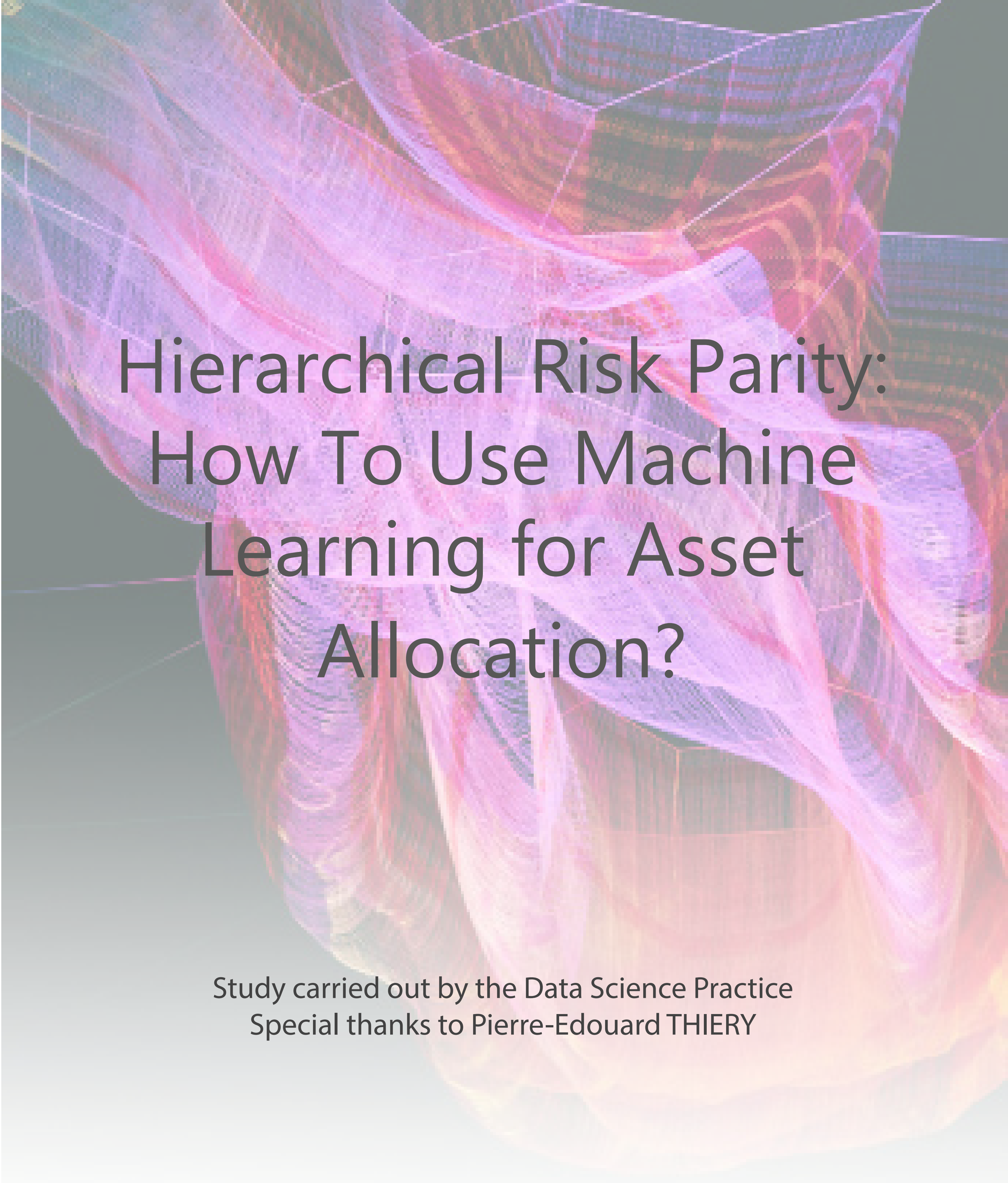

Hierarchical Risk Parity: How To Use Machine Learning for Asset Allocation?

