

# Part 12: Trees, Boosting, and Bagging

---

Chris Conlon

April 26, 2020

Applied Econometrics II

- Chapters X of *Elements of Statistical Learning*

# Trees

---

# Decision Trees

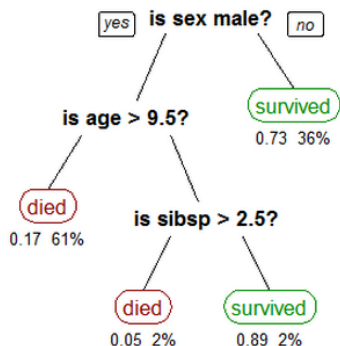
Start with  $y = f(x_i)$ :

- Construct a **tree** by **splitting** the sample on an  $x_i$ .
  - Choose the split to maximize the criterion function.
  - Choose the  $x_i$  to maximize the criterion given the proposed split.
- Which  $x_i$  do we choose?
  - There are  $K$  possibilities.
  - But how do we know which order to split?
- Which split do we choose?
  - This is usually single dimensional optimization.
- Resulting problem is  $NP$  hard.

What kind of tree?

- **Classification** Trees predict **discrete** outcomes.
- **Regression** Trees predict **continuous** outcomes.

# Decision Trees: Titanic Survival



A tree showing survival of passengers on the *Titanic* ("sibsp" is the number of spouses or siblings aboard). The figures under the leaves show the probability of survival and the percentage of observations in the leaf.

# Decision Trees

What is the criteria function?

- RSS:  $\min_f \sum_{i=1}^N (y - f(x_i))^2$
- Maximize Variance Reduction

$$I_V(N) = \underbrace{\frac{1}{|S|^2} \sum_{i \in S} \frac{1}{2} (x_i - x_j)^2}_{\text{Pre split Var}} - \left( \underbrace{\frac{1}{|S_t|^2} \sum_{i \in S_t} \sum_{j \in S_t} \frac{1}{2} (x_i - x_j)^2}_{\text{Post split TRUE}} + \underbrace{\frac{1}{|S_f|^2} \sum_{i \in S_f} \sum_{j \in S_f} \frac{1}{2} (x_i - x_j)^2}_{\text{Post split FALSE}} \right)$$

- Gini Impurity: Randomly label an element, odds of being correct?  $\sum_j p_j \times (\sum_{k \neq j} p_k) = \sum_j p_j (1 - p_j) = 1 - \sum_j p_j^2$
- Information Gain/Entropy:

$$H(T) = I_E(p_1, p_2, \dots, p_J) = - \sum_{i=1}^J p_i \log_2 p_i$$

# What are trees actually doing?

Think about the *RSS* case :  $\min_f \sum_{i=1}^N (y - f(x_i))^2$

- This is **nonparametric regression**
- It is also a **kernel model**.
- Partition the data:  $\{R_1, \dots, R_M\}$  into **leaves** that are **disjoint** and **span the space** of  $X$ .
- $f(x) = \sum_{m=1}^M c_m 1(x \in R_m)$ .
  - Where  $c_m$  is mean of  $y_i$  within leaf  $R_m$ .
- This is just a **locally constant** regression. But distance is not determined using bandwidth...

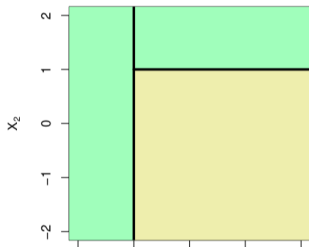
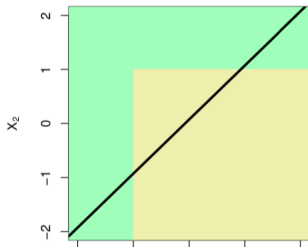
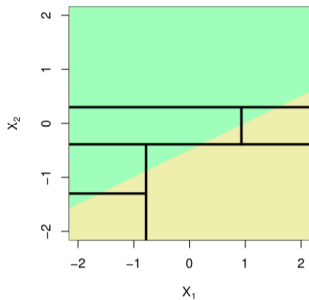
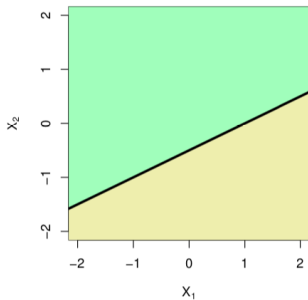


# What are trees actually doing?

When are trees the right tool?

- Because of multiple levels of splits: work best when true relationship is highly nonlinear.
- Who has a heart attack?
  - Lots of factors: high BP, overweight, age, family history, diabetes, etc.
  - Logit: these enter **additively** in the index and increase **log odds proportionally**.
  - Can interact multiple factors *highBP*  $\times$  *overweight*, etc.
- If the true model is highly interacted: trees will do well.
- In general trees have **low bias** but **high variance**
  - Small changes in data can lead to wildly different splits (and trees).

