Finance II Project 2

Isaac Clark

April 25, 2025

Hierarchical Risk Parity (HRP) Approach

Step one involves tree clustering from the correlation matrix of a portfolio. First, after calculating the correlation matrix $\{\rho_{i,j}\}$, we convert it to a distance matrix $\{d_{i,j}\}$ with the function $d_{i,j} = \sqrt{\frac{1}{2}(1-\rho_{i,j})}$.

Second, we compute the euclidean distance between any two column vectors of $\{d_{i,j}\}$ in the usual way to get the distance matrix $\{\tilde{d}_{i,j}\}$ so that all distances are defined over the entire matrix space of $\{d_{i,j}\}$ rather than just a particular correlation pair.

Third, we cluster together a pair of columns such that the pair satisfies $\text{argmin}_{(i,j),i\neq j}\{\tilde{d}_{i,j}\}$ and denote the cluster as $u[1]$.

Fourth, we need to find the distance between the cluster $u[1]$ and the other unclustered items, in what is called "linkage criterion," by using the nearest point algorithm. We donote the new disance vector as $\{\dot{d}_{i,u[1]}\}$.

Fifth, the matrix $\{\tilde{d}_{i,j}\}$ is updated by appending $\{\dot{d}_{i,u[1]}\}$ and dropping the columns and rows $j \in u[1]$

Sixth, we recusively repeat the previous to append all other clusters, at which point the final cluster contains all of the original items and we stop the tree clustering algorithm. This allows us to define a linkage matrix where each item reports a different aspect of the correlation matrix useful for our analysis.

We move onto the next step two, which is to "quasi-diagonalize" the covariance matrix. That is, we reorganize the matrix so that largest values lie along the diagonal. This is done so that similar investments are placed together, and dissimilar investments are placed far apart.

In the final step, with the quasi-diagonalized covariance matrix in hand, we apply recusive bisection. This takes advantage of the fact that inverse-variance allocation is optimal for a diagonal covariance matrix and is used in two ways, bottom-up, to find the variance of a contiuous subset, and top-down, to split allocations between adjacent subsets. At this point, we have the optimal weights for each of the assets in our portfolio.

Efficient Frontier and HRP Portfolios from PyPortfolioOpt

Stocks chosen for the portfolio analysis were as follows:

- Apple (AAPL)
- Microsoft (MSFT)

- Amazon (AMZN)
- Johnson & Johnson (JNJ)
- JPMorgan Chase (JPM)
- Exxon Mobil (XOM)
- Pfizer (PFE)
- NVIDIA (NVDA)
- Walmart (WMT)
- Disney (DIS)

The historical performance data for each stock over the last three months was pulled into the python environment using the YFinance package. Importing the EfficientFrontier and HRPOpt assets from PyPortfolioOpt, we calculate the optimal portfolios under each algorithm and simulate the performance over five days by taking the average daily return from 1000 simulations for each portfolio.

Portfolio Allocation		
Ticker	Efficient Frontier	Hierarchical Risk Parity
AAPL	0%	0.3%
MSFT	0%	0.4%
AMZN	0%	0.4%
JNJ	100%	4.9%
JPM	0%	1.0%
XOM	0%	10.5%
PFE	0%	76.8%
NVDA	0%	1.0%
WMT	0%	3.8%
DIS	0%	1.0%

5 Day Simulated Portfolio Performance		
Day	Efficient Frontier	Hierarchical Risk Parity
1	1.0	1.0
2	1.0016	1.0008
3	1.0006	1.0023
4	1.0023	1.0028
5	1.0023	1.0032

We can see from the tables that the HRP method is much better at creating a diverse portfolio from the given assets than the Efficient Frontier method. Observing the daily performance, both portfolios perform similarly well, however, the HRP portfolio clearly has less variability than the Efficient Frontier portfolio and experiences more linear growth. As such, the HRP method seems to be better at finding the optimal portfolio and creating more predictable returns.

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from pypfopt import EfficientFrontier, risk_models, expected_returns,
    HRPOpt
from datetime import datetime, timedelta
import warnings

warnings.filterwarnings("ignore")

tickers = ["AAPL", "MSFT", "AMZN", "JNJ", "JPM", "XOM", "PFE", "NVDA",
     "WMT", "DIS"]
end_date = datetime.today()
start_date = end_date - timedelta(days=90)

data = yf.download(tickers, start=start_date, end=end_date)['Close']
data.dropna(inplace=True)

mu = expected_returns.mean_historical_return(data)
S = risk_models.sample_cov(data)

ef = EfficientFrontier(mu, S)
ef_weights = ef.max_sharpe()
ef_cleaned_weights = ef.clean_weights()
ef_perf = ef.portfolio_performance(verbose=False)

hrp = HRPOpt(data)
hrp_weights = hrp.optimize()
hrp_perf = hrp.portfolio_performance(verbose=False)

ef_df = pd.DataFrame.from_dict(ef_cleaned_weights, orient='index',
    columns=['Efficient Frontier'])
hrp_df = pd.DataFrame.from_dict(hrp_weights, orient='index', columns=[
    'HRP'])
weights_df = ef_df.join(hrp_df)

num_simulations = 1000
num_days_forward = 5

ef_sim_results = np.zeros((num_days_forward, num_simulations))
hrp_sim_results = np.zeros((num_days_forward, num_simulations))

last_prices = data.iloc[-1].values

for sim in range(num_simulations):
    future_returns = np.random.normal(loc=0.0005, scale=0.02, size=(
        num_days_forward, len(tickers)))

    future_prices = last_prices * np.exp(np.cumsum(future_returns,
        axis=0))
    future_prices_df = pd.DataFrame(future_prices, columns=tickers)
```

```
    ef_vals = future_prices_df @ weights_df['Efficient Frontier']
    hrp_vals = future_prices_df @ weights_df['HRP']

    ef_sim_results[:, sim] = ef_vals
    hrp_sim_results[:, sim] = hrp_vals

ef_avg = np.mean(ef_sim_results, axis=1)
hrp_avg = np.mean(hrp_sim_results, axis=1)

ef_avg /= ef_avg[0]
hrp_avg /= hrp_avg[0]

perf_df = pd.DataFrame({
    'EF': ef_avg,
    'HRP': hrp_avg
})

ef_vals = future_prices_df @ weights_df['Efficient Frontier']
hrp_vals = future_prices_df @ weights_df['HRP']

print(weights_df)
print(perf_df)
```