Study carried out by the Data Science Practice
Special thanks to Pierre-Edouard THIERY

coperneec

Mars 2021

Summary

Introduction

When it comes to asset allocation, the mathematical framework introduced by Henri Markowitz [1] is still ubiquitous and widely used: the allocation question can be seen as mathematical problem of optimization under constraints. The main conclusion which can be drawn from this work is that diversification plays a significant role to improve the performance of a portfolio, while reducing its risk. From a mathematical point of view, if there exist several formulations of the optimization problem, they all share one commonality: a quadratic optimizer is necessary in order to estimate the optimal solution. Markowitz himself devised a new algorithm, called the Critical Line Algorithm (CLA), to solve such optimization problems.

However, in real life, the CLA is very unstable, insofar as the optimal solutions it reaches are not stable: a small difference in the input can lead to huge differences in the portfolio which is ultimately allocated. This is particularly the case in Markowitz's approach since the expected returns plays a significant role: we try to find the best trade-off between the expected return of the overall portfolio and its risk. It is highly difficult to forecast the returns of an asset, and this explains why new methods were required to allocate wealth among several assets.

Since returns are difficult to estimate, some authors decided to disregard the expected returns to focus only the risk of the portfolio. Risk-parity portfolio construction epitomized this new paradigm [2]. However, once again, the mathematical solutions of optimization problems based only on risk are highly unstable because of the role the covariance matrix plays in the optimization algorithm. This matrix must be inverted, and such a process is very unstable, especially when the assets are highly correlated. This is precisely the case where diversification is the most needed.

In this paper we set forth a new way of allocating wealth among assets, based on Machine Learning and Graph Theory. Its purpose is to circumvent the numerical instabilities of the traditional quadratic approaches, whether they are based on mean-variance or only on risk. After explaining why the traditional quadratic approaches are intrinsically flawed, we present a new Machine Learning-based approach as well as an implementation of this approach.

1 From The Flaws Of Traditional Quadratic Optimizers To Graph Theory

We have already mentioned why Markowitz's approach is unstable when implemented in real life: it is extremely difficult to estimate the expected returns of an asset with sufficient accuracy. The CLA can then reach very different solutions when a small change occurs in the input. The idea of disregarding returns to focus only on risk, albeit quite interesting, is nonetheless not sufficient to avoid numerical instabilities. Indeed, the covariance matrix plays a significant role in quadratic optimizer, especially in finance insofar as it enables to compute the risk of a given portfolio. Indeed quadratic programming methods require the inversion of this matrix, which must be supposed to be positive-definite.

If the covariance matrix is very simple, for instance merely diagonal, there is no difficulty in inverting it. When the matrix becomes more complex, for instance when the assets display many correlation features in finance, it turns out that inverting the covariance matrix becomes a very unstable process [3]. A small change in the estimation of the correlations can lead, once again, to very different portfolios. It then required even more data to estimate the correlations with the utmost accuracy: for a matrix of size N, at least $\frac{1}{2}N(N+1)$ independent and identically distributed (IID) observations are needed to make sure that the estimated covariance matrix is not singular. If $N = 50$, this means that at least 5 years of daily IID are required: it is very unlikely that the correlation structures remain invariant over such a long time frame.

Those mathematical questions explain why it has been possible, over long periods of time, to beat mean-variance portfolios and risk-parity portfolios with simple portfolios, such as equally-weighted portfolios [4].

In order to address the limitations of an approach based on the inversion of the covariance matrix, it is necessary to take a look at how our data are represented. The traditional approaches rely heavily on linear algebra. The return series can be seen as the vectors of a vector space. Within such a representation, every vector can be eligible to play the role of another. The vectors can be viewed as a fully connected graph, there is no hierarchy among the vectors. This complexity partially explains why inverting the covariance matrix is unstable.

Furthermore, this no-hierarchy approach is not in line with popular wisdom. When an investor desires to build a diversified portfolio, first he is going to consider the various asset classes: stocks, bonds, real estate etc. Then, among each asset class, he is going to consider some sub-categories: office real estate, residential real estate etc. Stocks for instance can be classified depending on their liquidity, size and region. This means that some assets are equivalent and can be seen as similar, whereas others are not. If the investor decides to invest a fraction of his/her wealth in a large publicly traded company in the banking sector, it is similar to invest either in BNP Paribas or in Société Générale, but this is not equivalent to investing in a small private Asian bank.

So investment is based on hierarchy, and the lack of hierarchy within traditional quadratic approaches explains the instability: the weights can vary in all possible directions. It would much better to make sure that our data, instead of being seen as a fully connected graph, are represented with a kind a tree structure.

