Chapter 8: Tree-based Models

Jeremy Selva

Introduction

This chapter introduces tree-based methods for regression and classification

- Decision Trees
- Bagging
- Random Forest
- Boosting
- Bayesian Additive Regression Trees

8.1.1 Regression Trees

1. Divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and **non-overlapping** regions, $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, where $1 \leq j \leq J$, take the **mean** of the response values for the training observations in R_j.

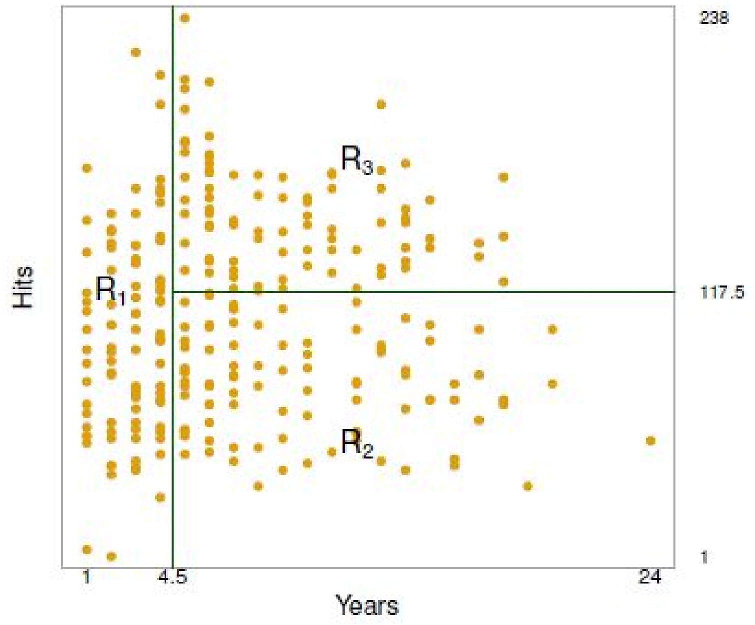


FIGURE 8.2. *The three-region partition for the* Hitters *data set from the regression tree illustrated in Figure 8.1.*

8.1.1 Regression Trees

The goal is to find boxes $R_1, \ldots, R_J$ that **minimize** the RSS, given by

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

where $\hat{y}_{R_j}$ is the **mean** response for the training observations within the jth box.

Unfortunately, it is **computationally infeasible** to consider every possible partition of the feature space into J boxes.

8.1.1 Regression Trees

One way is given all predictors $X_1, X_2, \ldots, X_p$, find the **best** cut off point s for a given predictor X_k, where $1 \leq k \leq p$ such that the resulting two regions

$$R_1(k,s) = \{X|X_k < s\} \text{ and } R_2(k,s) = \{X|X_k \geq s\}$$

have the **least** RSS for all predictors

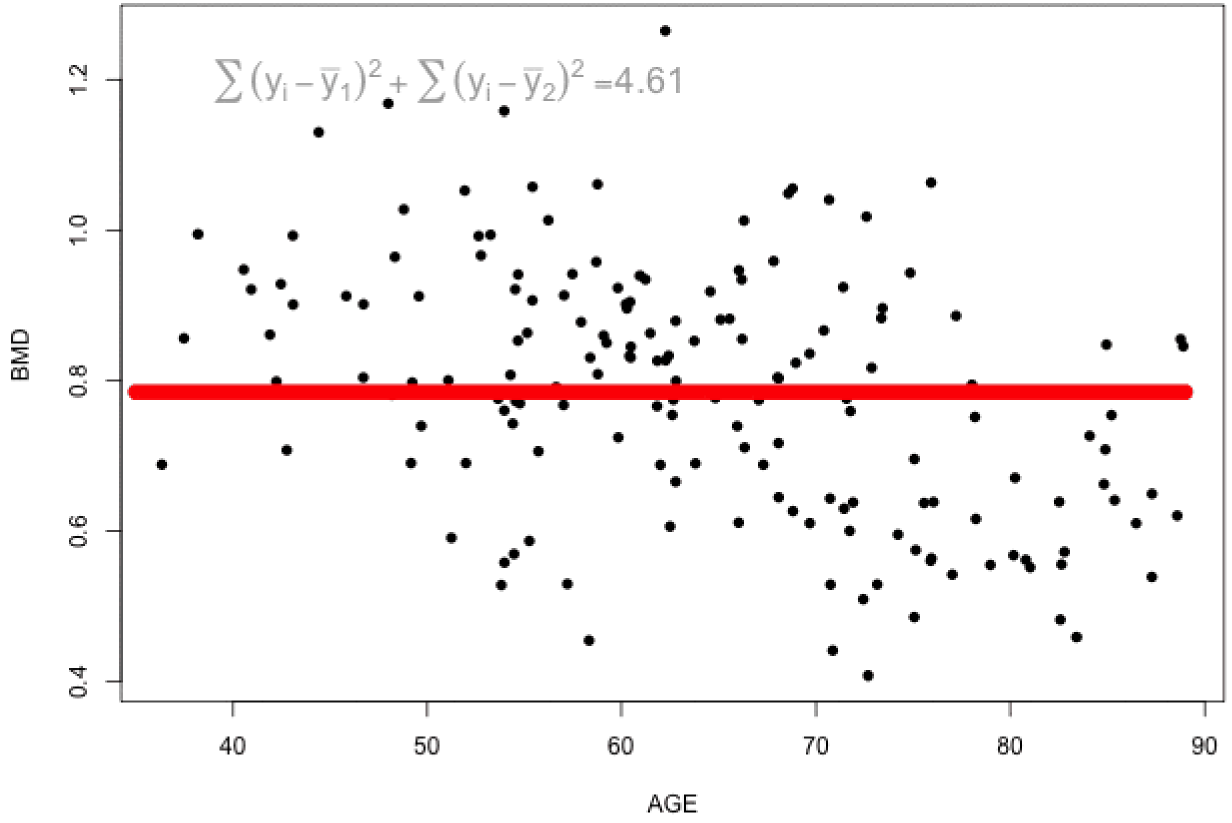
$$RSS = \sum_{i:x_i \in R_1(k,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(k,s)} (y_i - \hat{y}_{R_2})^2$$

