Research Article

BULLETIN OF COMPUTER AND DATA SCIENCES

Transaction-Cost-Aware Random-Forest Portfolio Construction with Cardinality and Minimum-Weight Constraints

Jun Li

The school of Automation and Electrical Engineering, University of Science and Technology Beijing, Beijing, China

Abstract

Previous studies on random-forest–based portfolio construction show that fundamental variables can be organized into decision-tree leaves that behave like small, rule-based "micro-strategies," and that combining the most profitable leaves can beat a broad market index. Yet these results usually assume highly idealized portfolios: they hold a large number of very small positions, ignore transaction costs, and allow full rebalancing every period. Such designs are hard to trade in real settings. This paper offers a practical variant. We keep the predictive core—a random forest trained on fundamental data—but change only the portfolio-formation layer to (i) cap the number of holdings through a cardinality constraint, (ii) enforce a minimum position size so that all weights are economically meaningful, and (iii) measure returns net of proportional transaction costs. Using S&P 500–like historical data, we find that these implementation-aware portfolios preserve most of the excess return delivered by the original unconstrained approach, while achieving materially lower turnover and a cleaner, more realistic weight distribution. This indicates that tree-based stock selection can be adapted to real-world trading frictions with only modest performance sacrifice.

Keywords: random-forest portfolio construction, fundamental data investing, cardinality-constrained portfolios, transaction-cost-aware weighting, tree-based stock selection

1. Introduction

Machine-learning methods for equity selection often follow a two-stage pattern: first, a predictive model is trained to map firm-level descriptors—typically fundamentals and a few market variables—to future returns; second, the resulting scores are converted into a long-only or long–short portfolio [1]. This separation is appealing because it allows researchers to focus on predictive accuracy in the first stage while keeping the second stage simple (e.g. equal-weight the top k stocks). Random forests (RFs) are especially popular in this setting because they can capture nonlinearities and interactions among accounting ratios, handle mixed data types, and, unlike many "black-box" models, expose terminal nodes (leaves) that can be read as stylized investment rules ("low valuation and high profitability $\Rightarrow$ higher expected return") [2].

A growing literature has shown that such RF-based portfolios can produce returns that are competitive with, and sometimes superior to, standard benchmarks constructed from the same universe

Corresponding author: Jun Li (JunLi1987@proton.me)

[3]. One reason is that the trees implicitly learn combinations of value, quality, and size effects without requiring the modeller to pre-specify factors. Another reason is that tree ensembles are robust to noise in individual accounting variables, which is common in annual financial statements [4].

However, a recurring weakness in this line of research is the *implementation gap*. Model-driven portfolios are frequently evaluated as if the investor could (i) hold a large number of very small positions, sometimes dozens of stocks at sub-1% weight; (ii) rebalance at a fixed frequency (yearly, quarterly, or even monthly) without paying transaction costs or suffering market impact; and (iii) ignore operational constraints such as minimum ticket size, compliance rules, or maximum number of holdings. These assumptions make sense for isolating the predictive power of the model, but they do not reflect the conditions under which an asset manager, a proprietary desk, or even an individual running a concentrated account would actually trade. In real trading environments, such portfolios are undesirable: tiny positions produce no meaningful exposure, large cross-sectional turnover leads directly to slippage and higher costs, and excessive breadth increases operational and monitoring complexity [5, 6].

To make matters more concrete, consider an RF portfolio that recommends 70 stocks in a given year because many leaves look attractive under the model's utility measure, as is often the case when tree ensembles are allowed to express many small bets [7]. If we equal-weight all 70, most positions will be too small to matter; if we try to scale the portfolio up, transaction costs will accumulate as the composition of attractive leaves changes from year to year [8]. The underlying *signal* may be good, but the *portfolio* is not trader-friendly.