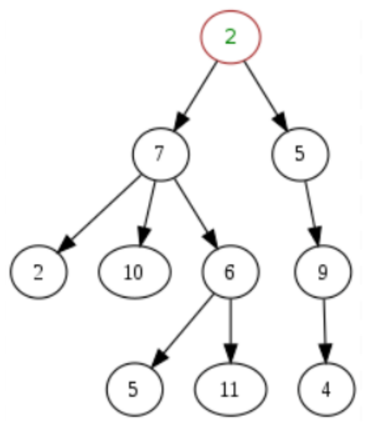

Figure 1: Tree structure

In the rest of this paper, we set forth a new method for asset allocation, called Hierarchical Risk Parity. This method is based only on the covariance-correlation matrix, this is why its name refers to "risk-parity", but it does not require the brutal inversion of the covariance matrix. Instead it makes the most of the hierarchical structure within the data. This new method is made of three steps: tree clustering, quasi-diagonalization and recursive bisection.

2 Tree Clustering: When An Asset Is Not Necessarily Similar To Another

As of now, we consider N assets, denoted X_i with $i = 1 \dots N$, whose returns are observed over T periods of time, for $t = 1 \dots T$. Generally, we consider the daily returns for each asset: r_t^i is the daily return between day $t-1$ and day t for asset X_i.

With those data, it is possible to compute the correlation matrix C, of size $N \times N$, thanks to the common correlation estimator:

$$\rho(X_i, X_j) = \rho_{i,j} = \frac{\frac{1}{T}\sum_{t=1}^{T}\left(r_t^i - \bar{r^i}\right)\left(r_t^j - \bar{r^j}\right)}{\sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(r_t^i - \bar{r^i}\right)^2}\sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(r_t^j - \bar{r^j}\right)^2}}$$

with $\bar{r^i} = \frac{1}{T}\sum_{t=1}^{T} r_t^i$. $C = \left(\rho_{i,j}\right)_{1 \le i,j \le N}$

Thanks to the correlation matrix C, we can define a distance between two assets X_i and X_j:

$$d\left(X_i, X_j\right) = \sqrt{\frac{1}{2}(1-\rho_{i,j})}$$

Each asset X_i can be seen as a vector of size T, i.e. the vector of its returns. The distance d is then a true mathematical distance: $d\left(X,Y\right) \ge 0$, $d\left(X,Y\right) = 0 \Leftrightarrow X = Y$ and $d\left(X,Y\right) \le d\left(X,Z\right) + d\left(Z,Y\right)$.

It is now possible to define a new matrix of size $N \times N$, a distance matrix D: $D = \left(d_{i,j}\right)$.

If we consider a numerical example in dimension 3 and if the correlation matrix is equal to:

$$C = \begin{pmatrix} 1 & -0.003036 & 0.010026 \\ -0.003036 & 1 & 0.008572 \\ 0.010026 & 0.008572 & 1 \end{pmatrix}$$

the distance matrix is equal to

$$D = \begin{pmatrix} 0 & 0.708179 & 0.703553 \\ 0.708179 & 0 & 0.704070 \\ 0.703553 & 0.008572 & 0 \end{pmatrix}$$

Each asset X_i can be assimilated to the $i-th$ column of matrix D: the asset X_i is no longer defined by its return series, but thanks to the distances between X_i itself and all the other assets X_j. Those quantities exactly correspond to the $i-th$ column of D. As of now, we use notations X_i and D_i for asset i.

We then compute the Euclidian distance between two of the columns of D:

$$\tilde{d}_{i,j} = \tilde{d}\left(D_i, D_j\right) = d_E\left(D_i, D_j\right) = \sqrt{\sum_{k=1}^{N}\left(d_{k,i} - d_{k,j}\right)^2}$$

Therefore we can now define a third $N \times N$ matrix $\tilde{D}$ by: $\tilde{D} = \left(\tilde{d}_{i,j}\right)$. So the element (i,j) of matrix $\tilde{D}$ measures the distance between asset X_i and X_j, but this distance includes information for the entire correlation matrix C, whereas $\rho_{i,j}$ only included information from the two assets X_i and X_j. With our numerical example:

$$\tilde{D} = \begin{pmatrix} 0 & 1.21903 & 0.99498 \\ 1.21903 & 0 & 0.704137 \\ 0.99498 & 0.704137 & 0 \end{pmatrix}$$

