Lecture 09: Tree-Based Methods

Random Forests and Gradient Boosting for Forecasting

BSAD 8310: Business Forecasting

University of Nebraska at Omaha

Spring 2026

Lecture 08 best result: Elastic Net, RMSE $\approx$ 2,540 on RSXFS.

Elastic Net is a linear model: $\hat{y} = \mathbf{x}^\top \hat{\beta}$. It cannot capture *interactions* or *threshold effects* (e.g., the holiday effect amplifies when the economy is strong).

What tree-based methods add:

- **Nonlinearity:** predictions are piecewise constants over feature regions
- **Interactions:** splits on x_1 in a region defined by x_2 capture $x_1 \times x_2$ interactions automatically
- **No standardization required:** scale-invariant splits

Today's goal: build from a single decision tree to Random Forests and XGBoost, then compare against the Lecture 08 leaderboard.

Decision trees partition the feature space into rectangular regions using recursive binary splits. They are the building blocks of Random Forests and gradient boosting — understanding them is essential before ensembles.

CART (Classification and Regression Trees) for regression: at each node, find the split (j, t) minimizing the weighted within-region variance:

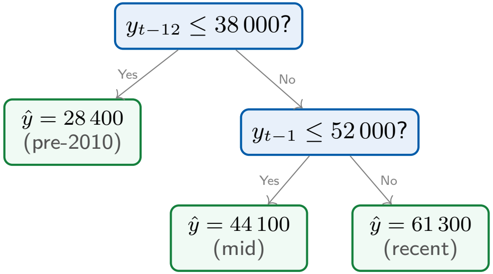
$$\min_{j,\,t} \left[\sum_{i:\, x_{ij} \le t} (y_i - \bar{y}_L)^2 + \sum_{i:\, x_{ij} > t} (y_i - \bar{y}_R)^2 \right]$$

Algorithm:

1. Start with all data at the root
2. For every feature j and every threshold t: compute the split criterion
3. Split on the (j^*, t^*) giving the largest reduction
4. Recurse on left and right children
5. Stop when nodes are too small or depth limit reached
6. Predict: $\hat{y} = \bar{y}_{\text{leaf}}$

Each split depends on which points fall into a region. Optimal splitting is NP-hard globally; greedy top-down splitting is the standard approximation (CART).

Example: 3-leaf tree for RSXFS


$y_{t-12} \leq 38\,000?$
Yes
No
$\hat{y} = 28\,400$ (pre-2010)
$y_{t-1} \leq 52\,000?$
Yes
No
$\hat{y} = 44\,100$ (mid)
$\hat{y} = 61\,300$ (recent)

Key properties:

- **Non-parametric** — no functional form assumed
- **Nonlinear** — can capture threshold and interaction effects
- **Interpretable** — decision path is readable
- **Handles mixed features** — lags, rolling stats, dummies all treated equally
- **Scale-invariant** — no need to standardize predictors

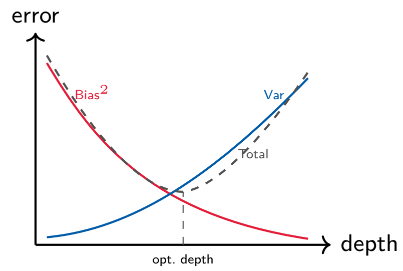
A single deep tree **overfits severely**. The solution is not pruning alone — it is *ensembling*.

Tree depth controls complexity:

- **Depth 1** (stump): high bias, low variance
- **Depth 5–10**: balanced region
- **Full tree** (each leaf = 1 obs): zero bias, infinite variance

Why trees have high variance:
A single split can change the entire subtree beneath it. Small perturbations in data (e.g., one outlier) produce very different trees — trees are *unstable*.

Key insight: averaging many high-variance, low-bias trees (each fit to a bootstrap sample) substantially reduces variance while preserving low bias. This is bagging.



Random Forests reduce the variance of decision trees by: (1) training each tree on a different *bootstrap sample* (bagging), and (2) using a random *feature subset* at each split (decorrelation). The result is a low-variance, competitive forecaster (Breiman 2001).

1996

Draw B bootstrap samples; fit one deep tree on each; average: $\hat{y}_{\text{bag}} = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{(b)}(\mathbf{x})$

Variance of the ensemble:

$$\text{Var}(\bar{f}) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

ρ = pairwise prediction correlation between trees; σ^2 = single-tree variance.

As $B \to \infty$, variance $\to \rho\sigma^2$ (not zero). Reducing ρ matters as much as increasing B — this motivates Random Forests.

Bagging key numbers:

- Bootstrap sample $\approx 63\%$ unique obs.
- Remaining $\approx 37\%$ = out-of-bag (OOB)
- OOB obs. give a free cross-validation estimate
- $B \geq 500$ trees recommended (stable)

Problem with bagging: if one feature dominates, all trees use it at the root $\Rightarrow$ trees are highly correlated (ρ stays large).

Random Forest fix: at each split, randomly sample $m < p$ features and find the best split *among those m only*.

