Data Driven Methods in Finance:

Portfolio Weights

Fall 2023: IEOR 4576

Naftali Cohen

Introduction

- Portfolio managers need to assign relative weights to stocks in a portfolio rather than merely selecting "good" or "bad" stocks.
- Stock weighting can range from simple methods, like equal weighting or market capitalization weighting, to complex methods derived from modern portfolio theory.
- When managing portfolios against a benchmark, managers have multiple weighting possibilities, including matching weighted-average factor exposures or maximizing expected return while limiting tracking error.
- The presentation covers mathematical and statistical aspects of portfolio weighting, discussing ad-hoc methods and the mean-variance optimization method, and comparing four approaches for portfolios managed against a benchmark.

Mean-variance optimization

- Mean-variance optimization (MVO) uses quadratic programming to find the portfolio with the lowest risk for a given expected return, based on the mean and variance of future stock returns.
- The method involves theoretically computing and comparing the risk and expected return of **all possible portfolios** from individual stock return data.
- A challenge with MVO is that it may assign **large weights to outlier stocks**, potentially due to estimation errors in stock means and variances.
- Portfolio managers can address this by **adding constraints** during optimization, such as short-sale constraints, diversification constraints, and sector constraints.
- Care must be taken when adding constraints to ensure they don't conflict with one another, and only the most essential constraints should be imposed.

Mean-Variance Optimization Without Constraints

- Mean-variance optimization (MVO) finds the portfolio with the lowest risk for a given expected return.
- **Setup**: Use a vector **μ** for expected stock returns and a matrix **Σ** for variances and covariances of stock returns.
- **Portfolio Specification**: Defined by a weight vector **w**, summing to 1, representing the weight of each stock in the portfolio.
- **Optimization Problem**: The task is to minimize the portfolio's risk **w'Σw** for a set expected return **w'μ** = μ_P.
- **Solution Method**: Quadratic programming is used to solve this quadratic optimization problem.

Mean-Variance Optimization With Equality Constraints

- General constraints are written as: $\mathbf{Aw=b}$.
- Minimize the objective $\mathbf{w'\Sigma w}$ subject to the equality constraints $\mathbf{Aw=b}$ results in a closed form solution: $\mathbf{w=\Sigma^{-1}A'(A\Sigma^{-1}A')^{-1}b}$

Solve analytically and numerically

Given:

Sigma = np.array([[452, 249 , 189, 70, 481 , 106],
[249, 1094, 356, 93, 1216, 135],
[189, 356 , 617, 161, 1304, 110],
[70 , 93 , 161, 372, 462, 107],
[481, 1216, 1304, 462, 5658, 425],
[106 , 135, 110, 107, 425 , 244]])

A = np.array([[1,1,1,1,1,1], [14,10,9,7,20,2]])
b = np.array([1, 8])

Use the solution: $\mathbf{w=\Sigma^{-1}A'(A\Sigma^{-1}A')^{-1}b}$

Solve:

min $\mathbf{w'\Sigma w}$

s.t. $\mathbf{Aw=b}$

Solution:

```
import numpy as np
from numpy.linalg import inv
import cvxpy as cp

#Analytic solution:
w = inv(Sigma) @ A.T @ inv( A @ inv(Sigma) @ A.T)
@ b
print(w, w.T @ Sigma @ w)
```

```
import numpy as np
from numpy.linalg import inv
import cvxpy as cp

#Numerical solution:
N = 6
w = cp.Variable(N)
risk = cp.quad_form(w, Sigma)
prob = cp.Problem(cp.Minimize(risk), [A@w == b])
prob.solve(solver=cp.SCS)

print(w.value, prob.value)
```

w = [0.34693742 0.0600378 0.18721722 0.28894524 -0.07771101 0.19457333] 160.73067023861194
457333]
w'Σw = 160.73067023861194