Once the distance matrix $\tilde{D}$ has been computed, it is possible to start clustering together the assets of our universe. To do so, we start by finding the pair of assets $\left(i^\star, j^\star\right)$ such that

$$\left(i^\star, j^\star\right) = \arg\min_{i \ne j}\left\{\tilde{d}_{i,j}\right\}$$

So, with our example, we see that $\left(i^\star, j^\star\right) = \left(2,3\right)$. It means that assets X_2 and X_3 are so close that they can actually be considered to be similar: they form the first cluster $cl\left(1\right) = \left(2,3\right)$.

Once a cluster has been identified, we can compute the distance between every asset X_i and this cluster: for $1 \le i \le N$:

$$u_{i,cl(1)} = \min_{j \in cl(1)}\left\{\tilde{d}_{i,j}\right\}$$

This defines a vector $\vec{u} \in \mathbb{R}^N$. Using our numerical example we see that:

$$\vec{u} = \begin{pmatrix} 0.99498 \\ 0 \\ 0 \end{pmatrix}$$

We then append vector $\vec{u}$ to matrix $\tilde{D}$ in the following manner in order to get a new matrix M of size $(N+1) \times (N+1)$:

$$M = \underbrace{\left(\begin{array}{c|c} \tilde{D} & \vec{u} \\ \hline \vec{u}^t & 0 \end{array}\right)}_{N+1}$$

M is actually still a distance matrix: appending $\vec{u}$ is equivalent to considering a new asset within our universe, which is cluster $cl(1)$. The last column of M gives the distance between all the assets of the universe, X_1, X_2, ..., X_N and $cl(1)$ to the cluster $cl(1)$.

Nonetheless, insofar as $X_{i^\star}$ and $X_{j^\star}$ are already contained in cluster $cl(1)$, there is no interest in continuing to consider them in the distance matrix. Thus we can update the distance matrix $\tilde{D}$ by removing lines and columns $i^\star$ and $j^\star$.

$$\tilde{D} \leftarrow (m_{i,j})_{1 \leq i,j \leq N+1,\ and\ i \neq i^\star, i \neq j^\star, j \neq i^\star, j \neq j^\star}$$

With our numerical example, since $cl(1) = \left(2, 3\right)$, this means that $\tilde{D}$, once updated, is equal to:

$$\tilde{D} = \begin{pmatrix} 0 & \bullet & \bullet & 0.99498 \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ 0.99498 & \bullet & \bullet & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0.99498 \\ 0.99498 & 0 \end{pmatrix}$$

Once updated, $\tilde{D}$ contains the distance between the assets X_i, $i \neq i^\star$ and $i \neq j^\star$ and cluster $cl(1)$.

We apply the same process to the new matrix $\tilde{D}$ in order to identify a second cluster $cl(2)$, and we then update once again $\tilde{D}$ and so on and so forth until $\tilde{D}$ is of size 2×2. At this stage, we have already identified $N-2$ clusters, $cl(1)$, $cl(2)$, ..., $cl(N-2)$: we create a final cluster, $cl(N-1)$, which contains the last two assets corresponding to the last two columns of $\tilde{D}$.

In our example, this means that our final cluster is equal to $cl(2) = \left(1, cl(1)\right)$. It is important to notice that $cl(1)$ is now seen as a proper asset, even though it is already a cluster of two other assets.

For each cluster, it is possible to define the number of original items it contains. If the two elements in the cluster are original assets X_i with $1 \leq i \leq N$, then the number of original items contained by the cluster cl is:

$$\mathscr{R}(cl) = 2$$

If the cluster cl contains one original asset and another cluster, denoted cl^a, then:

$$\mathscr{R}(cl) = 1 + \mathscr{R}(cl^a)$$

And, if a cluster is made of two other clusters cl^a and cl^b:

$$\mathscr{R}(cl) = \mathscr{R}(cl^a) + \mathscr{R}(cl^b)$$

We necessarily have $\mathscr{R}(cl(N-1)) = N$.

