Big Data in Finance

Lecture 3: Tree-Based Regression Methods

Dr Daniele Bianchi

Spring 2026

Contents

Overview

In Lecture 2, we studied penalized regression methods — Ridge, Lasso, and Elastic Net — that address the high-dimensional prediction problem by shrinking coefficients. These methods are powerful, but they share a fundamental assumption: the relationship between predictors and outcome is *linear*. The effect of each predictor is constant across its range, and the effects of different predictors simply add together.

But what if reality is more complicated? What if the dividend-price ratio predicts returns differently at high versus low levels? What if the effect of the term spread depends on whether we are in a recession or an expansion? What if there are interaction effects that we have not thought to specify?

This lecture introduces **tree-based methods**, which abandon the linearity assumption entirely. Instead of fitting a global linear function, trees partition the feature space into regions and fit simple (constant) predictions within each region. The result is a fundamentally different kind of model — one that can capture nonlinearities and interactions automatically, without requiring us to specify them in advance.

But this flexibility comes at a cost: individual trees are notoriously unstable. Small changes in the training data can produce completely different trees. The solution is **ensemble learning**—combining many trees to achieve more stable and accurate predictions. We study two major ensemble strategies: Random Forests (which reduce variance by averaging many independent trees) and Gradient Boosting (which reduces bias by sequentially correcting errors).

Sections 1 to 4 develop the theory of trees and ensembles. Sections 5 to 8 apply these methods to our running market timing example and confront a surprising puzzle: the method with the best statistical accuracy does not have the best economic performance.

1 From Linear to Nonlinear Models

All the methods we studied in Lecture 2 assume a linear prediction function:

$$\widehat{f}(x) = \widehat{\beta}_0 + \sum_{j=1}^{p} \widehat{\beta}_j x_j$$

This functional form imposes three strong restrictions:

Linearity: The effect of predictor x_j on the outcome is constant. If increasing x_j by one unit raises the prediction by $\widehat{\beta}_j$, this is true regardless of whether x_j is small or large. There are no threshold effects, saturation effects, or other nonlinearities.

Additivity: The effect of x_j does not depend on the values of other predictors. The model cannot capture interactions—situations where the effect of one variable depends on another.

Global specification: The same coefficients $\widehat{\beta}$ apply everywhere in feature space. There is no concept of "different rules for different situations."

Of course, we can partially address these limitations within the linear framework by adding polynomial terms (x_j^2) and interaction terms ($x_j \cdot x_k$). But this requires us to specify which nonlinearities and interactions to include—a daunting task when we have many predictors and no strong prior theory. With 25 predictors, there are 300 pairwise interactions alone, and the model dimension explodes if we add higher-order terms.

1.1 Why Nonlinearity Might Matter for Market Timing

Consider the equity premium prediction problem from Lecture 2. There are good reasons to suspect that linear models miss important structure:

Nonlinear effects: The dividend-price ratio might predict returns differently at extreme values. Very high D/P might signal distress rather than value, reversing the typical relationship. Valuation ratios might have threshold effects that linear models cannot capture.

Interactions: The predictive power of one variable might depend on the values of others. Perhaps the dividend-price ratio predicts well when the term spread is high (a steep yield curve, a healthy economy), but not when the term spread is low or inverted.

Regime dependence: Predictive relationships might differ across market regimes. The factors that predict returns in recessions might not be the same as those that predict returns in expansions.

Tree-based methods promise to automatically discover such patterns without requiring us to specify them in advance. But the question remains: do these patterns actually exist in the data, and can we reliably detect them with the sample sizes available in finance?

2 Regression Trees

A regression tree approaches prediction in a fundamentally different way than linear models. Instead of fitting a global function, it partitions the feature space into regions and makes a constant prediction within each region.

The prediction function takes the form:

$$\widehat{f}(x) = \sum_{m=1}^{M} \widehat{c}_m \cdot \mathbf{1}(x \in R_m)$$

where $R_1, \ldots, R_M$ are disjoint regions that cover the feature space, $\widehat{c}_m$ is the prediction for region R_m, and $\mathbf{1}(x \in R_m)$ is an indicator that equals 1 if x falls in region R_m.