where $\hat{y}_{R_1}$ is the **mean** response for the training observations in $R_1(k,s)$, and $\hat{y}_{R_2}$ is the **mean** response for the training observations in $R_2(k,s)$.

Repeat the process until no region contains more than five observations.

8.1.1 Regression Trees

See https://bookdown.org/tpinto_home/Beyond-Additivity/regression-and-classification-trees.html for animation.



8.1.1 Regression Trees

See StatQuest with Josh Starmer 22 min animation.

8.1.1 Regression Trees

However, the previous method may result in a tree that **overfits** the data.

A better strategy is to

1. grow a very large tree T_0 as before
2. tune α in the cost complexity pruning formula to prune/penalise T_0 to give **a sequence** of sub trees Ts.
3. apply cross validation or validation set on those sub trees Ts and pick α that gives the **best sub tree** (smallest RSS).

The cost complexity pruning formula is

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

where $\alpha \geq 0$, $|T|$ is the number of **terminal nodes** the sub tree T holds. R_m is the **box/region** corresponding to the mth terminal node and $y_i - \hat{y}_{R_m}$ is the **mean** response for the training observation in R_m

8.1.1 Regression Trees

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

When $\alpha = 0$, the sub tree $= T_0$ as the cost complexity pruning formula equals to the RSS.

The cost complexity **penalises** complex trees with a lot of terminal nodes.

Similar to lasso, as α increase, some terminal nodes will not be used.

8.1.1 Regression Trees

See StatQuest with Josh Starmer 16 min animation.

8.1.2 Classification Trees

In the case of classification, instead of taking the mean of a given region, we take the **most commonly occurring class** (or mode) of training observations in the region to which it belongs.

For the measurement of error, instead of RSS, one way is to use the fraction of the training observations in that region that do not belong to the most common class.

$$E = 1 - \max_k(\hat{p}_{mk})$$

where $\hat{p}_{mk}$ the **proportion of training observations** in the mth region that are from the kth class. Hence, $0 \leq \hat{p}_{mk} \leq 1$

However, this error rate is unsuited for tree-based classification because E does not change much as the tree grows (**lack sensitivity**).

Instead, **more sensitive** classification error measurements like the Gini index or the entropy are used.

8.1.2 Classification Trees

The Gini index and the entropy are very similar measures, where **small values** indicate that the node contains **large number of observations from a single class**.

The Gini index is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

The value varies between 0 and 1, where, 0 denotes that **all** observations belong to a certain class, 1 denotes that the observations are **randomly** distributed across various classes and 0.5 denotes **equally** distributed observations into some classes.