3 The Shake-Up Of The Correlation Matrix: The Quasi-Diagonalization Step

The purpose of this step is to reorganize the rows and columns of the correlation matrix in order to corral the assets which are similar. This will transform the correlation matrix into a quasi-diagonal matrix where the largest values are placed near the diagonal.

The process is fairly simple: each cluster $cl(k)$ for $1 \leq k \leq N-1$ is made of two elements, either another cluster or one of the original assets. For the rest of this section, we denote those two elements $cl(k)(i)$ with $i = 1, 2$. We start by considering and empty vector $V = \left[\right]$ and the last cluster, $cl(N-1)$. We initialize V in the following manner:

$$V = \left[cl(N-1)(1), cl(N-1)(2)\right]$$

Then, we apply the same process to each element of V. If an element of V is an original asset, it is not replaced, but if it is a cluster, we replace it by the two elements it contains. Since $\mathscr{R}(cl(N-1)) = N$, we are certain that, at the end, V will contain all the original items, but they will appear in a new order. This is the order we use to rewrite the correlation matrix.

We develop in the rest of this section a numerical example which exemplifies the quasi-diagonalization step. We consider ten assets ($N = 10$), which will be referred to with their index: asset 1, asset 2, ..., asset 10. We work with simulated data: we generate daily returns for $T = 10000$ days, and we add a correlation factor among our observations when generating the data.

Figure 2 displays the heat map of the original correlation matrix. The color varies from purple, for the lowest values in the correlation matrix, to yellow, for the highest values. In this representation, it is worth noticing that the correlation matrix has been rotated: for instance $\rho_{1,1}$ lies at the bottom-left of the matrix. This explains why the "yellow" diagonal goes from bottom-left to top-right.

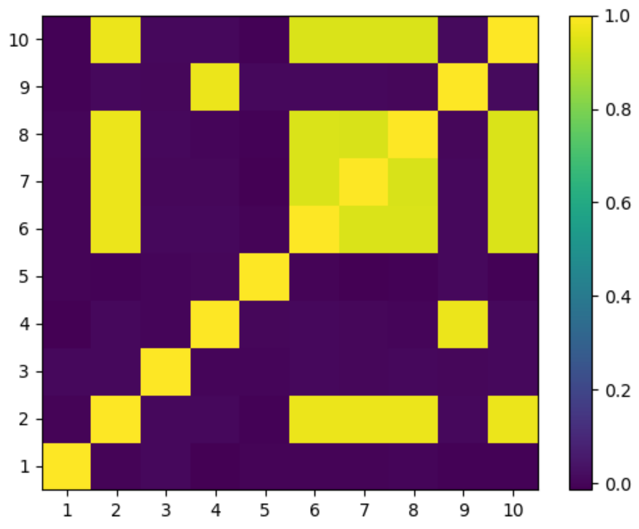

Figure 2: Heat map of the original correlation matrix

If we perform the clustering step, we get the following clusters:

- cl(1) = (4,9)

- cl(2) = (2,10)
- cl(3) = (6, cl(2))
- cl(4) = (7, cl(3))
- cl(5) = (8, cl(4))
- cl(6) = (1, 3)
- cl(7) = (5, cl(6))
- cl(8) = (cl(1), cl(7))
- cl(9) = (cl(5), cl(8))

This gives us the following reordering rule:

$$V = \left[8, 7, 6, 2, 10, 4, 9, 5, 1, 3\right]$$

We can now plot the heat map of the correlation matrix once the rows and columns have been reordered. This is Figure 3. Now, the yellow parts of the matrix are all concentrated close to the diagonal.