What should the prediction be within each region? Given any partition, the optimal constant prediction (in the squared error sense) is simply the average of the training outcomes in that region:

$$\widehat{c}_m = \frac{1}{n_m} \sum_{x_i \in R_m} y_i = \overline{y}_{R_m}$$

So if we knew the optimal partition, the predictions would just be regional averages. The challenge is finding a good partition.

2.1 Growing a Tree: Recursive Binary Splitting

Finding the globally optimal partition is computationally infeasible — there are too many possible ways to partition the feature space. Instead, trees are grown using a **greedy** algorithm called recursive binary splitting.

The algorithm starts with all data in a single region. At each step, it finds the single best split: one variable j and one threshold s such that dividing the current region at $x_j = s$ produces the largest reduction in residual sum of squares. Mathematically, we solve:

$$\min_{j,s} \left[\sum_{x_i \in R_1(j,s)} (y_i - \widehat{c}_1)^2 + \sum_{x_i \in R_2(j,s)} (y_i - \widehat{c}_2)^2 \right]$$

where $R_1(j, s) = \{x : x_j \leq s\}$ and $R_2(j, s) = \{x : x_j > s\}$ are the two regions created by the split.

The algorithm searches over all predictors j and all unique values of x_j in the data, evaluating the RSS reduction for each candidate split. This is computationally efficient: $O(np)$ operations per split.

After the first split creates two regions, the algorithm recursively applies the same procedure to each region, continuing until some stopping criterion is met (minimum node size, maximum depth, or minimum improvement).

2.2 What Trees Discover: A Simple Example

To build intuition, consider predicting equity premium returns using just two predictors: the dividend-price ratio (D/P) and the term spread (TMS).

Suppose the greedy algorithm first splits on D/P at a threshold of 2.0, creating a "low D/P" region and a "high D/P" region. The high D/P region might have average returns of +3%, while the low D/P region has average returns of -1%.

Next, the algorithm might split the low D/P region on TMS at a threshold of 1.5. This creates three final regions: (1) low D/P and low TMS, with average returns of -2%; (2) low D/P and high TMS, with average returns of +1%; (3) high D/P, with average returns of +3%.

What has the tree learned? It has discovered that high D/P predicts positive returns regardless of TMS. But for low D/P stocks, the term spread matters: low TMS predicts negative returns, while high TMS predicts mildly positive returns. The tree has automatically discovered an *interaction*—the effect of TMS depends on the level of D/P.

A linear model would need us to specify an interaction term D/P $\times$ TMS manually. The tree found it automatically.

2.3 Controlling Tree Complexity

A fully grown tree that splits until each leaf contains a single observation would perfectly fit the training data—and catastrophically overfit. We need to control complexity.

Stopping rules prevent the tree from growing too large. Common criteria include minimum samples per leaf (stop if a split would create a node with fewer than k observations), maximum depth (limit the number of sequential splits), and minimum impurity decrease (stop if the RSS reduction is below a threshold).

Pruning is a more sophisticated approach: grow a large tree first, then "prune" it back by removing splits that do not improve out-of-sample performance. The cost-complexity criterion balances fit against tree size:

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \widehat{c}_m)^2 + \alpha |T|$$

where $|T|$ is the number of terminal nodes and $\alpha \geq 0$ is a penalty parameter. This is analogous to the Lasso penalty on coefficient magnitude: both trade off fit against complexity, with the tuning parameter selected via cross-validation.

2.4 The Instability Problem

Trees face a fundamental problem: they are notoriously unstable. A small change in the training data can lead to a completely different tree structure.

Consider what happens if the best split at the root node changes due to sampling variation. A different root split cascades through the entire tree—different children, different grandchildren, and ultimately different predictions for many observations. The tree structure can change dramatically, even though the underlying relationship remains the same.

This instability is the cost of flexibility. Trees have low bias (they can fit complex patterns) but high variance (they are sensitive to the training sample). For prediction, this high variance is problematic: we want predictions that are stable and reliable, not ones that vary widely across samples.

The solution is not to use a single tree, but to combine many trees in an **ensemble**.

3 Ensemble Learning: Bagging and Random Forests

The key insight behind ensemble methods is simple: averaging reduces variance. If we have B independent estimators, each with variance σ^2, the variance of their average is σ^2/B—a factor of B reduction.

Think of economic forecasting. A single agency's forecast might be 60% accurate. But if you average forecasts from ten independent agencies, accuracy might improve to 75%. Individual forecasting errors tend to cancel out when averaged.

