Tree-Based Methods: CART, Bagging, Random Forests, Boosting

Key Formulas & Definitions

Pierre Pudlo

1 Tree-Based Methods: Overview

Tree-based methods stratify or segment the feature space into simple regions. Decision rules can be summarized in a tree structure, making them interpretable but often less accurate than other methods when used individually.

Solution: Combine many trees through ensemble methods like bagging, random forests, and boosting.

Advantages: Simple, interpretable, handle qualitative features naturally, no need for dummy variables.

Disadvantages: High variance, lower predictive accuracy (for single trees), not robust to small changes in data.

2 CART: Classification and Regression Trees

2.1 Recursive Binary Splitting

Goal: Divide the feature space into J non-overlapping regions $R_1, \dots, R_J$ that minimize:

$$\text{RSS} = \sum_{j=1}^{J} \sum_{i: x_i \in R_j} (y_i - \hat{y}_{R_j})^2$$

Algorithm: At each step, for every feature X_j and cutpoint s:

1. Define: $R_1(j,s) = \{X : X_j \leq s\}$ and $R_2(j,s) = \{X : X_j > s\}$
2. Find (j,s) that minimizes: $\sum_{i \in R_1} (y_i - \hat{c}_1)^2 + \sum_{i \in R_2} (y_i - \hat{c}_2)^2$
3. Repeat on each resulting region until minimum node size reached

2.2 Cost-Complexity Pruning

Large trees overfit. **Solution:** Prune using cost-complexity criterion:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$$

where $|T|$ = number of leaves, N_m = observations in leaf m, $Q_m(T)$ = within-leaf variance, $\alpha \geq 0$ = complexity parameter.

Choosing α via K-fold Cross-Validation:

1. For each fold k: build tree $\mathcal{T}_0^{-k}$, prune to get $\mathcal{T}_\alpha^{-k}$, evaluate on fold k
2. Compute CV error: $\text{CV}(\alpha) = \frac{1}{K} \sum_{k=1}^{K} \text{MSE}_k(\alpha)$
3. Select $\alpha_{\min}$ that minimizes $\text{CV}(\alpha)$

1-SE Rule (recommended): Choose largest α such that $\text{CV}(\alpha) \leq \text{CV}(\alpha_{\min}) + \text{SE}(\alpha_{\min})$ for simpler, more interpretable trees.

2.3 Classification Trees

Prediction: Assign class with highest proportion in leaf: $\hat{k}(m) = \arg\max_k \hat{p}_{mk}$ where $\hat{p}_{mk} = \frac{1}{N_m} \sum_{i \in R_m} \mathbb{1}(y_i = k)$

Node Impurity Measures:

- **Gini index:** $G_m = \sum_k \hat{p}_{mk}(1 - \hat{p}_{mk})$
- **Cross-entropy:** $D_m = -\sum_k \hat{p}_{mk} \log(\hat{p}_{mk})$
- **Classification error:** $E_m = 1 - \max_k \hat{p}_{mk}$

Use Gini or cross-entropy for tree growing (more sensitive to node purity).

3 Bootstrap and Bagging

3.1 Bootstrap

Idea: Sample n observations **with replacement** from original data to create new dataset $\mathcal{D}^*$.

Key fact: Each bootstrap sample contains ~63.2% unique observations (probability an observation is excluded: $(1 - 1/n)^n \to e^{-1} \approx 0.368$). Remaining 36.8% are **out-of-bag (OOB)** observations.

3.2 Bagging (Bootstrap Aggregating)

Motivation: Reduce variance. For i.i.d. $Z_1, \ldots, Z_n$ with variance σ^2: $\text{Var}(\bar{Z}) = \sigma^2/n$.

Algorithm:

1. For $b = 1, \ldots, B$: generate bootstrap sample $\mathcal{D}^{*b}$, fit model $\hat{f}^{*b}$
2. **Regression:** $\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$
3. **Classification:** majority vote

Variance reduction: If correlation between trees is ρ: $\text{Var}(\hat{f}_{\text{bag}}) \to \rho\sigma^2$ as $B \to \infty$

OOB Error: For observation i, average predictions from trees where i was OOB: $\hat{y}_i^{\text{OOB}} = \frac{1}{|\mathcal{B}_i|}\sum_{b \in \mathcal{B}_i} \hat{f}^{*b}(x_i)$. Then $\text{MSE}_{\text{OOB}} = \frac{1}{n}\sum_i (y_i - \hat{y}_i^{\text{OOB}})^2$ provides validation without separate test set.

4 Random Forests

4.1 Key Idea

Problem with bagging: Trees are correlated (same strong predictors used) → limited variance reduction.

Random Forest solution: At each split, randomly select m features (out of p total) to consider.