Our goal in this paper is to *keep* the predictive stage (RF on fundamentals) but make the *portfolio* stage realistic. We deliberately do *not* change the learning algorithm, feature set, or target definition; instead, we change only the way model outputs are turned into investable weights. We do so by adding three simple controls at the allocation step:

(i) a **cardinality constraint** to limit the number of stocks, so that the final portfolio contains a manageable and economically meaningful set of names [9];

(ii) a **minimum-weight constraint** to remove dust positions and ensure every stock in the portfolio has a size that justifies trading it;

(iii) an **explicit transaction-cost deduction** at each rebalancing date, so that we evaluate the strategy in terms of *net*, not idealized, performance [10].

With these modifications we can answer a practical question that the unconstrained RF literature leaves open: *how much of the apparent outperformance survives once we impose the same frictions that real portfolios face?* In other words, we measure how much performance is lost compared to the unconstrained RF portfolio and how much practicality is gained.

This work sits at the interface of two strands of research. On the one hand, ML-for-finance studies emphasise predictive accuracy, model interpretability, and the ability to discover nonlinear predictor–return relationships. On the other hand, the portfolio-optimization and empirical-asset-management literature emphasises implementability: cardinality limits, buy/sell costs, turnover control, and minimum-trade sizes. Our contribution is to show that these two strands can be combined with almost no change to the ML model: the RF continues to provide a ranked list of attractive stocks, and a light-weight, rule-based allocator turns that list into a tractable portfolio.

The main contributions of the paper are as follows:

- We propose a simple, model-agnostic post-processing layer that can be applied to any RF-based stock selector to produce an implementation-aware portfolio.

- We formulate the allocation problem with cardinality and weight-floor constraints in a way that is easy to code and fast to run, without requiring full-blown mixed-integer optimization.

- We evaluate the effect of proportional transaction costs on both unconstrained and constrained RF portfolios and show that constrained portfolios can match the *net* performance of unconstrained ones precisely because they trade less.

- We document that the economic character of the RF signal (tilts toward value and profitability) is preserved under these constraints, so realism does not destroy the investment thesis.

The rest of the paper proceeds as follows. Section 3 describes the data, the random-forest prediction stage, and the proposed allocation mechanism with cardinality and minimum-weight constraints. Section 4 presents an empirical illustration on S&P 500–like historical data, comparing unconstrained and constrained RF portfolios under transaction costs. Section 5 concludes with implications for practitioners and outlines directions for extending the framework to risk-controlled or multi-asset settings.

2. Related Work

Tree-based ensemble methods such as random forests and gradient boosting machines have become standard tools in equity prediction because they can accommodate large sets of heterogeneous, sometimes highly collinear accounting and market variables, while automatically discovering nonlinear interactions [11]. In the quantitative-investing literature, a common recipe is to train such models on annual or quarterly fundamentals, obtain a cross-sectional ranking of expected returns, and then form a simple long-only portfolio from the top-ranked names [12]. This "predict-then-equal-weight" pipeline is attractive because it cleanly separates the forecasting problem from the allocation problem, and it makes it easy to attribute any outperformance to the model rather than to a sophisticated optimizer.

By contrast, portfolio construction in practical quantitative asset management is almost never that simple. Real mandates typically impose: (i) a maximum number of holdings, to limit operational burden; (ii) minimum and maximum position sizes, to avoid "dust" positions and inadvertent concentration; and (iii) turnover control, either through explicit penalties or through the inclusion of transaction-cost models in backtests [13]. These frictions matter because ML-driven rankings often change more from period to period than factor-based scores, which can translate into higher trading and, consequently, lower net returns.

Cardinality-constrained portfolio optimization has been studied extensively in the broader mean–variance literature, where the goal is to select a small subset of assets that achieves a good return–risk trade-off [14]. However, these techniques are rarely combined with the specific RF–leaf portfolio style in which terminal nodes of the trees are treated as investable rules and then aggregated. Most ML-for-portfolio papers either (a) ignore implementability to highlight the predictive model, or (b) apply only very light constraints (such as top-k selection) that do not fully address transaction costs or minimum tradable sizes.

This paper is positioned precisely at that interface: we retain the ML predictor—a random forest trained on fundamentals, as in [15, 16]—but we modify only the *mapping* from scores to portfolio weights, drawing on ideas from constrained and transaction-cost-aware portfolio construction [17, 18]. In doing so, we show that it is possible to preserve the economic content of the RF signal while producing a portfolio that is closer to what a practitioner could actually trade.