# Linear vs. Nonlinear Relationships



# How do splits work?

- Binary variables: Trivial
- Continuous Variables: Choose a split value  $s$  so that  $x > s$  or  $x \leq s$  optimizes your criteria.
  - This is a single dimensional search (Golden Section, etc.).
- Multiple Categories ( $A, B, C$ ): this is the hard case
  - Order by the outcome variable  $y_b > y_a > y_c$ .
  - Then treat like the continuous case.

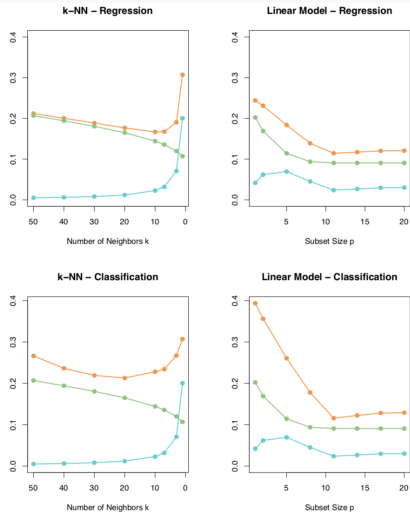
- Let  $|T| = M$  denote the number of terminal nodes in  $T$ . We will use  $|T|$  to measure the complexity of a tree. For any given complexity, we want the tree minimizing square error on training set.
- Finding the optimal binary tree of a given complexity is computationally intractable (NP hard).

# Tree Algorithms

How do we build a tree?

- Because the problem is *NP* hard: no ideal way.
- Mostly heuristics.
- Greedy Algorithm: Choose best split first for best  $x$  go from there.
  - Need not be optimal...
- Avoiding over-fitting
  - How deep to make tree?
  - Minimum number of observations per branch?
  - Pruning?
  - Trees will always overfit if you let them!

# Overfitting: Bias Variance



**FIGURE 7.3.** Expected prediction error (orange), squared bias (green) and variance (blue) for a simulated example. The top row is regression with squared error loss; the bottom row is classification with 0-1 loss. The models are  $k$ -nearest neighbors (left) and best subset regression of size  $p$  (right). The variance and bias

# Why Prune a Tree?

To avoid overfitting

- Any useful split will be made (eventually).
- We (may) end up with perfectly predicted outcomes  $E[y_i|x_i]$ . case we know for sure that  $y_i = 1$  ?
- Remove leaves with too few elements.
- Usually use a hold-out sample (test set) and remove leaves if it increases OOS criteria.
- Other canned pruning algorithms will generate several candidate subtrees
  - Used out of sample criteria to pick the best subtree
- Can also do early stopping

# Tree Algorithms

About **criteria** and **heuristics**.

- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)
- Chi-square automatic interaction detection (CHAID). Performs multi-level splits when computing classification trees.
- MARS: extends decision trees to handle numerical data better.
- Conditional Inference Trees. Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning.



# Bagging

---

What is Bagging? Bootstrap Aggregation.

- We re-sample a new dataset the size of our original dataset with replacement
- Before we re-ran our estimation to get  $\hat{\theta}^b$  and took  $Var(\hat{\theta}^{(1)}, \hat{\theta}^{(2)}, \dots, \hat{\theta}^{(B)})$  and  $E(\theta) = \frac{1}{B} \sum_{b=1}^B \hat{\theta}^b$ .
- Since we are doing ML, let's bag  $\hat{y}_i$ .
- $E(y) = \frac{1}{B} \sum_{b=1}^B \hat{y}^b$  (we could also do this conditionally).

## Bagging: What's the point?

Why would we want to do this?

- If we are worried about high-variance and **overfitting** bagging will reduce our variance.
- Each bootstrapped sample  $b$  is like an IID realization of our data.
- Averaging over samples can reduce the variance of  $\hat{y}$  by  $\frac{1}{B}$ .

