Machine Learning in Asset Management: *Part 2: Portfolio Construction—Weight Optimization*

Derek Snow

Derek Snow
is a doctoral candidate of finance at the University of Auckland in Auckland, New Zealand.
d.snow@firmai.org

KEY FINDINGS

- Machine learning can help with most portfolio construction tasks, such as idea generation, alpha factor design, asset allocation, weight optimization, position sizing, and the testing of strategies.
- This is the second of a series of articles dealing with machine learning in asset management and more narrowly weight optimization strategies equipped with machine learning.
- Following from the previous article, different weight optimization methods are considered for supervised, unsupervised, and reinforcement learning frameworks.

ABSTRACT: *This is the second in a series of articles dealing with machine learning in asset management. This article focuses on portfolio weighting using machine learning. Following from the previous article (Snow 2020), which looked at trading strategies, this article identifies different weight optimization methods for supervised, unsupervised, and reinforcement learning frameworks. In total, seven submethods are summarized with the code made available for further exploration.*

TOPICS: ***Big data/machine learning, analysis of individual factors/risk premia, portfolio construction, performance measurement****

In recent years, we have seen a trend toward online or dynamic portfolio asset allocation methods, clustering techniques, and methods that combine metadata on financial assets to help decide on the final weighting. Trading strategies and weight optimization methods can generally be considered as part of an integrated system. A trading strategy here is the use of perceived signals to execute trades, and weight optimization is the optimal weight allocation of active and passive strategies.

Strong parallels can be drawn between weight optimization in machine learning and weight optimization in portfolio management. When working with neural networks, we must often choose which optimization algorithm will produce faster and better updates for the network's weight and bias parameters. This is needed for the internal nodes to learn the optimal relationship in data to minimize a prespecified loss function. Researchers end up testing the performance of various optimizers, such as gradient descent, stochastic gradient descent (SGD), AdaGrad, RMSProp, and Adam, on their dataset. In the process, they might come to conclude that nothing really works that well. After some tinkering, a researcher might add a momentum component to SGD to curb the

*All articles are now categorized by topics and subtopics. **View at PM-Research.com.**

high variance oscillation that makes standard SGD hard to converge. The next researcher might come along and see momentum as a problem when it ceases to stop as it reaches a local minimum. In response, the researcher could suggest a slope jump out of momentum as a reasonable correction.[1]

A development out of this constant tinkering was Adam, a method developed by Kingma and Ba (2014) that performs best by combining the properties of AdaGrad and RMSProp algorithms for a solution that can handle sparse gradients on noisy problems. In a similar spirit, we have seen modern portfolio optimization move from Markowitz's mean–variance (MV) portfolio using historical returns to dynamic models that use state-of-the-art reinforcement learning techniques. This process is not over; in both fields, new methods are brought to light on a frequent basis. Similar to machine learning, the optimal weighting strategy can and should be tested on validation data—in our parlance, back tests. We want to make sure that our data drive our optimization choices.

SUPERVISED LEARNING

Supervised learning techniques learn the relationship between independent attributes and a designated dependent attribute. Supervised learning refers to the mathematical structure describing how to make a prediction y_i given x_i. Instead of learning from the environment, as in reinforcement learning, supervised learning methods learn the relationships in data. All supervised learning tasks are divided into classification or regression tasks. Classification models are used to predict discrete responses (e.g., binary 1, 0; multiclass 1, 2, 3). Regression is used for predicting continuous responses (e.g., 3.5%, 35 times, \$35,000). In portfolio weight optimization, we are generally presented with a continuous response problem.

Deep Portfolio

A useful deep learning technique for finance is autoencoders. This is a learning process to train the architecture to replicate X itself, namely $X = Y$, via a bottleneck structure. An autoencoder can create a more cost-effective representation of X. Autoencoding is an unsupervised learning method because there is no target to predict. However, the low-dimensional encoding is used as an input to a supervised learning method to calibrate and map the inputs to the desired target. The autoencoder demonstrates that, in deep learning, it is not necessary to model the variance–covariance matrix explicitly because the model is already in predictive form. This portfolio optimization process follows four steps: (1) autoencoding, (2) calibrating, (3) validating, and (4) verifying. This method has been popularized by Heaton, Polson, and Witte (2017).