Short-sale and diversification constraints

- Short-sale Restrictions: Portfolio managers often face constraints like not being allowed to short securities. This requires weights for each stock to be at least zero ($\mathbf{w \geq 0}$) and adds complexity to the optimization problem.
- Efficient Frontier Shift: When short-sale constraints are added, the efficient frontier shifts to the right, indicating that minimum risk portfolios will inherently have **higher risk**.
- Constrained vs. Unconstrained Portfolios: A constrained portfolio might have **higher risk and fewer stocks** than its unconstrained counterpart but will adhere to regulations and practical needs of a long-only portfolio manager.
- Diversification Constraints: In addition to short-sale restrictions, managers might incorporate diversification limits ($\mathbf{w^B \leq w \leq w^T}$) to ensure a balanced exposure to stocks, reducing diversifiable risk.

Sector, Industry, and Trading Volume constraints

- **Sector or Industry Constraints**: Managers, especially those aligning with benchmarks, often adjust portfolios to ensure specific sector weightings are maintained ($w^B_j \leq w \leq w^T_j$).
- **Trading-Volume Constraint**: For large portfolios, like those valued at $500 million, managers introduce constraints to prevent their transactions from significantly impacting stock prices.
- **Example of Trading Constraint**: A manager may want a stock holding to be less than 10% of the stock's average daily trading volume (ADV). If a stock's portfolio weight is represented by w_i and its average trading volume is x_i (in millions of dollars), then the constraint is set such that $500w_i \leq 0.1x_i$.
- **General Constraint Expression**: This trading volume constraint can be broadly expressed with a formula involving the stock's portfolio weight, its average daily trading volume in dollars, and a predefined threshold ($w_i \leq cx_i$).

Risk-Adjusted Return

- **Risk-Adjusted Return vs. Traditional Methods**: While traditionally mean-variance optimization has been about risk minimization, some managers prefer maximizing expected returns based on their specific goals: max $\mathbf{w'\mu}$, s.t.$\mathbf{w'\Sigma w}=\sigma_P$
- **Expected Return Maximization**: Useful when a portfolio manager has a predetermined target risk level but no specific expected return.
- **Risk-Aversion Parameter (A)**: This represents how much risk a manager is willing to take. A high A value suggests a manager views risk as costly. For instance, an A value of 2 means equating a 1% variance increase to a 2% expected return decrease.
- **Formulation Utility**: Adjusting expected returns for risk, through parameters like A, provides a versatile approach beneficial in various portfolio management scenarios: max $\mathbf{w'\mu}$-A$\mathbf{w'\Sigma w}$

Benchmark

- **Portfolio Management Styles**:
  - Managers can closely **follow** a benchmark (index managers) or
  - manage more loosely aiming to **outperform** the benchmark (active or enhanced index managers).
- **Balancing Act**: Active managers aim to outperform a benchmark without straying too far from its structure, especially if the benchmark isn't inherently efficient.
- **Tools for Tracking**: Various methods exist to align portfolios with benchmarks, including (1) factor exposure targeting and (2) tracking error minimization, the latter being favored for its balance of risk control and stock selection freedom.

Benchmark: Ad Hoc, and Stratification

- **Simple Ad Hoc Approach**: A basic method to create a benchmarked portfolio is to select the top holdings of the benchmark by market capitalization and adjust their weights based on preferred stocks.
    - **Z-Score Adjustment**: Portfolio managers can tweak the relative market capitalization weighting using the aggregate Z-score methodology, adjusting weights based on a computed multiplier.
- **Stratification Method**: Stratification, or stratified sampling, is a technique where stocks are divided into non-overlapping groups, often by industry or size, ensuring broad diversification. The idea is to select representative stocks (often at random) from each group, allowing portfolio managers to focus on high-α stocks while still controlling risk relative to the benchmark.