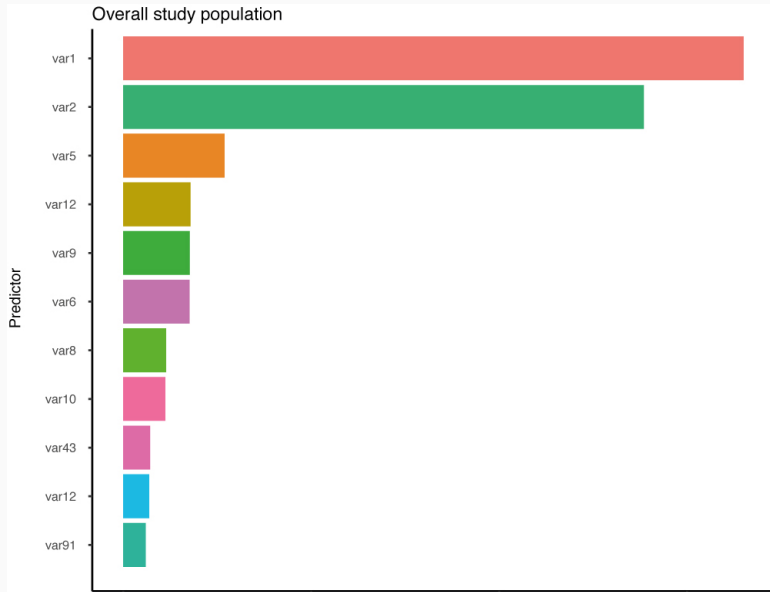
The problem **Interpretability**

- If my “model” is OLS I can report average coefficients.
- If my model is a tree, what do I even report averaged over 1000 bootstrapped simulations?

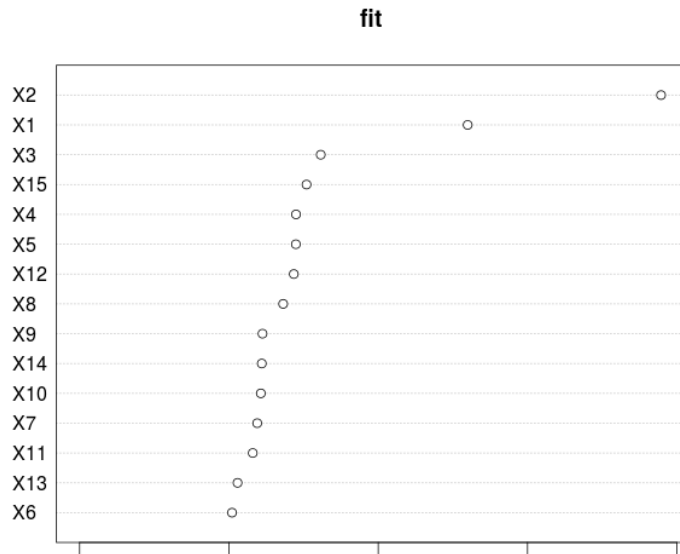
## How to interpret variables?

- Pick a tree  $T^b$  calculate how much a particular variable  $x_1$  increases the Gini Index, or decreases RSS, etc.
- Average over all trees (some trees will be zero if they don't include  $x_1$ ).
- Report the average in **relative** terms for all  $x_k$ .

# Variable Importance Plots



## Variable Importance Plots

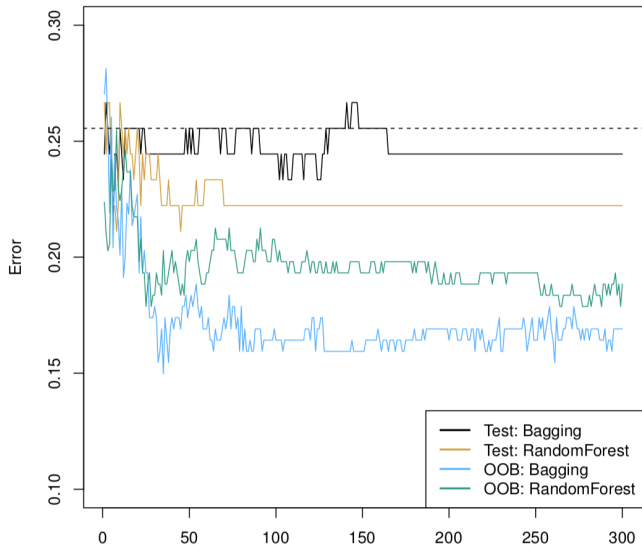


# Validation: How do assess model fit?

## Out of Bag Error Rate (OOB)

- Use **cross validation**
- Draw a bootstrap sample of size  $N$ .
- Split sample into two parts: training  $\frac{2}{3}$ , test  $\frac{1}{3}$
- Fit on the training sample
- Predict on the test sample:  $(y_i - \hat{y}_i^{OOB})^2$
- Report average or (mean-squared) error on the test sample only.
- Gives us expected out of sample fit.

## Bagging: OOB error





## Why do Bagging do so poorly?

- Even though we bootstrap our sample many times  $\rightarrow (T^B, T^{B'})$  are highly correlated.
- Why does this happen?
  - We end up with the same  $X$ 's and the same splits in each model.
- Adding more trees doesn't really improve forecast performance.
- We would like to add more trees but have them be **less correlated** with one another.