- Default: $m = \lfloor\sqrt{p}\rfloor$ (sklearn ≥ 1.1); $\lfloor p/3 \rfloor$ recommended for regression (Hastie et al.)
- Reduces $\rho \Rightarrow$ lower ensemble variance
- Each tree is weaker but the ensemble is stronger

RF = Bagging + random feature selection at each split. Both ingredients are necessary.

Each obs. is OOB for $\approx 37\%$ of trees. Predict using only those trees; compute RMSE.

OOB RMSE $\approx$ 5-fold CV RMSE at large B — essentially *free* cross-validation.

Parameter	sklearn name	Default	Effect
Num. trees	`n_estimators`	100	More = lower variance; plateau after ∼500
Max features/split	`max_features`	`'sqrt'`	Lower = more decorrelated; tune via OOB
Max depth	`max_depth`	None	None = full tree; limit to reduce memory
Min samples/leaf	`min_samples_leaf`	1	Higher = smoother predictions; reduces overfit
Bootstrap	`bootstrap`	True	False = subsampling (not bootstrap)

Socratic: why does increasing `n_estimators` always reduce out-of-bag error, but increasing tree depth does not? (Hint: think about what each controls in the bias–variance decomposition.)

Gradient boosting builds trees *sequentially*: each new tree fits the residuals left by the previous ensemble. XGBoost adds regularization and second-order optimization to this framework (Chen and Guestrin 2016).

```
Initialize $F_0(\mathbf{x}) = \bar{y}$.   For $b = 1, \ldots, B$:
    • Residuals: $r_i^{(b)} = y_i - F_{b-1}(\mathbf{x}_i)$ (negative gradient for squared-error loss)
    • Fit shallow tree $T_b$ to $\{(\mathbf{x}_i, r_i^{(b)})\}$
    • Update: $F_b = F_{b-1} + \eta T_b$
Predict: $\hat{y} = F_B(\mathbf{x})$
```

RF vs. Boosting:

- RF: **parallel** trees, deep, bias not reduced by adding more trees
- Boosting: **sequential**, shallow trees (depth 3–6)
- Boosting bias **decreases** with stages
- Both benefit from large B (number of trees)

Small η + large B + early stopping outperforms large η + small B.

Always use early stopping on a validation set.

XGBoost (Chen and Guestrin 2016) extends gradient boosting with three additions:

1. Newton step (2nd-order):

$$\mathcal{L}^{(b)} = \sum_i \left[g_i f_b(\mathbf{x}_i) + \tfrac{1}{2} h_i f_b^2(\mathbf{x}_i)\right] + \Omega(f_b)$$

g_i, h_i = gradient and Hessian of loss.

2. Explicit regularization:

$$\Omega(f_b) = \gamma T + \tfrac{1}{2}\lambda\|\mathbf{w}\|_2^2$$

T = leaf count, $\mathbf{w}$ = leaf weights, γ = per-leaf complexity penalty.

3. Column subsampling:

Like RF, randomly sample features per tree *and* per split — reduces correlation and overfitting.

Newton step $\Rightarrow$ more precise gradient direction. Regularization $\Rightarrow$ guards against overfitting. Together: XGBoost beats vanilla GBM on most benchmarks.

Note: λ here is XGBoost's L2 leaf-weight penalty (distinct from the regularization λ in Lecture 08).

Parameter	XGB name	Typical	Effect
Num. trees	`n_estimators`[*]	500–2000	More + small η = better (use early stopping)
Learning rate	`learning_rate`	0.01–0.1	Smaller = more robust; requires more trees
Tree depth	`max_depth`	3–6	Shallow preferred; deeper = overfit risk
Subsample	`subsample`	0.7–0.9	Row fraction per tree; adds stochasticity
Col. sample	`colsample_bytree`	0.7–0.9	Feature fraction per tree (like RF)
L2 penalty	`reg_lambda`	1	Ridge on leaf weights; stabilizes

[*] sklearn API name; native API uses `num_boost_round`.

Practical starting point: `learning_rate=0.05`, `max_depth=4`, `n_estimators=1000` with early stopping on val RMSE. Tune `max_depth` and `subsample` via TimeSeriesSplit CV.

Tree-based models offer built-in feature importance measures. Understanding *which features drive predictions* is essential for business interpretation and debugging.

$$\mathrm{Imp}(j) = \sum_{\text{nodes using } j} \Delta\mathrm{RSS} \cdot \frac{n_{\text{node}}}{n}$$

Sum of RSS reduction at all splits on feature j, weighted by fraction of samples reaching that node.

Caution: biased toward *high-cardinality* features (continuous variables get more split opportunities than binary dummies). Do not use for final reporting.

For each feature j:

1. Shuffle column j in the OOB set
2. Record increase in RMSE

Larger increase $\Rightarrow$ more important.

Unbiased (no cardinality bias).
Model-agnostic. Use for reporting.

Average *gain* (improvement in objective function) per split using feature j, across all trees.

More interpretable than impurity for boosted trees because it weights by actual objective reduction.