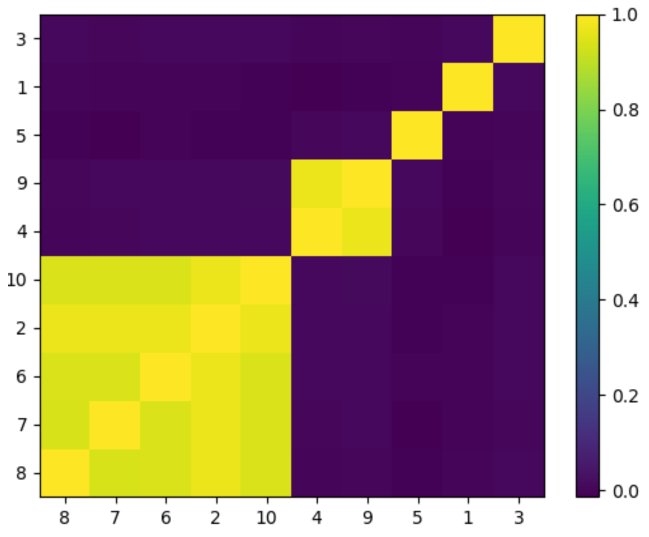

Figure 3: Heat map of the original correlation matrix

4 Allocation Through Recursive Bisection

This final step is premised on the fact that the inverse-variance allocation is optimal for a diagonal matrix. If we consider the following optimization problem with $V = \left(v_{i,j}\right)_{1\leq i,j\leq N}$ a covariance matrix

$$\min_{\vec{w}\in\mathbb{R}^N} \vec{w}^t V \vec{w}$$

with the constraint $\vec{w}^t\vec{1} = 1$, the optimal solution is given by:

$$\vec{w}^\star = \frac{V^{-1}\vec{1}}{\vec{1}^t V^{-1}\vec{1}}$$

If the matrix V is diagonal, there is no need to compute the inverse of V: for $1 \leq n \leq N$

$$w_n = \frac{\frac{1}{v_{n,n}}}{\sum_{k=1}^{n}\frac{1}{v_{k,k}}}$$

We can also notice that, when $N = 2$:

$$w_1 = 1 - \underbrace{\frac{v_{1,1}}{v_{1,1} + v_{2,2}}}_{w_2}$$

All the above results will influence the way we perform the allocation thanks to the quasi-diagonalized correlation matrix. Indeed here is how we perform the recursive bisection.

4.1 Initialization of the recursive bisection

The initialization is fairly simple. We initialize L a list of lists with:

$$L = \{L_0\}$$

where $L_0 = V$, the reordering vector. It is important to notice that, for the rest of the recursive bisection, the order of the elements within L is preserved.

To all assets, we assign the same initial weight: $w_n = 1$ for $1 \leq n \leq N$. After the initialization step, we have only one element in L, denoted L_0, and $Card\left(L_0\right) = N$.

If we use the numerical example at step 2, we have:

$$L = \{\{8, 7, 6, 2, 10, 4, 9, 5, 1, 3\}\}$$

4.2 Iteration of the recursive bisection

First, we set the stopping conditions: if, for all element $L_i \in L$, we have $Card\left(L_i\right) = 1$, then we stop the process.

Then, for each element $L_i \in L$ such that $card\left(L_i\right) > 1$, we bisect it into two subsets:

$$L_i = L_i^1 \cup L_i^2$$

with $Card\left(L_i^1\right) = \left[\frac{1}{2}Card\left(L_i\right)\right]$, while preserving the order of the elements: so L_i^1 contains the first elements of L_i, whereas L_i^2 contains the last elements. What we do actually is only splitting L_i between its first and second halves.

With our numerical example, we have:

$$L_0^1 = \{8, 7, 6, 2, 10\}$$

$$L_0^2 = \{4, 9, 5, 1, 3\}$$

Then, for each subset L_i^j, we define the covariance matrix V_i^j of the assets which are present in L_i^j. Thanks to the quasi-diagonalization step, this means that those two covariance matrices will also be quasi-diagonal.

Indeed, if we reorganize the covariance matrix thanks to the reordering vector, the two covariance matrices we have to consider for L_0^1 and L_0^2 are merely the top-left block and the bottom-right block if we split the reordered covariance matrix into four equal square blocks.

Since V_i^j is quasi-diagonal, we can assimilate it to a diagonal matrix by considering actually the matrix $D_i^j = Diag\left(V_i^j\right)$:

D_i^j is a diagonal matrix, whose diagonal is the same as V_i^j.

We can now solve the same optimization problem as mentioned at the beginning of this section with $V = D_i^j$. So the optimal solution can be written:

$$\vec{w}_i^j = \frac{\left(D_i^j\right)^{-1}\vec{1}}{tr\left(\left(D_i^j\right)^{-1}\right)}$$

We then compute the "optimal variance" for the two subsets L_i^j: $W_i^j = \left(\vec{w}_i^j\right)^t V_i^j \vec{w}_i^j$.

