Tree-based Methods and Boosting

PS690 Computational Methods in Social Science

Jiawei Fu
Department of Political Science
Duke University

Sep 9, 2025

Overview

1. Tree-based Methods

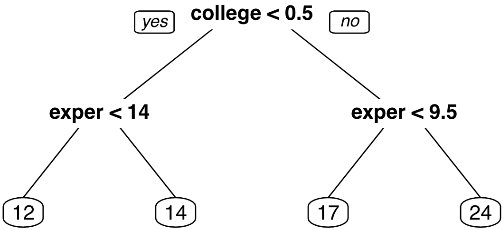
2. Bagging

3. Random Forests

4. Boosting

Decision Trees

- The linear methods we have studied are not well suited to capturing the underlying nonlinear relationships between the response and the predictors.
- We also want to allow for interactions among the predictors.
- Although we can add higher-order terms and interaction terms in a linear model, we prefer to let the data itself reveal the appropriate structure.
- A popular class of models is tree-based methods.



Building a Tree

- Our general goal: Divide the predictor space $(X_1, X_2, ..., X_p)$ into J distinct regions and make one prediction for each region.
- Formally, we hope to find regions $R_1, ..., R_J$ that

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

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- Solution: Top-down, greedy approach, known as recursive binary splitting
  - Top-down: begins at the top of the tree
  - Greedy: the best split is made at the *local* step

Classification and Regression Trees (CART)

- We select a predictor X_j and cut point s, splitting the predictor space into two regions

$$R_1(j,s) = \{X|X_j < s\} \; and \; R_2(j,s) = \{X|X_j \geq s\}$$

- Q: Which variable is to be split? and What is the criterion to split?
- Objective function:

$$\min_{j,s}[\min_{c_1} \sum_{i:x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{i:x_i \in R_2(j,s)} (y_i - c_2)^2]$$

- Given j, s, it is clear that $\hat{c}_k = \mathbb{E}_n[y_i|x_i \in R_k(j,s)]$ for $k = 1, 2$.
- For each j, finding s is easy. Then go through all j.

CART

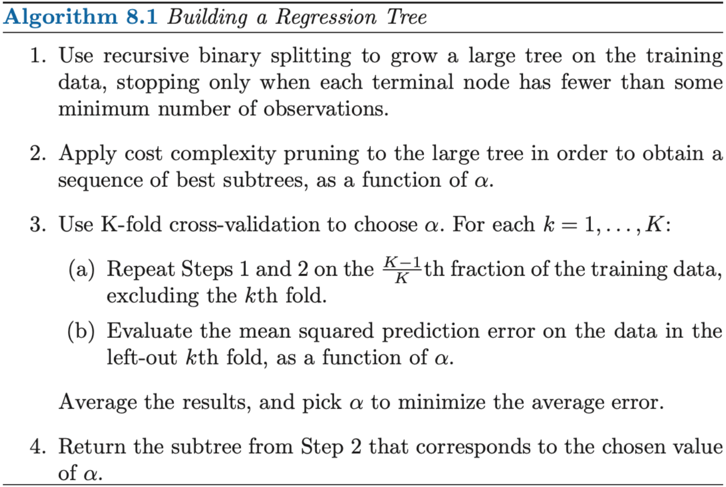
- Repeat the splitting process on each of the two regions, and continue.
- Q: When to stop? Or, say, what is the tree size?
  - Too deep (large): high variance, low bias; overfit
  - Too shallow (small): low variance, high bias

1. Stop until the decrease in RSS is too small
   - Problem: too short-sighted since a seemingly worthless split early on in the tree might be followed by a very good split
2. Better solution: grow a very large tree, stopping the splitting process only wehn some minimum node size (say 5) is reached, and then prune it
   - Cost complexity pruning: For each tuning parameter α, find the unique subtree T_α s.t.

     $$\min \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

     where $|T|$ denote the number of terminal nodes in T, m denotes the terminal nodes.
   - The first term is just the training error. Therefore, larger α increases the price to pay for having a tree with many terminal nodes.

CART



Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Figure: From [James et al., 2013]

Advantage and Disadvantage trees

- Advantage
  - Interpretable and seems to be close to human decision-making.
  - They are insensitive to monotone transformations of the inputs (because the split points are based on ranking of the data points).
  - They perform automatic variable selection.
- Disadvantage
  - High variance: a small change in the data can result in a very different series of splits.
  - Why? Hierarchical nature of the process: the effect of an error in the top split is propagated down to all of the splits below it.
  - Lack of smoothness.

Bagging

- Bootstrap aggregation: a general-purpose procedure for reducing the variance of a statistical learning method
  - Given a set of n iid observations with variance σ^2, the variance of the mean is $\frac{\sigma^2}{n}$
  - In other words, averaging a set of observations reduces variance.
  - Natural Idea: take many training sets from the population, build separate models, and average them.
  - However, it is not practical because we generally have single one training set.
  - Solution: Bootstrap!