Benchmark: Factor exposure targeting

- **Factor Exposure Targeting**: This approach aligns the portfolio with the benchmark by setting the benchmark's factor exposures as the target for the portfolio, often aiming for a portfolio's overall beta close to 1 with respect to the benchmark.
- **Benchmark Beta**: The beta of any portfolio is the weighted average of individual stock betas. By adding constraints, a portfolio can ensure its beta aligns with the benchmark's desired range. For example, $0.75 \leq \mathbf{w}'\boldsymbol{\beta} \leq 1.25$.
- **Effect of Constraints**: Constraints can lead to different portfolio compositions and may increase variances. In the given example, portfolios with a beta constraint had higher variances compared to unconstrained portfolios.
- **Factor Tilting**: Beyond just beta, a portfolio manager can also adjust (or tilt) other factor exposures in the portfolio. This allows for alignment with specific market views or strategies, such as favoring growth investments by setting a minimum exposure to the growth factor.

Benchmark: Tracking-error minimization

- **Tracking-Error Minimization**: Professional portfolio managers often use this approach to weight stocks and build a portfolio. The goal can be to either minimize tracking error for a specified excess return over the benchmark or maximize the excess return without exceeding a set tracking-error constraint.
- **Defining Tracking Error (TE)**: It is typically defined as the **standard deviation of the difference** between portfolio returns and benchmark returns: TE = $std(r_P - r_B)$
- $Var(r_P - r_B) = Var(r_P) - 2C(r_P, r_B) + Var(r_B) \approx Var(r_P) - 2C(r_P, r_B)$

To minimize the tracking error:

$$\min \mathbf{w'\Sigma w} - 2\mathbf{w'\gamma} \text{ s.t. } \mathbf{w'\mu} = \mu_P = \mu_B + \delta$$

where $\boldsymbol{\gamma} = [C(r_1, r_B), C(r_2, r_B), \ldots C(r_n, r_B)]$

Benchmark: Tracking-error minimization

- To minimize the tracking error:

$$\min \mathbf{w'\Sigma w} - 2\mathbf{w'\gamma} \text{ s.t. } \mathbf{w'\mu} = \mu_P = \mu_B + \delta$$

$$\text{where } \boldsymbol{\gamma} = [C(r_1, r_B), C(r_2, r_B), \ldots C(r_n, r_B)]$$

- Other forms:
  - Risk-Adjusted w/ target tracking error: max $\mathbf{w'\mu}$ s.t. $\text{Var}(r_P - r_B) = \sigma^2_x$
  - Risk-Adjusted w/ tracking-error aversion parameter: max $\mathbf{w'\mu} - A\,\text{Var}(r_P - r_B)$

Quick optimization review: simple linear programming

To produce one unit of Product A, the company requires 2 hours of labor and 3 pounds of material. For Product B, it takes 3 hours of labor and 2 pounds of material to produce one unit. The company has a total of 80 hours of labor and 100 pounds of material available, which has to be shared between the two products. The profit from each unit of Product A is $5, and the profit from each unit of Product B is $4. The objective is to determine the optimal number of units of Product A and Product B to produce, in order to maximize the company's profit, while considering the constraints in labor and material availability.

- **Objective Function (to be maximized):** Profit = $5x_1 + 4x_2$
  - where x_1 is the number of units of Product A and x_2 is the number of units of Product B.
- **Constraints:**
  - Labor constraint: $2x_1 + 3x_2 \leq 80$
  - Material constraint: $3x_1 + 2x_2 \leq 100$
- **Non-negativity constraints:**
  - $x_1 \geq 0$
  - $x_2 \geq 0$