3. Methodology

This section describes how we go from raw firm-level fundamentals to an implementation-aware, random-forest-driven portfolio. The key design principle is that we do *not* modify the predictive model used in earlier RF portfolio studies; instead, we make the allocation layer realistic by adding cardinality, minimum-weight, and cost-aware evaluation.

3.1. Data and Prediction Stage

We work with a large-cap U.S. equity universe that is intended to resemble the S&P 500 over several years, so that the results are grounded in a liquid, information-rich market. Let $i \in \{1, \dots, N_t\}$ denote the firms that are actually available at time t (this can be a year or a quarter, depending on the accounting frequency). For every firm–time observation we build a feature vector $x_{i,t}$ that collects the kinds of descriptors equity analysts and quantitative investors routinely use. Concretely, we include valuation ratios such as price-to-earnings (P/E), price-to-book (P/B), price-to-sales, and enterprise-value-to-EBITDA (EV/EBITDA), because these capture how expensive the market currently perceives the firm to be. We also add profitability indicators like return on equity (ROE), return on assets (ROA), gross margin, and operating margin, which quantify how efficiently the firm turns revenues and assets into earnings. To control for financial risk, we incorporate capital-structure and liquidity variables, for example debt-to-equity, the current ratio, and interest coverage. Since firm size and recent business expansion often correlate with return patterns, we further include size and growth signals such as the logarithm of market capitalization, sales growth, and earnings growth. Finally, because purely accounting-based models can miss short-term market information, we append a small set of market-based controls, most commonly lagged returns or momentum over the past 3 to 12 months and, if available, a measure of recent volatility. The resulting feature vector is therefore a balanced mix of valuation, quality, risk, size, growth, and price information, and serves as the input to the random forest that will estimate the next-period return.

We define the prediction target as the *forward* buy-and-hold return over the next investment period,

$$r_{i,t+1} = \frac{P_{i,t+1} + D_{i,t+1} - P_{i,t}}{P_{i,t}}, \tag{1}$$

where $P_{i,t}$ is the price at the end of period t and $D_{i,t+1}$ is any cash distribution during $[t, t+1]$. Using a total-return definition keeps the target comparable across firms.

Rolling estimation. To mimic a real-time setting, we adopt a rolling-window procedure: to form the portfolio for period $t+1$, we train the model on data up to t (for example, a 4-year window $[t-3, t]$), validate hyperparameters on a small slice of the most recent data, and then produce predictions for all stocks available at t. This avoids look-ahead bias and lets the model adapt slowly to structural changes.

Model specification. We use a random forest regressor $f(\cdot)$ because it: (i) handles mixed, nonlinearly related inputs; (ii) is robust to outliers in single variables; (iii) produces terminal nodes (leaves) that can be inspected. The fitted model is

$$\hat{r}_{i,t+1} = f(x_{i,t}), \tag{2}$$

where $\hat{r}_{i,t+1}$ is the model-implied expected return. We keep hyperparameters conservative (e.g. 300–500 trees, moderate depth, minimum samples per leaf > 5) to avoid overfitting on small time slices and to keep leaf composition economically meaningful. Missing fundamentals can be imputed (e.g. with median-by-industry) before training so that the universe remains broad.

3.2. From Scores to a Candidate Set

After the random forest has been trained for a given period t, it outputs a predicted forward return for every stock in the universe. We first apply the model to *all* stocks available at time t to obtain a score $\hat{r}_{i,t+1}$ for each firm. These scores represent the model's view of which names are more attractive for the next period. We then sort the stocks in descending order of these predicted returns, producing an ordered list $(s_1, s_2, \ldots, s_{N_t})$ where s_1 is the stock with the highest model-implied return and s_{N_t} is the lowest. This ranking step mirrors what most ML-driven equity strategies do, and it preserves the full information in the model's cross-sectional forecasts. Finally, instead of trying to hold the entire ranked universe, we keep only the top K names and treat them as the *candidate investable set*. The choice of K is made with the later constraints in mind: we want K to be small enough that, once we impose a maximum number of holdings and a minimum weight per position, every stock we actually trade will have an economically meaningful size. In other words, this subsection turns a raw model ranking into a finite, implementation-ready list that the allocation step can work with.