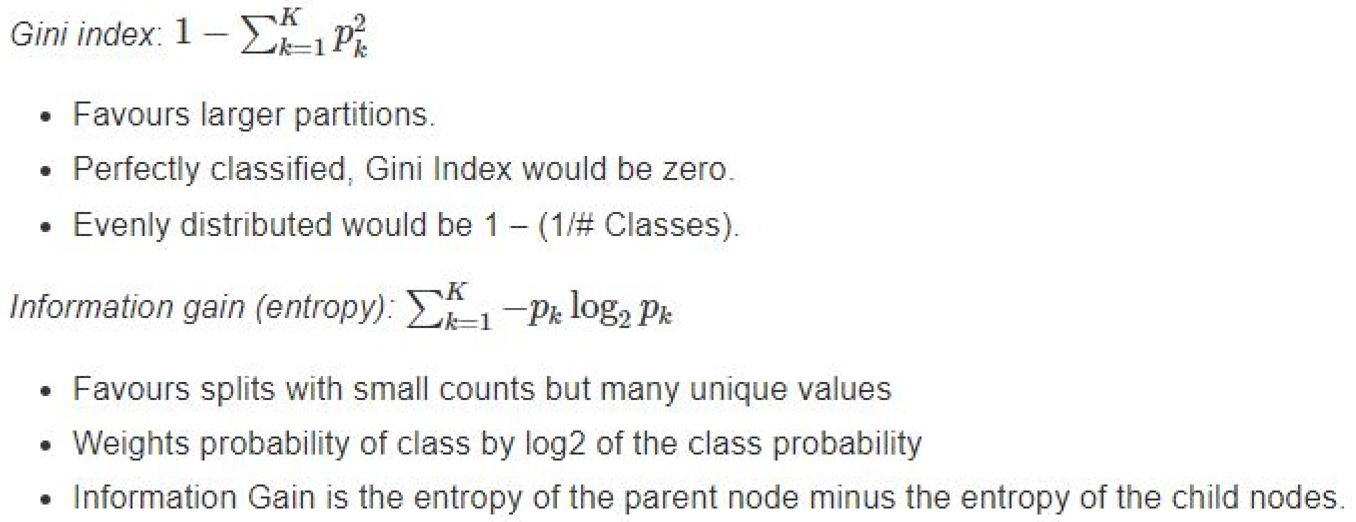
The entropy is defined by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$$

Both Gini index and Entropy are **small** when all of the $\hat{p}_{mk}$'s are close to **zero** or **one**.

8.1.2 Classification Trees

See https://bookdown.org/tpinto_home/Beyond-Additivity/regression-and-classification-trees.html



Gini index: $1 - \sum_{k=1}^{K} p_k^2$

- Favours larger partitions.
- Perfectly classified, Gini Index would be zero.
- Evenly distributed would be 1 – (1/# Classes).

Information gain (entropy): $\sum_{k=1}^{K} -p_k \log_2 p_k$

- Favours splits with small counts but many unique values
- Weights probability of class by log2 of the class probability
- Information Gain is the entropy of the parent node minus the entropy of the child nodes.

8.1.3 Trees Versus Linear Models

In terms of prediction accuracy, ... it depends

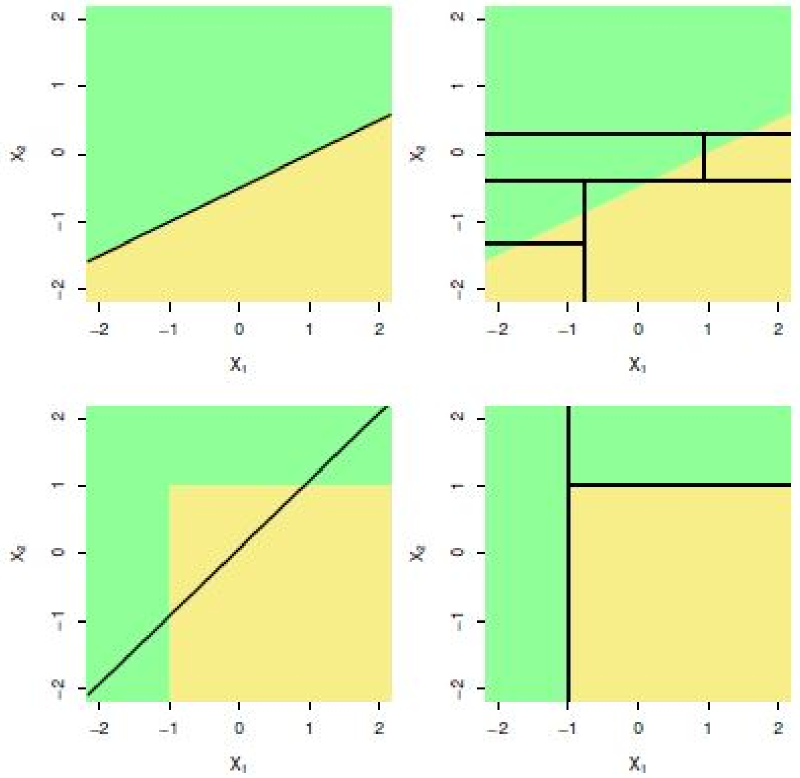


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: *Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).*

8.1.3 Trees Versus Linear Models

Compared to linear models, decision trees are **easier to explain** to others because they mirror human decision-making. They can be displayed graphically and easily interpreted by non-experts.

