov(cur_returns0.T)

# Remove datapoints for year 2018
data_prices = data_prices[day_ind_end0:,:]
dates_array = dates_array[day_ind_end0:,:]
dates = dates[day_ind_end0:]

# Initial positions in the portfolio
init_positions = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1045, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20000])
```

```
# Initial value of the portfolio
init_value = np.dot(data_prices[0,:], init_positions)
print('\nInitial portfolio value = $ {}\n'.format(round(init_value, 2)))

# Initial portfolio weights
w_init = (data_prices[0,:] * init_positions) / init_value

# Number of periods, assets, trading days
N_periods = 6*len(np.unique(dates_array[:,0])) # 6 periods per year
N = len(df.columns)-1
N_days = len(dates)

# Annual risk-free rate for years 2019-2020 is 2.5%
r_rf = 0.025
# Annual risk-free rate for years 2008-2009 is 4.5%
r_rf2008_2009 = 0.045

# Number of strategies
strategy_functions = ['strat_buy_and_hold', 'strat_equally_weighted', 'strat_min_variance',
                      'strat_max_Sharpe', 'strat_your_strategy']
strategy_names     = ['Buy and Hold', 'Equally Weighted Portfolio', 'Mininum Variance Portfolio',
                      'Maximum Sharpe Ratio Portfolio', 'Your Own Strategy']
N_strat = 1  # comment this in your code
#N_strat = len(strategy_functions)  # uncomment this in your code
fh_array = [strat_buy_and_hold, strat_equally_weighted, strat_min_variance, strat_max_Sharpe]

portf_value = [0] * N_strat
x = np.zeros((N_strat, N_periods),  dtype=np.ndarray)
cash = np.zeros((N_strat, N_periods),  dtype=np.ndarray)
for period in range(1, N_periods+1):
   # Compute current year and month, first and last day of the period
   if dates_array[0, 0] == 19:
       cur_year  = 19 + math.floor(period/7)
   else:
       cur_year  = 2019 + math.floor(period/7)

   cur_month = 2*((period-1)%6) + 1
   day_ind_start =
       min([i for i, val in enumerate((dates_array[:,0] == cur_year) & (dates_array[:,1] == cur_month)) if val])
   day_ind_end =
       max([i for i, val in enumerate((dates_array[:,0] == cur_year) & (dates_array[:,1] == cur_month+1)) if val])
   print('\nPeriod {0}: start date {1}, end date {2}'.format(period, dates[day_ind_start], dates[day_ind_end]))

   # Prices for the current day
   cur_prices = data_prices[day_ind_start,:]

   # Execute portfolio selection strategies
   for strategy  in range(N_strat):

      # Get current portfolio positions
      if period == 1:
         curr_positions = init_positions
         curr_cash = 0
         portf_value[strategy] = np.zeros((N_days, 1))
      else:
         curr_positions = x[strategy, period-2]
         curr_cash = cash[strategy, period-2]

      # Compute strategy
      x[strategy, period-1], cash[strategy, period-1] = fh_array[strategy](curr_positions, curr_cash, mu, Q, cur_prices)

      # Verify that strategy is feasible (you have enough budget to re-balance portfolio)
      # Check that cash account is >= 0
      # Check that we can buy new portfolio subject to transaction costs

      ###################### Insert your code here ############################
```

```
        # Compute portfolio value
        p_values = np.dot(data_prices[day_ind_start:day_ind_end+1,:], x[strategy, period-1]) + cash[strategy, period-1]
        portf_value[strategy][day_ind_start:day_ind_end+1] = np.reshape(p_values, (p_values.size,1))
        print('  Strategy "{0}", value begin = $ {1:.2f}, value end = $ {2:.2f}'.format( strategy_names[strategy],
               portf_value[strategy][day_ind_start][0], portf_value[strategy][day_ind_end][0]))


        # Compute expected returns and covariances for the next period
        cur_returns = data_prices[day_ind_start+1:day_ind_end+1,:] / data_prices[day_ind_start:day_ind_end,:] - 1
        mu = np.mean(cur_returns, axis = 0)
        Q = np.cov(cur_returns.T)

# Plot results
###################### Insert your code here ############################
```