The same principle applies to trees. If we could train trees on B different independent training datasets and average their predictions, the variance would be dramatically reduced. But we have only one training dataset. How can we create the diversity needed for averaging to work?

3.1 Bootstrap Aggregation (Bagging)

The answer is **bootstrap sampling**. We create B "pseudo-datasets" by sampling n observations with replacement from the original training data. Each bootstrap sample has the same size as the original data, but some observations appear multiple times while others are omitted entirely (on average, about 63% of unique observations appear in each bootstrap sample).

Bagging (Bootstrap Aggregating) works as follows:

1. Generate B bootstrap samples from the training data.
2. Fit a deep, unpruned tree to each bootstrap sample.
3. Average the predictions: $\widehat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}^{(b)}(x)$

Each individual tree overfits its bootstrap sample—we deliberately let it grow deep without pruning. But the overfitting is to different samples, so when we average, the overfit components tend to cancel out. The result is a prediction with the flexibility of deep trees but much lower variance.

3.2 The Correlation Problem

There is a catch. Bootstrap samples are not truly independent — they all come from the same underlying data. The trees trained on them will be correlated.

The variance of an average of B estimators with pairwise correlation ρ is:

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} \widehat{f}^{(b)}(x)\right) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

As B increases, the second term vanishes, but the first term $\rho\sigma^2$ remains. If trees are highly correlated (ρ close to 1), averaging provides limited benefit.

The problem is acute when one predictor dominates. If D/P is the strongest predictor, it will be selected for the root split in most or all trees, making the trees very similar despite the bootstrap variation.

3.3 Random Forests: Decorrelating the Trees

Random Forests (Breiman, 2001) address the correlation problem by adding a second source of randomness: at each split, only a random subset of m predictors (out of p) is considered.

This small modification has a profound effect. Even if D/P is the strongest overall predictor, it will be excluded from some splits simply by random chance. Other predictors—TMS, inflation, earnings yield—get opportunities to split, creating more diverse trees.

The Random Forest algorithm:

1. Generate B bootstrap samples.
2. For each sample, grow a tree, but at each split:
    - Randomly select m predictors from the p available.
    - Find the best split among only these m predictors.
3. Average the predictions.

The typical choice is $m \approx \sqrt{p}$ for classification or $m \approx p/3$ for regression. Smaller m creates more diverse (less correlated) trees but may produce weaker individual trees. The net effect is usually positive: the variance reduction from lower correlation outweighs any increase in individual tree variance.

3.4 Out-of-Bag Error

Bootstrap sampling provides a bonus: free cross-validation. Each bootstrap sample omits about 37% of the original observations. These "out-of-bag" (OOB) observations were not used to train that particular tree, so they can be used to evaluate it.

For each observation i, we compute its OOB prediction by averaging only the trees for which observation i was out-of-bag. The OOB error is then computed by comparing these predictions to actual outcomes. This provides an honest estimate of test error without needing a separate validation set. This requires extra care in a time-series setting, as the sequence of training and validation matters.

3.5 Random Forest Properties

Random Forests have several attractive properties:

No overfitting from more trees: Unlike single trees, more trees in a Random Forest never hurt. Each additional tree either helps (by further reducing variance) or is neutral. There is no need for early stopping.

Scale invariance: Trees split on rank order, not absolute values, so they are invariant to monotonic transformations of predictors. No need to standardize features.

Robustness: Random Forests are relatively robust to hyperparameter choices. The main tuning parameters are m (features per split) and minimum leaf size, but performance is often good across a range of values.

4 Gradient Boosting

Random Forests address the variance problem by averaging the predictions of many independent trees. Gradient Boosting takes a fundamentally different approach: it addresses the *bias* problem by sequentially correcting errors.

4.1 The Boosting Idea

The core idea of boosting is intuitive: fit trees to *residuals*. Start with a simple prediction (the overall mean). Compute the residuals—the errors from this initial prediction. Fit a tree to predict these residuals. Add this tree's predictions to the current model. Compute the new residuals. Fit another tree. Repeat.

Each tree "boosts" the model by correcting the mistakes of all previous trees. After B iterations, the prediction is:

$$\widehat{f}(x) = \sum_{b=1}^{B} h_b(x)$$

where each h_b is a tree fit to the residuals from the previous iteration.