1. **Autoencoding:** Find the market cap, denoted by $F^m_W(X)$, that solves the regularization problem

$$\min_W \left\| X - F^m_W(X) \right\|_2^2 \text{ subject to } \|W\| \le L^m$$

   For appropriately chosen F^m_W, this autoencodes X with itself and creates a more information-efficient representation of X.
2. **Calibrating:** For a desired target Y, find the portfolio map, denoted by $F^p_W(X)$, that solves the regularization problem

$$\min_W \left\| Y - F^p_W(X) \right\|_2^2 \text{ subject to } \|W\| \le L^p$$

   This creates a nonlinear portfolio from X for the approximation of objective Y.
3. **Validating:** Find L^m and L^p to suitably balance the trade-off between the two errors

$$\varepsilon_{m=} \left\| \hat{X} - F^m_{W_m}(\hat{X}) \right\|_2^2 \text{ and } \varepsilon_p = \left\| \hat{Y} - F^p_{W_p}(\hat{X}) \right\|_2^2$$

   where W_m and W_p are the solutions to the validation and verification step
4. **Verifying:** Choose market cap F^m and portfolio map F^p such that the validation step is satisfactory.[2]

Linear Regression

We can also apply linear models to the problem of finding optimal portfolios (Britten-Jones 1999). There is a deep connection between fitting linear models and portfolio optimization. The normal equation is $\hat{\theta} = (X'X)^{-1}X'y$. Note that, in statistics or econometrics, it is more common to see β(beta) instead of θ(theta).

[1] Nesterov accelerated gradient.

[2] Resources: data (https://drive.google.com/open?id=1bJcUZbrZ8HFXs-cd0vGHeMop16Vf3n23); code (https://drive.google.com/open?id=1-hOEAiJqaNTUYIyamj26ZvHJNZq9XV09).

The normal equation makes sense if you do want a linear model and the number of explanatory variables (features) is not too big (e.g., <100,000)—perfect for a small universe of stocks.

It is not necessary to use the normal equation; one can also solve for the best coefficients using numerical search such as gradient descent. With this portfolio optimization method, regularization becomes a critical tool to improve out-of-sample performance. The tested regularization methods include l2-norm (ridge regression), L1-norm (LASSO), and a combination of L1 and L2 norms (elastic net regression). Regularization effectively helps one to manage the variance–bias trade-off. Britten-Jones (1999) demonstrated a close equivalence between Markowitz portfolio optimization and ordinary least squares (OLS) regression.

Specifically, regress the excess returns of N stocks on a constant = 1 without an intercept. Collect the coefficients on each of the stocks in θ. Rescale θ so that the sum of the elements in θ is 1: $(\theta^* = \frac{\theta}{1^T\theta})$. The rescaled coefficients are the Markowitz optimal portfolio weights. As a result, any machine learning approach that yields coefficients for a linear model can be reinterpreted as a portfolio optimization approach (e.g., ridge regression → Tikhonov regularized optimal portfolio).[3]

Bayesian Sentiment

Advances in sentiment mining techniques led researchers in the 2010s to consider the use of public opinion for stock market prediction. Xing et al. (2018) decided to do the same for asset allocation by accounting for public mood using an online Bayesian asset allocation model. This model straddles between a trading strategy and weight optimization and falls within the supervised learning section because of its use of two neural models to generate market views.

To address the much-discussed limitations of the Markowitz model, Bayesian methods take additional information such as investor judgement and market fundamentals into account, a method once proposed by Black and Litterman (1990). The difference is that classical methods rely on financial experts and overlook public opinion and sentiment. A fair criticism of the Black and Litterman model is its subjective nature on market views; it leaves unanswered the question of how to assess these views. With the proposed method, public sentiment from the web can help researchers to formalize market views automatically.

The Black and Litterman model assumes that equilibrium returns are normally distributed $r_{eq} \sim N(\Pi, \tau\Sigma)$, where Σ is the covariance matrix of asset returns and τ is an indicator of the confidence level of the capital asset pricing model estimation of Π, the equilibrium risk premium of the market. The market views on the expected returns held by an investor agent are also normally distributed as $r_{views} \sim N(Q, \Omega)$. The posterior distribution of the portfolio returns that provide the views are therefore also Gaussian.