Note

Bagging does not reduce bias. Therefore, each tree should be grown deep, and are not pruned. Hence each individual tree has high variance, but low bias.

Bagging

- Bagging may not work well if trees are highly correlated!
- Suppose the positive pairwise correlation ρ, the variance of the average is $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$
- This may happen if there is one very strong predictor.
- Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split.
- Solution: De-correlation!
- BTW: Bagging is a frequentest concept. Bayesian approach: Bayesian adaptive regression trees (BART)

Random Forests

- That is a pretty dramatic name, right?
- To de-correlate trees:
  - a random sample of $m = \sqrt{p}$ predictors is chosen as split candidates from the full set of p predictors.
  - The split is allowed to use only one of those m predictors.
  - A fresh sample of m predictors is taken at each split.
- The estimator can be written as

$$\hat{F}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{T}_b(x)$$

  where $\hat{T}_b(x)$ is a tree estimator based on a subsample (or bootstrap) of size s using m randomly selected features. The trees are usually required to have some number k of observations in the leaves.

- There are three tuning parameters: s, m, k.

Cross-validation of Random Forests

- It turns out that there is a very straightforward way to estimate the test error of a bagged model, without the need to perform cross-validation.
- Not all observations are sampled in one subset. The remaining observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the i-th observation using each of the trees in which that observation was OOB.
- An OOB error estimate is almost identical to that obtained by N-fold cross-validation
- Hence unlike many other nonlinear estimators, random forests can be fit in one sequence, with cross-validation being performed along the way.

Inference

- Under some regular conditions, including honesty (we will introduce it in details in the later lectures), subsampling of size $s = n^{\beta}$, $\beta < 1$, [Wager and Athey, 2018] show that $\frac{\hat{F}(x) - F(x)}{\sigma(x)} \rightarrow N(0,1)$
- They also shows that $\hat{\sigma}(x) = \frac{n-1}{n}(\frac{n}{n-s})\sum_{i=1}^{n}[Cov(\hat{T}_b(x), N_{ib})]^2$, where N_{ib} indicate whether or not the i-th training example was used for the b-th tree.
- Random forests are considered one of the best all purpose classifiers. But it is still a mystery why they work so well.

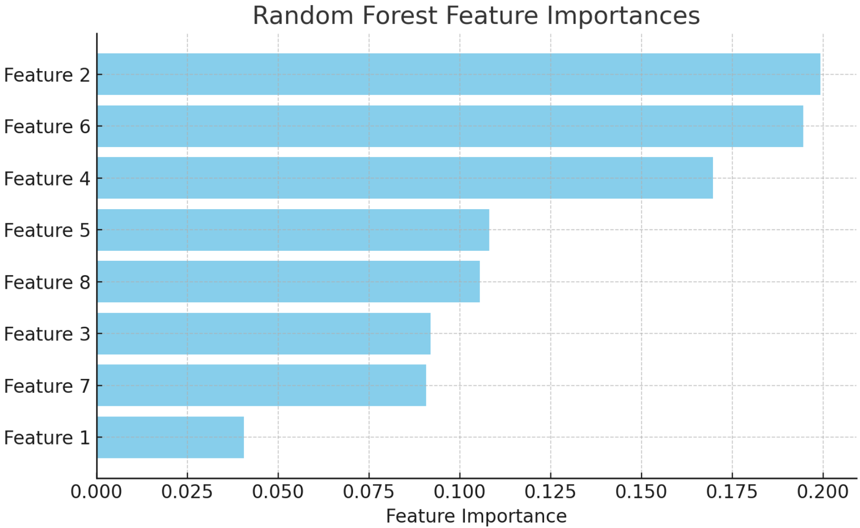
Variable Importance Measures

- How important is feature X_j?
- One intuitive way to answer this is to fit the forest with all the data and fit it again without using X_j.
- If leaving out a covariate X_j barely changes predictive accuracy, its “true” contribution is small.
- Practically, a random forest builds many trees; because for each tree, we randomly subsampling some features, we already have lots of trees that never saw a given variable during construction.
- This method is called LOCO, Leave-Out-COvariates. For each X_j, we compare the inflation of the prediction error by not having access to X_j, and then the rank of importance is based on the inverse rank of this value.

Variable Importance Measures

- A different approach is called Permutation Feature Importance.
- We measure the increase in the prediction error of the model after we permute the values of the feature in the test set.
- A feature is 'important' if shuffling its values increases the model error, because in this case, the model relied on the feature for the prediction.
- A feature is 'unimportant' if shifting its values leaves the model error unchanged because, in this case, the model ignored the prediction feature.