However, it is **less robust** than the linear models. A small change in the data set can cause a large change to the decision tree. This may hurt the decision trees performance in prediction.

To improve its performance, methods like bagging, random forests, and boosting are used (sometimes at the cost of interpretability).

8.2.1 Bagging

Bagging is an approach that tries to “stabilise”(make it more robust to change of data) the predictions from trees

The idea is to generate B different **bootstrapped training datasets** to create B different regression trees $\hat{f}^{*b}$ respectively for $1 \leq b \leq B$. Given a new observation to predict called x, we use the B regression trees to create B predictions. The final prediction value is calculated by **averaging** all the predictions.

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$$

In the case of classification, we record the class predicted by each of the B trees, and take a **majority** vote.

What is missing is how do we estimate the test error rate (without using the test data) to pick the best B value or number of bagged trees ?

8.2.1 Bagging

Each bagged tree is trained using **two-thirds** of the training data. The other one-third of the training data, known as **out-of-bag** or OOB observations are then used.

Given an OOB observation, we first predict its response **using only bagged trees** that do not use this OOB observation in its training. An OOB prediction is made by averaging the predicted response (regression) or a majority vote (classification) among these specific bagged trees.

As each OOB observation has an OOB prediction and a true response, the OOB error can be computed via the overall OOB RSS (regression) or classification error like Gini index. The B value that gives the **least** OOB error will be taken.

8.2.1 Bagging

Bagging usually improves prediction accuracy at the expense of interpretability.

It is no longer clear which predictors are more important as we now have more than one tree (which may gives differing views on the importance of a given predictor).

An overall summary of the importance of each predictor can be achieved by recording how much the average RSS or Gini index **improves (or decreases)** when each tree is splitted over a given predictor.

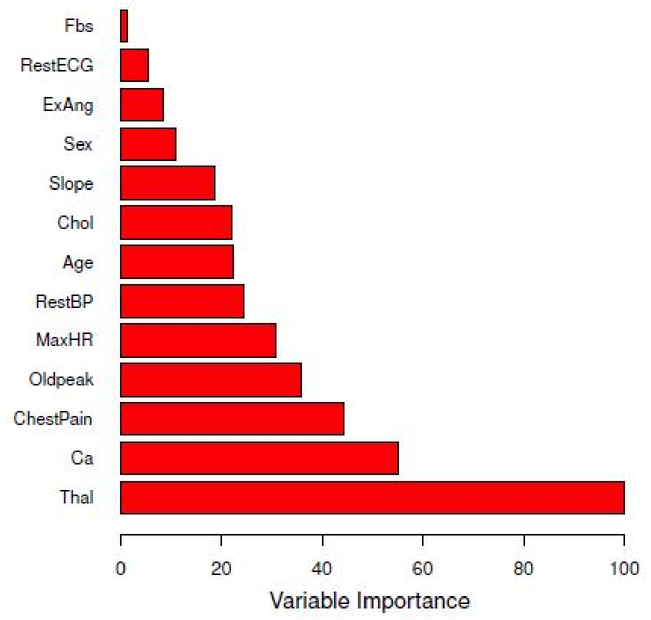


FIGURE 8.9. *A variable importance plot for the* Heart *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

8.2.2 Random Forests

One problem with bagging is that each bagged tree may be **highly similar** to one another.

This happens if there is a strong predictor in the data set. By the greedy principle, most of the bagged trees will **use this strong predictor** in the top split, regardless of changes in the training set led by bootstrapping.

This may cause the tree model to be **less robust** to changes in the data over time.

To further improve the robustness of the model, a **change** is made during the creation of a split in a given bagged tree on bootstrapped training samples.

Instead of using the full set of p predictors as split candidates, only **a random sample** of $m \approx \sqrt{p}$ predictors are used instead.

This reduces the odds of the strong predictor being selected during the split and provide other predictors a chance.

8.2.2 Random Forests

See StatQuest with Josh Starmer 10 min animation.