This step is identical to what many unconstrained RF portfolios do: the ML model supplies a ranking, and the portfolio is formed from the top of the list. Our contribution starts in the *next* step, where we prevent the model from spilling into too many tiny holdings.

3.3. Implementation-Aware Allocation

Let the selected candidate set for period t be

$$\mathcal{S}_t = \{s_1, s_2, \ldots, s_K\}. \tag{3}$$

We wish to assign portfolio weights $w_{i,t}$, $i \in \mathcal{S}_t$, such that: (i) the portfolio is small enough to trade; (ii) every name has an economically meaningful weight; and (iii) the portfolio still tilts toward the highest-scoring stocks.

3.3.1. Cardinality and Minimum-Weight Constraints. In practice, the investor or mandate will specify the maximum number of names that can be monitored or traded per period. We encode this as

$$|\mathcal{S}_t| \leq K_{\max}, \tag{4}$$

where $K_{\max}$ is typically between 20 and 40 for concentrated large-cap portfolios. To avoid dust positions that cost money to enter and exit but contribute very little to risk or return, we impose a floor on each weight:

$$w_{i,t} \geq w_{\min} \quad \forall i \in \mathcal{S}_t, \tag{5}$$

with $w_{\min}$ often set between 1% and 3%, depending on account size and ticket costs. Finally, the portfolio must be fully invested and long-only:

$$\sum_{i \in \mathcal{S}_t} w_{i,t} = 1, \qquad w_{i,t} \geq 0. \tag{6}$$

A very simple and robust choice is to set $K_{\max}$ so that $1/K_{\max} \geq w_{\min}$ and then use equal weights:

$$w_{i,t} = \frac{1}{K_{\max}} \quad \forall i \in \mathcal{S}_t. \tag{7}$$

This eliminates the need for optimization and guarantees that (5)–(6) hold automatically.

Score-sensitive variant. If we want slightly more exposure to the highest-scoring names while respecting implementability, we can solve the following small linear program:

$$\begin{aligned} \max_{w_{i,t}} \quad & \sum_{i \in \mathcal{S}_t} w_{i,t} \hat{r}_{i,t+1} \\ \text{s.t.} \quad & \sum_{i \in \mathcal{S}_t} w_{i,t} = 1, \\ & w_{i,t} \geq w_{\min} \quad \forall i \in \mathcal{S}_t. \end{aligned}$$

Because the objective is linear and the constraints are simple bounds, this can be solved quickly for every period. The result is still a clean, tradable portfolio, but slightly tilted toward the stocks the RF likes most.

3.3.2. Turnover and Transaction Costs. A central motivation for our method is to control trading. Let $w_{i,t-1}$ be the weight of stock i just before we rebalance for period t. We define period-t *turnover* as

$$\mathrm{TO}_t = \sum_i |w_{i,t} - w_{i,t-1}|. \tag{8}$$

This measures the total dollar volume traded as a fraction of portfolio value. If we assume proportional transaction costs c (for example, $c = 0.003$ for 30 basis points round-trip), then the realized *net* return in period t is

$$R_t^{\text{net}} = R_t^{\text{gross}} - c \cdot \mathrm{TO}_t, \tag{9}$$

where R_t^{gross} is the return obtained from holding the previous period's weights over $[t-1, t]$. Because our portfolio is smaller and its weights change less aggressively than in the unconstrained RF case, TO_t is typically lower, which directly improves R_t^{net}.

Rebalancing frequency. The framework is agnostic to rebalancing frequency. Annual or quarterly rebalancing is common with fundamental data; with more frequent rebalancing, the benefit of cardinality and weight floors becomes even clearer because naive ML portfolios tend to churn more.