4.2 Why Residuals?

Fitting to residuals is a form of gradient descent in function space. For squared error loss, the negative gradient of the loss with respect to the current prediction is exactly the residual: $r_i = y_i - \widehat{f}(x_i)$. Fitting a tree to the residuals and adding it to the model moves in the direction that most rapidly decreases the loss.

More formally, gradient boosting minimizes a loss function $L(y, f(x))$ by iteratively adding functions that point in the negative gradient direction. For squared error, this reduces to fitting residuals. For other loss functions (absolute error, quantile loss), the procedure generalizes naturally.

4.3 The Gradient Boosting Algorithm

The algorithm proceeds as follows:

Algorithm 1 Gradient Boosting for Regression

1: Initialize $\widehat{f}^{(0)}(x) = \bar{y}$ (mean of training targets)
2: **for** $b = 1$ to B **do**
3: Compute residuals: $r_i = y_i - \widehat{f}^{(b-1)}(x_i)$ for all i
4: Fit a shallow tree $h_b(x)$ to residuals $\{(x_i, r_i)\}_{i=1}^{n}$
5: Update: $\widehat{f}^{(b)}(x) = \widehat{f}^{(b-1)}(x) + \nu \cdot h_b(x)$
6: **end for**
7: **Output:** $\widehat{f}(x) = \bar{y} + \nu \sum_{b=1}^{B} h_b(x)$

The learning rate $\nu \in (0, 1]$ controls how much each tree contributes. Smaller ν means more conservative updates—each tree makes a smaller correction. This is a form of regularization: by taking small steps, we are less likely to overshoot and overfit.

A key difference from Random Forests: gradient boosting uses *shallow* trees ("stumps" with depth 1–4, or slightly deeper trees with depth 4–6), not deep trees.

Why the difference? Random Forests rely on averaging to reduce variance, so individual trees should be complex (low bias), and the averaging handles the variance. Gradient boosting relies

on sequential correction to reduce bias, so individual trees should be simple (low variance), and the sequential fitting handles the bias.

Tree depth in gradient boosting also controls the order of interactions. A depth-1 tree (stump) captures only main effects. A depth-2 tree can capture pairwise interactions. A depth-d tree can capture up to d-way interactions.

4.4 Early Stopping

Unlike Random Forests, gradient boosting *can* overfit with too many trees. Each tree is designed to correct remaining errors, but eventually it starts correcting noise rather than signal. The training error continues to decrease, but the test error begins to increase.

The solution is **early stopping**: monitor the validation error during training and stop when it starts to increase. This is another form of regularization—we limit the effective number of trees based on out-of-sample performance.

The two ensemble strategies have complementary strengths:

	Random Forest	**Gradient Boosting**
Tree construction	Parallel (independent)	Sequential (dependent)
Individual trees	Deep (low bias)	Shallow (low variance)
Ensemble goal	Variance reduction	Bias reduction
Overfitting risk	Low	Higher (needs early stopping)
Tuning difficulty	Easier	Harder

In practice, gradient boosting (especially implementations such as XGBoost) often achieves slightly higher predictive accuracy when carefully tuned. But Random Forests are more robust to default settings and easier to use. Both are standard tools—the best practice is to try both and compare out-of-sample performance.

5 Can Trees Improve Market Timing?

We now return to the equity premium prediction problem from Lecture 2. Recall the sobering results: OLS catastrophically overfit, Ridge provided the best balance of statistical and economic performance, and Lasso essentially collapsed to predicting the historical mean.

The motivation for trying tree-based methods is clear. Linear models might miss important nonlinearities and interactions. Maybe D/P predicts differently at extreme values. Maybe the effect of the term spread depends on the inflation regime. Trees can discover such patterns automatically.

But will they? The promise of flexibility must be weighed against the reality of low signal-to-noise ratios and limited sample sizes in financial data.