- **Typical choices:** $m \approx \sqrt{p}$ (classification), $m \approx p/3$ (regression)
- Forces diversity → decorrelates trees → smaller ρ → lower variance

Algorithm: Same as bagging, but at each node of each tree, randomly select m features before finding best split. **Do not prune trees.**

4.2 Hyperparameters

- **Features per split m:** Tune using OOB error
- **Number of trees B:** 100-500 (more is better, diminishing returns)
- **Minimum node size:** Default usually works

4.3 Variable Importance

Measure: For each feature X_j, compute total decrease in RSS (regression) or Gini index (classification) from splits on X_j, averaged across all trees.

Advantages: Excellent performance, handles high dimensions, OOB validation, robust to overfitting, works well with defaults.

Disadvantages: Less interpretable, slower than single tree, requires more memory.

5 Boosting

5.1 The Boosting Idea

Key difference from bagging/RF: Build trees **sequentially**, each correcting errors of previous trees.

Principles:

1. **Learn slowly:** Each tree corrects current ensemble's errors
2. **Weak learners:** Use shallow trees (low variance, high bias)
3. **Sequential:** Each tree focuses on previous mistakes
4. **Additive:** New trees added to ensemble

5.2 Algorithm for Regression

Input: Training data, number of trees B, learning rate λ, tree depth d

Initialize: $\hat{f}(x) = \bar{y}$, residuals $r_i = y_i - \bar{y}$

For $b = 1, \ldots, B$:

1. Fit tree $\hat{f}^b$ with d splits to predict residuals r_i
2. Update residuals: $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$

Output: $\hat{f}(x) = \bar{y} + \lambda \sum_{b=1}^{B} \hat{f}^b(x)$

Effect: Reduces **bias** (unlike bagging which reduces variance).

5.3 Boosting as Gradient Descent

Boosting minimizes loss $L(\hat{f}) = \sum_i \ell(y_i, \hat{f}(x_i))$ via gradient descent in function space.

For squared loss $\ell(y, \hat{f}) = \frac{1}{2}(y - \hat{f})^2$:

- Gradient: $\frac{\partial L}{\partial \hat{f}(x_i)} = -(y_i - \hat{f}(x_i)) = -r_i$
- Update: $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$ where $\hat{f}^b$ fits residuals r_i

Interpretation: Each tree approximates negative gradient, λ is step size.

Generalization: Extends to any differentiable loss (classification, ranking, etc.) $\rightarrow$ **Gradient Boosting Machines**

5.4 Key Hyperparameters

1. **Number of trees B:** Can overfit if too large. Use CV or early stopping. Typical: 100-5000.
2. **Learning rate λ:** Controls learning speed. Small λ needs large B. Typical: 0.01-0.1.
3. **Tree depth d:** Controls complexity. $d = 1$ (stumps, no interactions), $d = 2$ (2-way interactions). Typical: 1-6.
4. **Subsampling:** Fraction of data per tree. Typical: 0.5-0.8. Reduces variance.
5. **Column sampling:** Random features per tree (like RF). Decorrelates trees.

Important: Boosting is sensitive to hyperparameters. Careful tuning essential!

5.5 XGBoost (Extreme Gradient Boosting)

Key improvements:

- **Regularization:** L1/L2 penalties on leaf weights: $\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2$
- **Second-order gradient:** Uses Hessian for better approximation
- **Efficient sparse data handling**
- **Parallel processing:** Fast tree construction
- **Built-in CV and early stopping**
- **Missing value handling:** Learns best direction

State-of-the-art for tabular data in ML competitions.

6 Summary and Practical Recommendations

6.1 When to Use Each Method

Single Tree: Need interpretability, quick baseline, small datasets, domain requires explainability.

Bagging: Reduce variance of high-variance models, stable predictions needed.

Random Forest: Default choice. High-dimensional data, accuracy paramount, need feature importance, don't need interpretability. Works well with defaults.

XGBoost: Maximum predictive performance, tabular data, ML competitions, willing to tune carefully, have computational resources.

6.2 Practical Workflow

1. **Start simple:** Fit single tree as baseline, and linear model if appropriate
2. **Try Random Forest:** Good defaults, fast to run, usually works well
3. **Try XGBoost:** For better performance (requires more tuning)
4. **Validate properly:**
   - Use OOB error for RF (no separate validation needed)
   - Use CV for hyperparameter selection
   - Use early stopping for boosting
   - Always evaluate on held-out test set

6.3 Key Hyperparameter Tuning

Random Forest:

- Main: `max_features` ($\sqrt{p}$ or $p/3$)
- Less critical: `n_estimators` (100-500)

XGBoost:

- Critical: `learning_rate`, `max_depth`, `n_estimators`
- Important: `subsample`, `colsample_bytree`
- **Use early stopping with validation set**

6.4 Resources

- Hastie, Tibshirani, Friedman (2009). *The Elements of Statistical Learning* https://hastie.su.domains/ElemStatLearn/
- James, Witten, Hastie, Tibshirani (2023). *An Introduction to Statistical Learning (with Applications in Python)* https://www.statlearning.com/
- scikit-learn: https://scikit-learn.org/
- XGBoost: https://xgboost.readthedocs.io/