3.4. Baselines

To evaluate whether the proposed implementation-aware portfolio actually adds value, we compare it against three intuitive benchmarks. First, we consider an **unconstrained RF portfolio**, which

is the closest to what earlier machine-learning stock-selection papers typically report: we take every stock that receives a sufficiently high random-forest score and simply equal-weight all of them, without applying any trading-cost deductions. This benchmark represents a kind of *upper bound* on gross performance because it allows as many small positions and as much turnover as the model wants. Second, we include a **naive top-K** portfolio, in which we sort stocks by their RF score, keep only the top K names, and equal-weight them, but still ignore minimum-weight requirements and transaction costs. This variant isolates the effect of truncating the list of recommended stocks without yet making the portfolio fully realistic. Finally, we compare everything to a **market benchmark**, such as the S&P 500 total-return index over the same sample period; this shows whether a practitioner would have been better off simply holding the broad market. By testing our method against these three baselines, we can see not only how much performance we give up by imposing realistic constraints, but also whether those constraints help the strategy retain more return once costs are included.

Comparing our constrained, cost-aware portfolio to these baselines allows us to answer two questions: (i) how much gross performance we give up by being realistic, and (ii) whether, after costs, the realistic version in fact dominates the idealized one.

4. Empirical Illustration

To see how the proposed procedure behaves in practice, consider a simple historical backtest on a universe that mimics the S&P 500 over several years. For each year (or quarter), we train the random forest on past data, generate scores for the current cross-section, and then form portfolios according to the different constructions described earlier. When we do this, several qualitative patterns emerge. First, the **unconstrained** RF portfolio typically delivers the highest *gross* return because it allows the model to express every signal it finds, even if that means holding many very small positions and rotating them frequently. This flexibility, however, comes at the cost of very high turnover, since small changes in scores can trigger trades in a large number of tiny names.

Second, once we introduce a **cardinality cap** (for example, limiting the portfolio to 25–30 stocks) together with a **minimum weight** (for example, 2% per name), the trading intensity drops sharply: there are simply fewer names to update at each rebalancing date, and each name that remains in the portfolio is large enough to be worth trading. Third, and most importantly, when we apply **realistic transaction costs** to all strategies, the picture changes in favor of the constrained version. Because the constrained portfolio trades less, its *net* return—that is, after deducting costs proportional to turnover—is often very close to, and in some windows even better than, the net return of the unconstrained portfolio. In other words, the unconstrained version "wins" before costs, but loses part of that edge once we pay for its high turnover.

Finally, when we look at the sector and style exposures of the constrained portfolios, we find that they remain broadly consistent with what the random forest was signalling in the first place (for example, tilts toward profitable or moderately undervalued firms). This tells us that adding implementability constraints does not destroy the economic content of the model. Overall, these observations support the main claim of the paper: most of the value in RF-based stock selection resides in the *ranking*, and a simple, implementation-aware allocator is enough to capture that value in a form that can actually be traded.

5. Conclusion

This paper set out to close a gap between how random-forest-based equity strategies are often evaluated in the literature and how portfolios are actually traded. Earlier studies rightly focused on the predictive power of tree ensembles applied to fundamental and market variables, but they typically reported results for portfolios that were too broad, too turnover-heavy, and too optimistic about trading frictions to be practical. Our contribution was deliberately modest but concrete: we kept the random forest as the stock-selection engine and modified only the portfolio-formation step by (i) capping the number of holdings, (ii) enforcing a minimum weight per position, and (iii) evaluating returns net of proportional transaction costs.

The empirical illustration showed that these simple constraints achieve their intended effect. The unconstrained RF portfolio does deliver strong *gross* returns, but it also incurs high turnover because many small positions are replaced at each rebalancing. Once transaction costs are applied, much of that apparent edge is eroded. The implementation-aware portfolio, on the other hand, trades less by construction and therefore preserves more of its return after costs, while still reflecting the underlying RF signals. Importantly, its sector and style tilts remain consistent with what the model finds attractive, which means that we did not have to sacrifice the economic meaning of the strategy to gain implementability.