With the distributions denoted as $r_{BL} \sim N(\bar{\mu}, \bar{\Sigma})$, the vector of expected returns $\bar{\mu}$ and the covariance matrix $\bar{\Sigma}$ will be a function of the aforementioned variables:

$$[\bar{\mu}, \bar{\Sigma}] = f(\tau, \Sigma, \Omega, \Pi, Q)$$

The function can be induced by applying Bayes' theorem on the probability density function of the posterior expected returns:

$$pdf(\bar{\mu}) = \frac{pdf(\bar{\mu}|\Pi)\,pdf(\Pi)}{pdf(\Pi|\bar{\mu})}$$

The optimized Bayesian portfolio weights now have a form similar to the Markowitz model by substituting the MV version of Σ and μ with the new variables $\bar{\Sigma}$ and $\bar{\mu}$:

$$w^*_{BL} = (\delta\bar{\Sigma})^{-1}\bar{\mu}$$

A time series of asset prices, trading volume, and public sentiment data is used to approximate optimal market views. The sentiment is computed from a range of social media platforms using natural language processing techniques. The standard deviation will be interpreted as the confidence about the expected return of the portfolio, and a relative view would be described as taking the form "I have ω_1 confidence that asset x will outperform asset y by a%." An absolute view, in contrast, will take the form "I have a ω_2 confidence that asset z will outperform the market by b%." As a result, a portfolio of n assets and a set of k views can be represented by three matrixes $P_{k,n}$, $Q_{k,1}$, and $\Omega_{k,k}$.

[3] Resources: code (https://drive.google.com/open?id=1YDZQvz6Pn2AFDX2Uprfaq9JoGvk7RpJy); paper (Britten-Jones 1999).

Apart from performing slightly better than traditional methods, it also allows one to tell a nice Bayesian story.[4]

UNSUPERVISED LEARNING

Unlike supervised learning, which finds patterns using both input data and output data, unsupervised learning methods find patterns using only input data. An unsupervised learning framework is useful when researchers are not quite sure what to look for. Such a framework is often used for exploratory analysis and problem discovery purposes. Most unsupervised learning techniques take the form of dimensionality reduction or cluster analysis to establish groups of data that have some measure of similarity based on characteristic values.

Two of the most important techniques are K-means clustering and principal component analysis (PCA). The only requirement to be called an unsupervised learning strategy is to learn a new feature space that captures the characteristics of the original space by maximizing some objective function.[5] PCA attempts to reduce the number of features while preserving the variance, whereas clustering reduces the number of data points by summarizing them according to their mean expectations. However, those cluster assignments can also be used to label each data point with its assigned cluster, leading to a dimensionality reduction toward only one feature. K-mean and PCA in some sense maximize a similar objective function, with K-means having an additional categorical constraint.

PCA

PCA and clustering techniques are used to build classes of similar assets. The steps here involve integrated portfolio selection and risk estimation to optimize the portfolio. Let $Y = (Y_1, \ldots, Y_n)^T$ denote an n-dimensional random vector with variance–covariance matrix Σ. The goal of PCA is to construct linear combinations $P_i = \Sigma^n_{j=1} W_{ij} Yj$, for $i = 1, \ldots n$ in such a way that the P_i values are orthogonal so that $\Sigma[P_i P_j] = 0$ for $i \neq j$ so that the P_i are ordered to explain the largest percentage of the total variability in the system.

Each P_i explains the most significant percentage of total variability in the system that has not already been explained by $P_i, \ldots, P_i - 1$. Using various selection techniques, one can identify an optimum level of these components and apply it to a universe of stocks. This is not a complicated method, but it is very powerful and can be used as a mediating step in various trading strategies, if not in the final allocation decision making. In the code notebook, there is additional experimentation to compare this technique with hierarchical clustering methods.[6]

HIERARCHICAL RISK PARITY

The problem, as stated by López de Prado (2016), is that the MV portfolios are optimal in sample (training set) but perform poorly out of sample (test set). One way to deal with this problem is to drop forecasts altogether (e.g., risk parity [RP]). The problem is that both RP and MV require the inversion of a positive-definite covariance matrix. A new method is therefore suggested to overcome the matrix inversion and forecast issue, called hierarchical risk parity.

Hierarchical risk parity works by grouping similar investments into clusters based on a distance metric. The covariance matrix's rows and columns also are reorganized so that the largest values lie along the diagonal. Lastly, the allocations are split through recursive bisections of the reordered covariance matrix.