## Solution: Random Forests

1. Draw a bootstrap sample  $b$ .
2. Fit a tree (usually a small one with limited depth)
3. At each node select a **random subset** of regressors  $m < K$  from  $X$ .
4. Average the predictions of (hopefully) less correlated trees. (Or Majority Vote).

How to choose  $m$ ? Usually  $\sqrt{K}$ .

## Solution: Random Forests

- Even though we bootstrap our sample many times  $\rightarrow (T^B, T^{B'})$  are highly correlated.
- Adding more trees doesn't really improve forecast performance.
- We would like to add more trees but have them be **less correlated** with one another.

# Evaluating Random Forests

How important is  $X^{(j)}$ ? Look at the trees that randomly don't include it in  $m$  (or don't select it)

$$\hat{\Delta}_j = \frac{1}{m} \sum_{i \in \mathcal{H}} (Y_i - \hat{m}_{(-j)}(X_i))^2 - (Y_i - \hat{m}(X_i))^2$$

Average this quantity across the entire training set:

$$\mathbb{E} [\hat{\Delta}_j | \mathcal{T}] = \mathbb{E} [(Y - \hat{m}_{(-j)}(X))^2 - (Y - \hat{m}(X))^2 | \mathcal{T}] \equiv \Delta_j$$

This is called **Leave-One-Out-CO**variates (LOCO).

- Technically possible but tricky
- Active area of research
- See Wager and Athey (2017).

- Technically possible but tricky
- Active area of research
- See Wager and Athey (2017).

# What about Boosting?

Boosting works somewhat differently

1. Set  $\hat{f}(x) = 0$  and  $r_i = y_i$  for  $i = 1, \dots, N$ .
2. For  $b = 1, \dots, B$  iterate on:
  - 2.1 Fit a tree  $\hat{f}^b$  with  $d$  splits to the response  $r_1, \dots, r_n$ .
  - 2.2 Update the prediction to

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

- 2.3 Update the residual

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Produce the final model by averaging

$$\hat{f}(x) = \sum^B \lambda \hat{f}^b(x)$$

# How does Boosting Work?

- At each bootstrap iteration we aren't fitting  $y_i$
- We are fitting  $r_i$  the residual  $y_i - \sum_{b'=1}^b \lambda \hat{f}^{b'}$ .
- $\lambda$  is usually small so that we update the model **slowly**.
- You can think about what we are doing as **re-weighting our training data**
  - Put more weight on the cases we fit the worst (large residuals).
- Usually set  $\lambda$  small .01
- Well known algorithm **AdaBoost**

Boosting > Bagging > Single Tree



## AdaBoost: Freund & Schapire, 1996

1. Weight observations equally  $w_i = \frac{1}{N}$ . for all  $i = 1, \dots, N$ .
2. From  $b = 1, \dots, B$  do the following:
  - 2.1 Fit a model  $f^{(b)}(x_i)$  to training data using  $w_i$ .
  - 2.2 Compute  $err_b = \frac{\sum_{i=1}^N w_i I[y_i \neq f^{(b)}(x_i)]}{\sum_{i=1}^N w_i}$
  - 2.3 Compute  $\alpha_b = \log\left(\frac{1 - err_b}{err_b}\right)$
  - 2.4 Update weights for  $i = 1, \dots, N$ :

$$w_i \leftarrow w_i \cdot \exp[\alpha_b I(y_i \neq f^{(b)}(x_i))]$$

renormalize so that  $\sum w_i = 1$ .

3. Compute  $\hat{f}(x) = \text{sign}[\sum_b \alpha_b \hat{f}^{(b)}(x_i)]$ .

# Additive Models

- $\hat{f}(x) = \sum_b \alpha_b \hat{f}^{(b)}(x_i)$  is an **additive model**
- Lots of examples in the literature:
  - Basis Functions (ie: polynomials)  $\sum_{k=1}^K \theta_k g_k(x)$ .
  - GAMs:  $f(x) = \sum_{k=1}^K f_k(x)$ .
- Usually we fit these in **one step** (OLS, MLE, etc.)
- Here we fit  $\alpha_b, \theta_b$  in a **stagewise** manner.
- Process fits slowly but avoids overfitting problem.

# What about Gradient Boosting?

Same as AdaBoost with some modifications

- Loss function  $L(y_i, \gamma)$
- Choose  $\gamma$  to minimize  $\min L(y_i, \gamma)$  and get  $h_b(x)$ .
- Update  $r_i = \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]$  this is the **gradient** evaluated at  $b$ th iteration of  $F_b(x_i)$ .
- Update  $F_b(x) = F_{b-1}(x) + \gamma_b h_b(x)$ .

## How do we do this in R?

- Sample Splitting, Bagging, etc. `caret`, `sample`
- Gradient Boosting. `gbm` (easy), `xgboost` (faster).
- Random forest `randomForest`.