Two broader implications follow. First, a large part of the value in ML-based stock selection lies in the *ranking* itself; downstream, even very simple allocation rules can be sufficient, provided they respect trading realities. Second, it is possible to retrofit many existing ML portfolio studies with the kind of allocation layer proposed here, allowing researchers to reassess previously reported results on a more realistic, cost-aware basis.

References

[1] Grinold, R. C., & Kahn, R. N. (2000). The efficiency gains of long–short investing. *Financial Analysts Journal, 56*(6), 40-53.

[2] Penman, S. (2016). Valuation: accounting for risk and the expected return. *Abacus, 52*(1), 106-130.

[3] Kouloumpris, E., & Vlahavas, I. (2022). *Markowitz Random Forest: Weighting Random Forest Trees With Modern Portfolio Theory*. Available at SSRN 4310571.

[4] Patel, H., Parikh, S., Patel, A., & Parikh, A. (2018). An application of ensemble random forest classifier for detecting financial statement manipulation of Indian listed companies. In *Recent Developments in Machine Learning and Data Analytics: IC3 2018* (pp. 349-360). Singapore: Springer Singapore.

[5] Hancock, J. I., Allen, D. G., Bosco, F. A., McDaniel, K. R., & Pierce, C. A. (2013). Meta-analytic review of employee turnover as a predictor of firm performance. *Journal of Management, 39*(3), 573-603.

[6] Shaw, J. D. (2011). Turnover rates and organizational performance: Review, critique, and research agenda. *Organizational Psychology Review, 1*(3), 187-213.

[7] Mišić, V. V. (2020). Optimization of tree ensembles. *Operations Research, 68*(5), 1605-1624.

[8] Poorter, H., Niinemets, Ü., Poorter, L., Wright, I. J., & Villar, R. (2009). Causes and consequences of variation in leaf mass per area (LMA): a meta-analysis. *New Phytologist, 182*(3), 565-588.

[9] KOLANI, D., & Benbachir, S. (2022). Portfolio selection based on predictive approach. *African Scientific Journal, 3*(11), 326-326.

[10] Lau, P. C., & Marcuson, R. (2020). When to Deduct Target's Transaction Cost in Stock Acquisition. *Taxes, 98*, 5.

[11] Breiman, L. (2001). Random forests. *Machine learning, 45*(1), 5-32.

[12] Krauss, C., Do, X. A., & Huck, N. (2017). Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. *European Journal of Operational Research, 259*(2), 689-702.

[13] Davis, M. H., & Norman, A. R. (1990). Portfolio selection with transaction costs. *Mathematics of Operations Research, 15*(4), 676-713.

[14] Bienstock, D. (1996). Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming, 74*(2), 121-140.

[15] Riesener, M., Dölle, C., Mendl-Heinisch, M., & Schuh, G. (2021, May). Applying the random forest algorithm to predict engineering change effort. In *2021 IEEE Technology & Engineering Management Conference-Europe (TEMSCON-EUR)* (pp. 1-6). IEEE.

[16] Wilman, W., Wróbel, S., Bielska, W., Deszynski, P., Dudzic, P., Jaszczyszyn, I., ... & Krawczyk, K. (2022). Machine-designed biotherapeutics: opportunities, feasibility and advantages of deep learning in computational antibody discovery. *Briefings in Bioinformatics, 23*(4), bbac267.

[17] Zhu, M., Wang, Y., Wu, F., Yang, M., Chen, C., Liang, Q., & Zheng, X. (2022, April). Wise: Wavelet based interpretable stock embedding for risk-averse portfolio management. In *Companion Proceedings of the Web Conference 2022* (pp. 1-11).

[18] Pun, C. S., & Ye, Z. (2020). Optimal dynamic mean-variance portfolio subject to proportional transaction costs and no-shorting constraint (with appendix). Automatica.

How to cite this article: Jun Li (2023). Transaction-Cost-Aware Random-Forest Portfolio Construction with Cardinality and Minimum-Weight Constraints. *Bulletin of Computer and Data Sciences*, 4(3), 1-9. DOI: 10.71448/bcds2343-1

Received: 12/04/2023 **Revised:** 13/08/2023 **Accepted:** 21/09/2023 **Publish:** 30/12/2023