Quick optimization review: simple linear programming

```python
import cvxpy as cp

# 1. Define the decision variables
x1 = cp.Variable() # Number of units of Product A
x2 = cp.Variable() # Number of units of Product B

# 2. Define the objective function
profit = 5*x1 + 4*x2
objective = cp.Maximize(profit)

# 3. Define the constraints
constraints = [
    2*x1 + 3*x2 <= 80,    # Labor constraint
    3*x1 + 2*x2 <= 100,   # Material constraint
    x1 >= 0,              # Non-negativity constraint for x1
    x2 >= 0               # Non-negativity constraint for x2
]

# 4. Solve the problem
problem = cp.Problem(objective, constraints)
problem.solve()

# 5. Display the results
print("Optimal profit:", problem.value)
print("Optimal number of units for Product A (x1):", x1.value)
print("Optimal number of units for Product B (x2):", x2.value)
```

```
Optimal profit: 171.9999999873071
Optimal number of units for Product A (x1): 28.00000000440914
Optimal number of units for Product B (x2): 7.999999991315351
```

```python
from gurobipy import Model, GRB

# 2. Initialize a new model
m = Model("Profit Maximization")

# 3. Define the decision variables
x1 = m.addVar(vtype=GRB.CONTINUOUS, name="x1")  # Number of units of Product A
x2 = m.addVar(vtype=GRB.CONTINUOUS, name="x2")  # Number of units of Product B

# 4. Define the objective function
m.setObjective(5*x1 + 4*x2, GRB.MAXIMIZE)

# 5. Add the constraints to the model
m.addConstr(2*x1 + 3*x2 <= 80, "Labor Constraint")
m.addConstr(3*x1 + 2*x2 <= 100, "Material Constraint")
m.addConstr(x1 >= 0, "Non-negativity Constraint for x1")
m.addConstr(x2 >= 0, "Non-negativity Constraint for x2")

# 6. Solve the model
m.optimize()

# 7. Display the results
if m.status == GRB.Status.OPTIMAL:
    print("Optimal profit:", m.objVal)
    print("Optimal number of units for Product A (x1):", x1.x)
    print("Optimal number of units for Product B (x2):", x2.x)
else:
    print("No solution found!")
```

Quick optimization review: binary linear programming

$$\begin{aligned} \min \quad & x_1 + x_2 + x_3 \\ \text{s.t.} \quad & x_1 + x_2 \geq 2 \\ & x_2 + x_3 \leq 1 \\ & x_1, x_2, x_3 \text{ are binary} \end{aligned}$$

Quick optimization review: binary linear programming

$$\begin{aligned} \min \quad & x_1 + x_2 + x_3 \\ \text{s.t.} \quad & x_1 + x_2 \geq 2 \\ & x_2 + x_3 \leq 1 \\ & x_1, x_2, x_3 \text{ are binary} \end{aligned}$$

```python
import cvxpy as cp

# 2. Define the decision variables
x1 = cp.Variable(boolean=True) # binary variable
x2 = cp.Variable(boolean=True) # binary variable
x3 = cp.Variable(boolean=True) # binary variable

# 3. Define the objective function
objective_function = x1 + x2 + x3
objective = cp.Minimize(objective_function)

# 4. Define the constraints
constraints = [
    x1 + x2 >= 2,
    x2 + x3 <= 1
]

# 5. Solve the problem
problem = cp.Problem(objective, constraints)
problem.solve()

# 6. Display the results
print("Minimum value of the objective function:", problem.value)
print("Optimal value for x1:", x1.value)
print("Optimal value for x2:", x2.value)
print("Optimal value for x3:", x3.value)
```

```
Minimum value of the objective function: 2.0
Optimal value for x1: 1.0
Optimal value for x2: 1.0
Optimal value for x3: 0.0
```