5.1 Implementation Details

We use the same Goyal, Welch, and Zafirov (2024) dataset: monthly S&P 500 excess returns from 1953 to 2021, with 25 predictor variables. The evaluation follows the same walk-forward protocol:

- Initial training window: 120 months (10 years)
- Expand training window each month, predict the next month
- Hyperparameters selected via time-series cross-validation

- Benchmark: historical mean forecast

For tree methods, the key hyperparameters are:

- **Decision Tree:** Maximum depth (2–6), minimum samples per leaf (5–20)
- **Random Forest:** Number of trees (100, fixed), maximum depth (3–7 or unlimited), features per split ($\sqrt{p}$, $p/3$, $p/2$), minimum samples per leaf (5–20)
- **Gradient Boosting:** Number of iterations (50–200), learning rate (0.01–0.1), maximum depth (2–4)

6 Statistical Performance

The results are illuminating:

Model	R^2_{OS} (%)	Assessment
OLS	-13.85	Severe overfitting
Ridge	-0.50	Slight underperformance
Lasso	$+0.10$	$\approx$ Historical mean
Elastic Net	$+0.14$	$\approx$ Historical mean
Decision Tree	-10.68	Severe overfitting
Random Forest	$\mathbf{+0.15}$	Best
Gradient Boosting	-0.88	Moderate overfitting

Several patterns emerge:

Single trees overfit as badly as OLS. The decision tree achieves $R^2_{OS} = -10.7\%$, comparable to OLS at -13.8%. Flexibility without ensemble averaging leads to fitting noise.

Random Forest achieves the best statistical accuracy. It is the only complex model with positive R^2_{OS} ($+0.15\%$). The combination of averaging and decorrelation effectively controls variance.

Gradient Boosting disappoints. At $R^2_{OS} = -0.88\%$, it underperforms Ridge. Sequential bias reduction is less useful when the signal is weak—there is not much bias to correct, and the sequential procedure can overfit.

Lasso and Elastic Net match the historical mean. Their aggressive shrinkage sets most coefficients to zero, effectively predicting a near-constant value.

Beyond R^2, we can ask: how often does each model get the direction right?

Model	Sign Accuracy (%)	When Up (%)	When Down (%)
Historical Mean	59.4	100.0	0.0
Ridge	58.1	73.2	36.0
Lasso	59.5	100.0	0.3
Decision Tree	56.5	74.2	30.8
Random Forest	58.0	82.8	21.7
Gradient Boosting	56.3	86.1	12.6

The "When Up" column shows accuracy conditional on the market actually going up; "When Down" shows accuracy conditional on the market going down.

Lasso achieves the highest overall sign accuracy (59.5%) but is almost always bullish—it rarely predicts negative returns. This makes sense: it has essentially collapsed to predicting the historical mean, which is always positive.

Tree methods are more balanced, predicting negative returns 13–31% of the time when the market actually falls. But this balanced prediction does not necessarily translate to better returns.

7 Economic Performance

We translate predictions into portfolio positions using the mean-variance optimal weight:

$$\omega_t = \frac{1}{\gamma} \frac{\widehat{r}_{t+1}}{\widehat{\sigma}_t^2}$$

with risk aversion $\gamma = 5$, estimated variance $\widehat{\sigma}_t^2$ from a rolling 60-month window, and position constraints $-1 \leq \omega_t \leq 1.5$ (allowing 100% short to 150% long).

Strategy	Ann. Return (%)	Volatility (%)	Sharpe	Sharpe Net
Buy & Hold	6.9	14.8	0.46	0.46
Historical Mean	4.3	11.1	0.39	0.38
Ridge	11.9	17.1	**0.69**	**0.53**
Random Forest	7.0	14.4	0.49	0.35
Gradient Boosting	5.1	14.4	0.36	0.22

The results are striking: **Ridge dominates economically despite having worse R^2_{OS} than Random Forest.**

Ridge achieves an annualized return of 11.9% with a gross Sharpe ratio of 0.69 and a net Sharpe (after 50 bps transaction costs) of 0.53. Random Forest, despite its positive R^2_{OS}, achieves only a 0.49 gross Sharpe and 0.35 net Sharpe.

7.1 The R^2 vs. Economic Performance Puzzle

This is a puzzle worth understanding deeply. How can Random Forest have better statistical accuracy (R^2_{OS}) but worse economic performance (Sharpe ratio)?

R^2_{OS} measures average squared error, treating all errors equally. But economic value depends on the *magnitude* of correct predictions, not just their frequency.

Random Forest predictions are conservative. Averaging many trees pushes predictions toward the center. Extreme predictions from individual trees get diluted. This is good for R^2—small predictions mean small squared errors—but bad for returns. Small predictions lead to small portfolio positions, which generate small returns even when correct.

Ridge takes bigger bets. When predictor values are extreme, Ridge makes more aggressive forecasts. Higher volatility in predictions leads to larger portfolio positions. When these predictions are correct, the returns are substantial.