One starts by defining a distance measure between investments from zero and one, $d_{i,j}$, after which one clusters the pair of columns (i^*, j^*) together such that $(i^*, j^*) \text{argmin}_{(i,j)} = \{\grave{d}_{i,j}\}$ and $i \neq j$. The next step is to update $\{\grave{d}_{i,j}\}$ with the new cluster and apply steps 3 and 4 recursively until all N-1 clusters are formed.

We now place correlated investments close together and uncorrelated investments further apart and carry out a top-down allocation: We assign unit weights to all items by recursively bisecting a list of items by computing the variance and the split factor and rescaling the allocations; we iterate this process until full allocation is achieved. In the code, I have included an implementation using Robert Martin's PyPortfolioOpt[7] and an implementation of Chapter 16 in *Advances in*

[4] Resource: code (https://colab.research.google.com/drive/1sMAoJZuuNIRnrivAzxHV5fulMOWO17mb).

[5] Inherent to the PCA is the maximization of variance through a simple linear algebra operation by taking the eigenvectors of a covariance matrix of features.

[6] Resource: code (https://colab.research.google.com/drive/1mm9r6EZOERHYkycDbc74GY7S2U6h1oTc).

[7] https://github.com/robertmartin8/PyPortfolioOpt.

Financial Machine Learning (López de Prado 2018) by Hudson & Thames,[8] a buy-side open-source research unit headed by Jacques Joubert.[9]

Network Graph

Given N assets in a portfolio, we have to identify the weights $w_i, (\Sigma_{i=1}^{N} w_i = 1)$ so that highly correlated assets obtain lower relative weights. To do this, we can use a weighted graph that is an ordered tuple, $G = (V, E, W)$, where V is a set of vertices (or nodes), E is a set of pairwise relationships (the edges) between the vertices, and W is a set of numerical values assigned to each edge. A useful representation of G is the adjacency matrix:

$$A_{ij} = \begin{cases} 1, & \text{if } i \text{ is adjacent to } j \\ 0, & \text{otherwise} \end{cases}$$

Here the pairwise relations are expressed as the *ij* entries of an $N \times N$ matrix where N is the number of nodes. The strategy is then to transform the historical pricing data into a graph with edges weighted by the correlation between each stock. We can then use graph centrality measures and graph algorithms to obtain the desired allocation weights. In five steps, we do the following:

1. Compute the distance correlation matrix $\rho_D(X_i, X_j)$ for the open, high, low, close, and return time series.
2. Use the NetworkX[10] module to transform each distance correlation matrix into a weighted graph.
3. Adopt a winner-takes-all method and remove edges with correlations below a threshold value of $\rho_c = 0.325$ (adjust this threshold value if the graph disconnects):

$$Cor_{ij} = \begin{cases} \rho_D(X_i, X_j), & \rho \geq \rho_c \\ 0, & \text{otherwise} \end{cases}$$

4. Inspect the distribution of edges (the so-called *degree distribution*) for each network.[11] The degree of the *i*th vertex is given as

$$Deg(i) = \sum_{j=1}^{N} A_{ij}$$

5. Finally, build a master network by averaging over the edge weights of the open, high, low, close, and return networks and derive the asset weights from its structure.

Multiple variations are available for constructing the network and obtaining the distance; a few different approaches are considered in the notebook. An extensive analysis can be found in a notebook by Maya Benowitz, a quant at CarVal.[12]

REINFORCEMENT LEARNING

Reinforcement learning in finance comprises the use of an agent that learns how to take actions in an environment to maximize some notion of cumulative reward. We have an agent that exists in a predefined environment. The agent receives as input the current state S_t and is asked to take an action A_t to receive a reward R_{t+1}, the information of which can be used to identify the next optimal action, A_{t+1} given the new state S_{t+1}. The final objective function can be the realized/unrealized profit and loss and even risk-adjusted performance measures such as the Sharpe ratio. Allocation decisions in finance are challenging to deal with because finance is partially observed, nonstationary, regime dependent, and noisy. Standard models apply and recombine single-period predictions using an optimizer, but in the real world, the actions could have long-term consequences that could be acted against by the environment. Reinforcement learning can help us to deal with some of these problems by taking a more holistic approach.

Deep Determinist Policy Gradient

Deep learning models can be used in reinforcement learning to solve high-dimensional problems.