Quick optimization review: integer linear programming

max $3x_1 + 2x_2$

s.t. $x_1 + x_2 \leq 4$

$x_1 \geq 0$

$x_2 \geq 0$

x_1, x_2 are integers

Quick optimization review: integer linear programming

$$\max \quad 3x_1 + 2x_2$$

$$\text{s.t.} \quad x_1 + x_2 \leq 4$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

$$x_1, x_2 \text{ are integers}$$

```python
import cvxpy as cp

# 2. Define the decision variables
x1 = cp.Variable(integer=True) # integer variable
x2 = cp.Variable(integer=True) # integer variable

# 3. Set the objective function
objective_function = 3*x1 + 2*x2
objective = cp.Maximize(objective_function)

# 4. Define the constraints
constraints = [
    x1 + x2 <= 4,
    x1 >= 0,
    x2 >= 0
]

# 5. Solve the problem
problem = cp.Problem(objective, constraints)
problem.solve()

# 6. Display the results
print("Maximum value of the objective function:", problem.value)
print("Optimal value for x1:", x1.value)
print("Optimal value for x2:", x2.value)
```

```
Maximum value of the objective function: 12.0
Optimal value for x1: 4.0
Optimal value for x2: 0.0
```

Quick optimization review: Quadratic Programming

$$\min \quad x^2_1 + x^2_2$$

$$\text{s.t.} \quad x_1 + x_2 \geq 1$$

Quick optimization review: Quadratic Programming

$$\min \quad x^2_1 + x^2_2$$

$$\text{s.t.} \quad x_1 + x_2 \geq 1$$

```python
import cvxpy as cp

# 2. Define the decision variables
x1 = cp.Variable()
x2 = cp.Variable()

# 3. Set the objective function
objective_function = cp.square(x1) + cp.square(x2)
objective = cp.Minimize(objective_function)

# 4. Define the constraints
constraints = [
    x1 + x2 >= 1
]

# 5. Solve the problem
problem = cp.Problem(objective, constraints)
problem.solve()

# 6. Display the results
print("Minimum value of the objective function:", problem.value)
print("Optimal value for x1:", x1.value)
print("Optimal value for x2:", x2.value)
```

```
Minimum value of the objective function: 0.5000000000000002
Optimal value for x1: 0.5000000000000001
Optimal value for x2: 0.5000000000000001
```

Quick optimization review: Quadratic Programming

max $w'\mu - Aw'\Sigma w$

s.t. $\Sigma x = 1$

$x \geq 0$

set N=5, random risk and return,

set gamma to 0.5

Quick optimization review: Quadratic Programming

max $w'\mu - Aw'\Sigma w$

s.t. $\Sigma x = 1$

$x \geq 0$

set N=5, random risk and return,

set gamma to 0.5

```python
import cvxpy as cp
import numpy as np

# Sample data for μ and Σ
n = 5  # Sample size
mu = np.random.rand(n, 1)
Sigma = np.random.rand(n, n)
Sigma = Sigma.T @ Sigma  # Making Sigma symmetric

# 2. Define the decision variables
x = cp.Variable(n)

# 3. Set the objective function
gamma = 0.5
objective_function = mu.T @ x - gamma * cp.quad_form(x, Sigma)
objective = cp.Maximize(objective_function)

# 4. Define the constraints
constraints = [
    cp.sum(x) == 1,
    x >= 0
]

# 5. Solve the problem
problem = cp.Problem(objective, constraints)
problem.solve()

# 6. Display the results
print("Maximum value of the objective function:", problem.value)
print("Optimal values for x:", x.value)
```

```
Maximum value of the objective function: 0.11533463565277624
Optimal values for x: [-1.72425186e-23  9.99204996e-01 -9.29515139e-24  4.88051785e-23
  7.95004327e-04]
```

Disclaimer

This course is for educational purposes only and does not offer investment advice or pre-packaged trading algorithms. The views expressed herein are not representative of any affiliated organizations or agencies. The main objective is to explore the specific challenges that arise when applying Data Science and Machine Learning techniques to financial data. Such challenges include, but are not limited to, issues like short historical data, non-stationarity, regime changes, and low signal-to-noise ratios, all of which contribute to the difficulty in achieving consistently robust results. The topics covered aim to provide a framework for making more informed investment decisions through a systematic and scientifically-grounded approach.