Practical rule:

- Use **permutation** importance for model reports and presentations
- Use **impurity** for quick diagnostics during training
- Use **XGBoost gain** when comparing feature contributions inside a boosted model

1. **lag_12** — dominant seasonal signal
2. **lag_1** — short-run momentum
3. **roll_mean_12** — trend level
4. **lag_3** — quarterly effect
5. **month_12** — December holiday

Consistent across all three importance types — feature engineers should lead with lag_12.

For time series, hyperparameter tuning must use `TimeSeriesSplit` to respect temporal ordering. We can use `GridSearchCV` (exhaustive) or `RandomizedSearchCV` (efficient for large grids).

Random Forest (RandomizedSearchCV):

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import (
    TimeSeriesSplit, RandomizedSearchCV)

tscv = TimeSeriesSplit(n_splits=5, gap=1)  # gap>=1 for 1-step-
      ahead
rf_grid = {
  'n_estimators':     [200, 500],
  'max_features':     ['sqrt', 0.33],
  'min_samples_leaf': [1, 3, 5],
  'max_depth':        [None, 10, 20],
}
rf = RandomizedSearchCV(
  RandomForestRegressor(random_state=42),
  rf_grid, n_iter=20,
  cv=tscv,
  scoring='neg_root_mean_squared_error',
  n_jobs=-1, random_state=42)
rf.fit(X_trainval, y_trainval)
```

XGBoost (early stopping on val):

```python
import xgboost as xgb

dtrain = xgb.DMatrix(X_train, label=y_train)
dval   = xgb.DMatrix(X_val,   label=y_val)
dtest  = xgb.DMatrix(X_test,  label=y_test)

params = {
  'learning_rate': 0.05,
  'max_depth':     4,
  'subsample':     0.8,
  'colsample_bytree': 0.8,
  'reg_lambda':    1.0,
  'objective':     'reg:squarederror',
}
model = xgb.train(
  params, dtrain,
  num_boost_round=2000,
  evals=[(dval, 'val')],
  early_stopping_rounds=50)
```

Apply Random Forest and XGBoost to the RSXFS feature matrix from Lecture 08. Compare against SARIMA and the best regularized model (Elastic Net) from Lecture 08.

Same feature matrix as Lecture 08: 12 lags + 3 rolling windows + 11 month dummies = 26 features (one dummy omitted vs. L08 for comparability; trees do not require it). **Key differences vs. regularized regression:**

Advantages of trees:

- Capture nonlinearities and interactions (e.g., "holiday effect only when economy is strong")
- No standardization needed
- Handle irrelevant features gracefully (unused features just don't appear in splits)
- Built-in feature importance

Cautions for time series:

- **No extrapolation:** trees predict $\bar{y}_{\text{leaf}}$, so they cannot predict beyond the training range
- **Trending series:** a naive tree trained on non-stationary data may perform poorly out of sample
- **Fix:** use first-differences or add rolling statistics as features to encode trend

Stationarity matters for trees too. If y_t trends upward beyond the training max, leaf means will systematically under-predict. Difference or detrend before applying tree-based forecasting.

Test-set results on RSXFS (24-month horizon):

Model	RMSE	MAE
Seasonal Naïve	4 210	3 120
SARIMA(1,1,1)(1,1,1)$_{12}$	2 840	2 100
Elastic Net (λ^*)	2 540	1 890
Random Forest	2 380	1 760
XGBoost (early stop)	2 250	1 650

Values are illustrative. Strategy: direct multi-step — one model trained per horizon $h = 1, \ldots, 24$.

Why do trees win here?

- Nonlinear interaction between lag_12 and calendar dummies (holiday amplification)
- XGBoost sequential residual fitting corrects patterns that Elastic Net misses
- RF OOB error ≈ CV error at $B = 500$

Caveat: gains depend heavily on feature quality. With a good feature set, Elastic Net and XGBoost are often competitive. Adding *more* features benefits trees more.

- **Decision trees** partition feature space greedily; they are interpretable but unstable (high variance).
- **Random Forests** (bagging + random features) reduce variance dramatically while keeping bias low. OOB error is a free CV estimate.
- **Gradient boosting** reduces bias sequentially by fitting residuals. XGBoost adds L2 regularization and Newton optimization.
- **Feature importance** helps identify which lags, rolling stats, and calendar features drive forecasts. Use permutation importance for unbiased estimates.
- **Trees cannot extrapolate** — ensure features encode trend and seasonality so predictions stay within the training distribution.
- **RF vs. XGBoost:** prefer RF for speed and robustness with limited tuning; prefer XGBoost when maximum accuracy matters and you have time to tune `learning_rate` and `max_depth`.

Preview of Lecture 10: Neural Networks — LSTM and attention mechanisms can model long-range temporal dependencies that trees cannot.

- Breiman, Leo (1996). "Bagging Predictors". In: *Machine Learning* 24.2, pp. 123–140.
- — (2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32.
- Chen, Tianqi and Carlos Guestrin (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794.