[8] https://hudsonthames.org/.

[9] Resources: data (https://drive.google.com/open?id=198fpHhD973i3rKa9D7oz-SrmBwPykQEc), code (https://drive.google.com/open?id=1z3Fe7QXZ6c566KOG3HtQEfCc84UAGwFf), code 2 (https://colab.research.google.com/drive/1-Z3OjjnIR-41E2tycKFosvxEt-RrAgZB).

[10] https://github.com/networkx/networkx.

[11] The degree of a node is simply the number of connections it has to other nodes.

[12] Code available at https://colab.research.google.com/drive/10WNiVuICvFajW2uTDrwI6w7aSUkjINPl.

To deal with the continuous action space of portfolio weighting, we can use Google DeepMind's off-policy and model free algorithm called deep deterministic policy gradient (Lillicrap et al. 2015). Using this method, we do not have to discretize continuous action spaces, which might lead to the curse of dimensionality and a loss of valuable information. The environment is derived from an OpenAI environment class.[13] This environment takes arguments like trading cost and window size into account to approach a realistic setting.

To start formalizing the problem, we set the number of stocks to N. At time step 1, we are fully invested, and the close/open relative price vectors are defined as

$$\gamma_t = \left[1, \frac{S_{1,t,close}}{S_{1,t,open}}, \frac{S_{2,t,close}}{S_{2,t,open}}, \ldots, \frac{S_{N,t,close}}{S_{N,t,open}}\right]$$

The portfolio weight vector is then defined as

$$w_t = [w_{o,t}, w_{1,t}, \ldots, w_{N,t}]$$

Here $w_{i,t}$ is the fraction of investment in stock i at time stamp t. Note that $w_{o,t}$ represents the fraction of cash maintained. The profit after timestamp T is

$$p_T = \prod_{t=1}^{T} \gamma_t \cdot w_{t-1}$$

Once we consider the trading cost factor of μ, the trading cost at each time stamp is

$$\mu_t = \mu \sum \left| \frac{\gamma_t \odot w_{t-1}}{\gamma_t \cdot w_{t-1}} - w_t \right|$$

where $\odot$ is the element-wise product. The profit can then be formulated as

$$p_T = \prod_{t=1}^{T} (1 - \mu_t) \gamma_t \cdot w_{t-1}$$

Now that we have dealt with the problem formulation, we have to formulate the Markov decision process. We set the state o_t as the fixed window W price history $\overrightarrow{S_{i,t}}$ of all the assets N:

$$o_t = [\overrightarrow{S_{1,t}}, \overrightarrow{S_{2,t}}, \ldots, \overrightarrow{S_{N,t}}]$$

$$\overrightarrow{S_{i,t}} = \begin{bmatrix} S_{i,t-W} \\ S_{i,t-W+1} \\ \vdots \\ S_{i,t-W} - 1 \end{bmatrix}$$

Moreover, the action a_t is merely the portfolio weight vector as previously defined, w_t. We therefore want to train a policy network: $\pi_\theta(a_t, o_t)$. The underlying evolution and transition of state are then determined by the market, and we can simply obtain the observation of the states. The reward is specified as the log-profit at each time step, $\log p_T$, which avoids the sparsity of rewards problem. Recall that with the deep deterministic policy gradient algorithm we want to learn the policy network $\pi_\theta(a_t, o_t)$ with a continuous action space reinforcement algorithm. The actor network is set to be the same as the policy network; the critic network is a linear combination of the actor network structure of the state and action; the exploration noise is set by the Ornstein–Uhlenbeck process with zero mean, 0.3 sigma, and 0.15 theta.[14]

The specific implementation in the code notebook looks at 15 stocks for the first 10 months in 2018 with data in minute format with open, close, high, low, and volume variables. This method is adapted from Jiang, Xu, and Liang's (2017) work published on arXiv. The action space contains a cash position, long positions, and short positions. The algorithm is hardcoded to only act every seven minutes. Note that reinforcement models can be very unstable and are generally hard to converge; when they do converge, they generally overfit. At this stage, without any further innovation, reinforcement learning models have to be implemented with great care.[15]

SUMMARY

There are several ways in which modern machine learning innovations can be used to help portfolio

[13] More can be found in their documentation: https://gym.openai.com/docs/.

[14] For more information on this distinction, see the original DeepMind paper referenced at the start.