Random Forest succeeds at variance reduction—that is its purpose. But in a low signal-to-noise environment, variance reduction can go too far. By averaging predictions toward zero, Random Forest loses the ability to make bold calls that generate returns.

This is a fundamental tension. For statistical accuracy, conservative predictions are optimal when the signal is weak. For economic value, you sometimes need to take positions based on imperfect signals.

Another perspective: Random Forest has more accurate predictions on average (higher "win rate" in R^2 terms), but Ridge has larger payoffs when correct. A strategy that is right 50% of the time but wins big can outperform a strategy that is right 55% of the time but wins small.

This mirrors a common pattern in financial markets: the most profitable strategies are not necessarily the most accurate. What matters is the combination of accuracy and payoff magnitude.

8 Why Don't Trees Dominate?

We expected trees to help because they capture nonlinearities and interactions automatically. The results suggest this promise is not fully realized for market timing. Why?

The Signal Is Too Weak: Monthly return volatility is about 4–5%. The predictable component is perhaps 0.5%, if it exists at all. The signal-to-noise ratio is extremely low. In this environment, any nonlinearities in the true relationship are buried in noise. Trees find patterns—but those patterns are mostly noise, not signal. The flexibility that allows trees to discover complex relationships also allows them to fit spurious patterns.

Sample Size Is Limited: We have about 800 months of data. This may seem like a lot, but trees require more data than linear models to detect complex patterns reliably. With only a few hundred independent observations, there simply may not be enough information to distinguish true nonlinearities from sampling variation.

Linear Approximation May Be Sufficient: Perhaps the true relationship is approximately linear, or the nonlinearities are small enough that linear models capture the economically relevant variation. Occam's razor suggests preferring simpler models when they perform comparably.

8.1 When Do Trees Excel?

Trees tend to outperform linear models when:

- **True interactions are strong:** Credit risk (income × debt ratio), trading signals (volume × momentum)
- **Nonlinearities are pronounced:** Threshold effects, saturation
- **Sample size is large:** Cross-sectional prediction with thousands of stocks per period
- **Signal-to-noise is higher:** Default prediction ($R^2 \approx$ 10–30%) vs. return prediction ($R^2 \approx$ 1–2%)

Market timing has weak signals, small samples (one market), and limited evidence of strong nonlinearities. This is precisely the setting where trees are least likely to help.

9 Summary and Looking Ahead

This lecture has covered tree-based methods—from single decision trees to Random Forests and Gradient Boosting. Let us recap the key lessons:

Trees partition feature space and predict within regions. They automatically capture nonlinearities and interactions without requiring us to specify them.

Single trees are unstable. Small changes in data lead to completely different trees. This high variance makes them unsuitable for prediction on their own.

Random Forests reduce variance by averaging decorrelated trees. Bootstrap sampling plus feature subsampling creates diverse trees whose errors partially cancel.

Gradient Boosting reduces bias by sequential error correction. Shallow trees fit residuals iteratively, with each tree correcting the mistakes of previous trees.

For market timing, trees do not clearly dominate. Random Forest achieves the best R^2_{OS}, but Ridge achieves the best economic performance. The mismatch arises because statistical accuracy and economic value depend on different aspects of predictions.

The value of ML depends on the application. Market timing has weak signals and limited data—simpler methods suffice. Cross-sectional prediction and credit risk have stronger signals and more data—trees and other ML methods provide more value.

In the next lecture, we turn from regression to classification. Instead of predicting continuous returns, we will predict binary outcomes: will this borrower default? This is a setting where trees often excel, and we will see how the methods we have developed adapt to classification problems.

Readings

Required:

- James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). *An Introduction to Statistical Learning with Applications in Python*, Chapter 8.

Supplementary:

- Breiman, L. (2001). "Random Forests." *Machine Learning*, 45(1), 5–32. The original Random Forest paper.
- Gu, S., Kelly, B., & Xiu, D. (2020). "Empirical Asset Pricing via Machine Learning." *The Review of Financial Studies*, Section 1.6. Evidence on tree methods for return prediction.
- Goyal, A., Welch, I., & Zafirov, A. (2024). "A Comprehensive 2022 Look at the Empirical Performance of Equity Premium Prediction." *The Review of Financial Studies*. The data and evaluation framework for market timing.