Those optimal variances allow us to compute a split factor α_i which will be applied to our weights. Here, we use the above-mentioned relation in the case of two assets:

$$w_1 = 1 - \frac{v_{1,1}}{v_{1,1} + v_{2,2}}$$

In this simple case, $v_{1,1}$ is the variance of the first asset, and $v_{2,2}$ the variance of the second asset. If we consider that our two subsets L_i^j can be seen as "clustered" assets, we can adapt the formula with the optimal variances:

$$\alpha_i = 1 - \frac{W_i^1}{W_i^1 + W_i^2}$$

α_i is the allocation factor which should be applied to the first "clustered asset", i.e. to all the assets which are present in L_i^1, whereas $1-\alpha_i$ must be applied to the assets which are present in L_i^2.

So for n such that $n \in L_i^1$, we update the weight:

$$w_n \leftarrow w_n \times \alpha_i$$

On the contrary, if n is such that $n \in L_i^2$:

$$w_n \leftarrow w_n \times (1 - \alpha_i)$$

We then replace in the general list L the element L_i by its two subsets L_i^1 and L_i^2. So now:

$$L = \{\ldots, \underbrace{L_i^1, L_i^2}_{previously\ L_i}, \ldots\}$$

and we continue the bisection.

Once the bisection stops, this gives us the weights w_n for each of the original item. For instance, with our numerical example, we get the following weights for the ten original assets: $w_1 = 0.196686$, $w_2 = 0.028694$, $w_3 = 0.19668$, $w_4 = 0.099523$, $w_5 = 0.197634$, $w_6 = 0.052846$, $w_7 = 0.054229$, $w_8 = 0.054119$, $w_9 = 0.092757$ and $w_10 = 0.026843$.

Conclusion

In this paper, we have addressed some of the shortcomings of the traditional asset allocation mathematical framework thanks to Machine Learning, especially Graph Theory. Indeed, as powerful as it is, Markowitz's original work is numerically flawed due to the necessity to forecast the returns of the assets of the universe, before implementing a traditional quadratic optimizers. Such optimizers, since they rely on the inversion of a covariance matrix, are quite unstable when the covariance matrix exhibits lots of correlation between the assets. Since the estimation of the expected returns is highly unstable, the optimal portfolio itself will be highly dependent on the quality of the estimations.

This also explains why alternative approaches, such as risk-parity, which deliberately rule out the expected returns, are unstable as well, even though they are less unstable than Markowitz's original one.

The true issue lies in the way the data are represented. In traditional quadratic optimizers, every asset can be a substitute for any other asset. In real life, this is obviously not the case. Instability is mainly due to the lack of hierarchical structure within the data.

This can be addressed thanks to hierarchical risk parity. This approach consists indeed in structuring the data in clusters, before implementing the allocation. We can then take advantage of the data restructuring to avoid many of the flaws that are inherent of the traditional quadratic optimizers when it comes to asset allocation.

References

[1] H. Markowitz. Portfolio selection. *Journal Of Finance*, 1952.

[2] Jurczenko. *Risk-Based and Factor Investing*. Elsevier Science, 2015.

[3] D. Bailey and M. Lopez de Prado. Balanced baskets: a new approach to trading and hedging risks. *Journal of Investment Strategies*, 2012.

[4] Garlappi De Miguel and Uppal. Estimation of the lead-lag parameter from non-synchronous data. *Review of Financial Studies*, 2009.

A propos de Coperneec

«From Revolution to Performance»

Coperneec est un cabinet de conseil cross-sectoriel spécialiste de la valorisation de la Data. Nous intervenons sur l'ensemble de la chaîne des savoir-faire autour de la Data Science, la Data Analyse et du Data Management.

Nos méthodes et techniques scientifiques éprouvées permettent de résoudre des problématiques dans tous les secteurs de l'industrie.

Notre vocation : extraire la connaissance à partir des données et pérenniser les avancées technologiques qui en découlent. La R&D est au cœur de notre ADN et les expertises de nos consultants (data scientists, data analysts, data engineers) sont en permanence challengées afin d'accompagner au plus près les révolutions technologiques et scientifiques.

Contactez-nous

Aymeric LISBONNE
Partner
alisbonne@coperneec.com
06 88 69 67 75

coperneec

est une marque de

canopee group