[15] Resource: code (https://colab.research.google.com/drive/1L3-D2ZmGZkPRsB9gb5BviGkSkMTLti7_).

managers optimally allocate assets. Machine learning is poised to partially replace traditionally rigid allocation methods. This article divides these methods into supervised, unsupervised, and reinforcement learning frameworks. Within the supervised learning framework, three techniques were considered: first, a more traditional linear approach using OLS, Ridge, and LASSO regressions; second, a nonlinear deep learning approach using autoencoders; and lastly a Bayesian sentiment method. The article further identified three unsupervised methods using principal component analysis, hierarchical clustering analysis, and network graphs. The article finished with a deep reinforcement learning approach to portfolio weight optimization. As with the first article, I expect to see much more growth in reinforcement learning methods in the coming years. To paraphrase Vladimir Vapkin, the inventor of the support-vector machine method: One should avoid solving difficult intermediate problems when what one truly wants to solve is a target problem.

REFERENCES

Black, F., and R. Litterman. 1990. "Asset Allocation: Combining Investor Views with Market Equilibrium." *Goldman Sachs Fixed Income Research* 115.

Britten-Jones, M. 1999. "The Sampling Error in Estimates of Mean–Variance Efficient Portfolio Weights." *The Journal of Finance* 54 (2): 655–671.

Heaton, J. B., N. G. Polson, and Witte. 2017. "Deep Learning for Finance: Deep Portfolios." *Applied Stochastic Models in Business and Industry* 33 (1): 3–12.

Jiang, Z., D. Xu, D., and J. Liang. 2017. "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem." arXiv preprint 1706.10059.

Kingma, D. P., and J. Ba. 2014. "Adam: A Method for Stochastic Optimization." arXiv preprint 1412.6980.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2015. "Continuous Control with Deep Reinforcement Learning." arXiv preprint 1509.02971.

López de Prado, M. 2016. "Building Diversified Portfolios That Outperform Out of Sample." *The Journal of Portfolio Management* 42 (4): 59–69.

———. *Advances in Financial Machine Learning*, 1st ed.. Newark: John Wiley & Sons, 2018.

Snow, D. 2020. "Machine Learning in Asset Management—Part 1: Portfolio Construction—Trading Strategies." *The Journal of Financial Data Science* 2 (1): 10–23.

Xing, F. Z., E. Cambria, L. Malandri, and C. Vercellis. 2018. "Discovering Bayesian Market Views for Intelligent Asset Allocation." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 120–135. Cham, Switzerland: Springer, 2018.

ADDITIONAL READING

Machine Learning in Asset Management—*Part 1: Portfolio Construction—Trading Strategies*

Derek Snow
The Journal of Financial Data Science
https://jfds.pm-research.com/content/2/1/10

ABSTRACT: *This is the first in a series of articles dealing with machine learning in asset management. Asset management can be broken into the following tasks: (1) portfolio construction, (2) risk management, (3) capital management, (4) infrastructure and deployment, and (5) sales and marketing. This article focuses on portfolio construction using machine learning. Historically, algorithmic trading could be more narrowly defined as the automation of sell-side trade execution, but since the introduction of more advanced algorithms, the definition has grown to include idea generation, alpha factor design, asset allocation, position sizing, and the testing of strategies. Machine learning, from the vantage of a decision-making tool, can help in all these areas.*

Building Diversified Portfolios that Outperform Out of Sample

Marcos López de Prado
The Journal of Portfolio Management
https://jpm.pm-research.com/content/42/4/59

ABSTRACT: *In this article, the author introduces the Hierarchical Risk Parity (HRP) approach to address three major concerns of quadratic optimizers, in general, and Markowitz's critical line algorithm (CLA), in particular: instability, concentration, and underperformance. HRP applies modern mathematics (graph theory and*

machine-learning techniques) to build a diversified portfolio based on the information contained in the covariance matrix. However, unlike quadratic optimizers, HRP does not require the invertibility of the covariance matrix. In fact, HRP can compute a portfolio on an ill-degenerated or even a singular covariance matrix—an impossible feat for quadratic optimizers. Monte Carlo experiments show that HRP delivers lower out-of-sample variance than CLA, even though minimum variance is CLA's optimization objective. HRP also produces less risky portfolios out of sample compared to traditional risk parity methods.