Permutation feature importance



Permutation feature importance

- Let $\hat{f}$ be the trained model, and let L be the loss.
- Step 1: estimate the prediction error:

$$error_{orig} = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} L(y_i, \hat{f}(\mathbf{x}_i))$$

e.g. mean squared error.

- For each feature j do:
  - permute feature X_j, generate $\mathbf{x}_i^{perm,j}$ (so that breaks the association between X_j and outcome y)
  - estimate $error_{perm}^{j} = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} L(y_i, \hat{f}(\mathbf{x}_i^{perm,j}))$
  - calculate permutation feature importance as quotient $\frac{error_{perm}^{j}}{error_{orig}}$ or difference $error_{perm}^{j} - error_{orig}$

Boosting

- Booting is one of the most powerful learning ideas introduced in the last twenty years.
- We estimate a simple prediction rule, then take the residuals and estimate another simple prediction rule for these residuals. Keep repeating.
- Consider additive basis-function model (ABM): $f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$
- These models are fit by minimizing a loss function:

$$\min_{f} \sum_{i=1}^{N} L(y_i, f(x_i)) = \min_{\beta_m, \gamma_m} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m b(x; \gamma_m))$$

  This is very hard problem.
- Boosting find each $b()$ sequentially, by an algorithm called a weak learner.
- Then, applying the weak learner sequentially to weighted versions of the data.
- More weight is given to data that does not learned well before.

Boosting

- We can tackle it sequentially. We initialize a $f_0(x)$, which is a simple solution. For example, $f_0(x) = 0$, $f_0(x) = \overline{y}$, or using GLM.
- For $m = 1$ to M, compute

$$(\beta_m, \gamma_m) = \text{arg } min \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$

- Then, set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$
- This method is called forward stagewise additive modeling.
- We continue this for a fixed number of iterations M. In fact M is the main tuning parameter of the method.

Boosting

- For example, consider square-error loss: $L(y, f(x)) = (y - f(x))^2$
- Then,

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (residual_{im} - \beta b(x_i; \gamma))^3 \end{aligned}$$

  $residual_{im} = y_i - f_{m-1}(x_i)$ is simply the residual on the ith observation.

- That is, we fit a model using the residuals.

Boosting

- How to choose M? If M is too large, it can lead to overfitting.
- Often we pick it by monitoring the performance on a separate validation set, and then stopping once performance starts to decrease; this is called early stopping.
- In practice, better (test set) performance can be obtained by performing "partial updates":

$$f_m(x) = f_{m-1}(x) + \nu \beta_m b(x; \gamma_m)$$

- $0 < \nu \leq 1$ is a step-size parameter. In practice it is common to use a small value such as $\mu = 0.1$.

Boosting Trees

- Note that a tree is an additive function: $T(x;\Theta) = \sum_{j=1} \hat{y}_{R_j} I(x \in R_j)$, where $\Theta = \{R_j, \hat{y}_{R_j}\}$, and

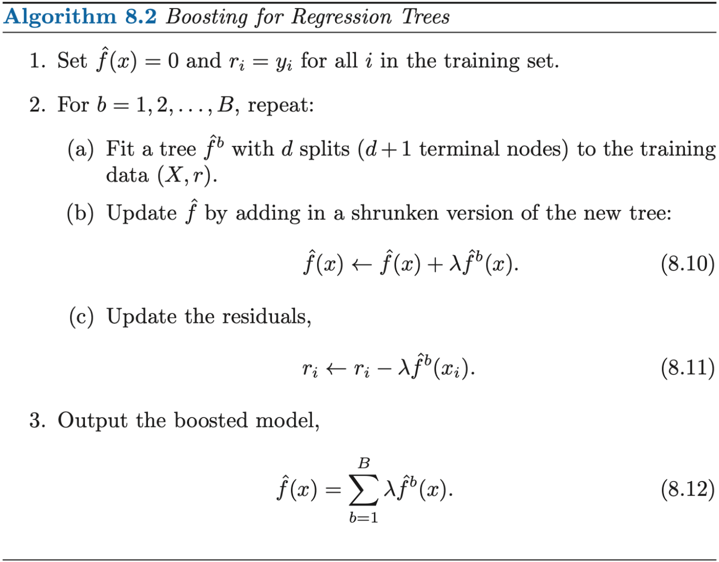
$$\hat{\Theta} = \arg\, min \sum_{j=1} \sum_{x_i \in R_j} L(y_i, \hat{y}_{R_j})$$

- For boosting trees, in each step, we solve

$$\hat{\Theta} = \arg\, min \sum_{j=1} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

- For squared-error loss, we just build tree for the residuals.

Boosting



Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

References

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013).
An introduction to statistical learning: with applications in R, volume 103.
Springer.

Wager, S. and Athey, S. (2018).
Estimation and inference of heterogeneous treatment effects using random forests.
Journal of the American Statistical Association, 113(523):1228–1242.