8.2.2 Random Forests

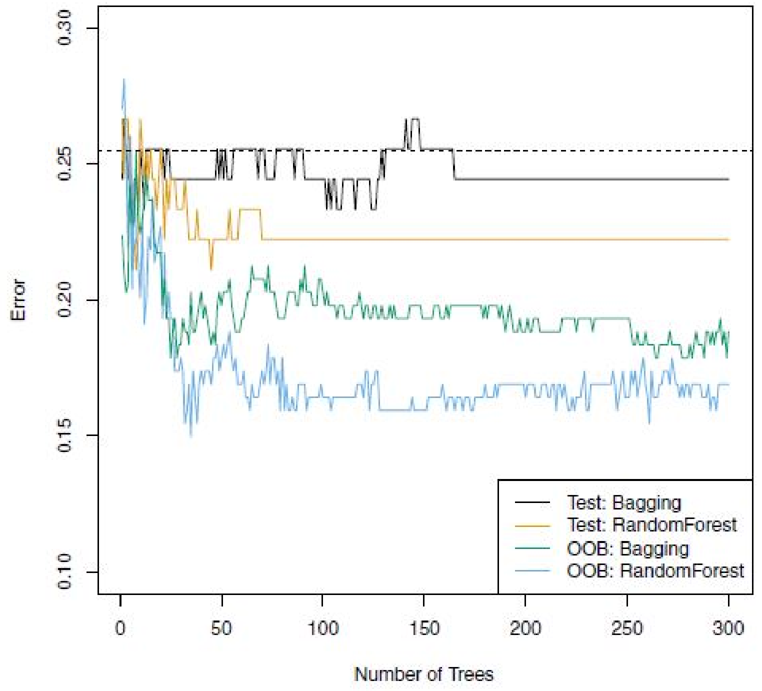


FIGURE 8.8. *Bagging and random forest results for the* Heart *data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is — by chance — considerably lower.*

8.2.3 Boosting

Recall in bagging, each tree is built on a bootstrap training data set.

On the other hand, in boosting, each tree is built on information that the previous trees are unable to catch. That is, we fit a tree using **the current residuals**, rather than the response.

This is similar to the partial least square components in which each component is build based on the response's residual left by the model created from the previous partial least square components.

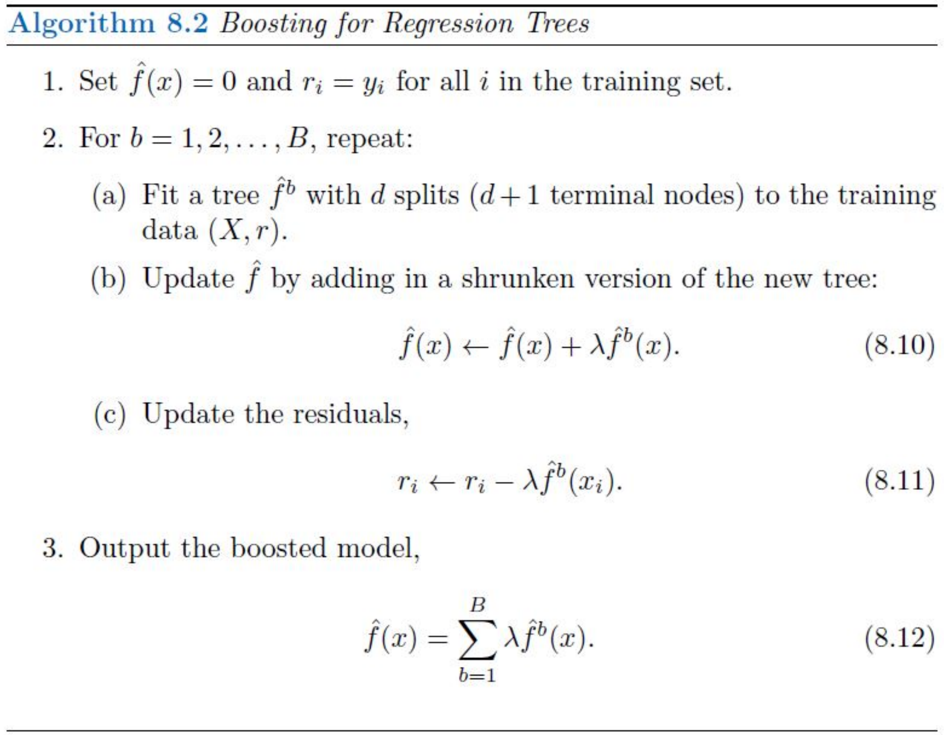
The key is to make each tree **small** (with a few terminal nodes) but still "different" from one another.

For simplicity, only boosting regression trees are described in the book.

8.2.3 Boosting

Boosting has three tuning parameters.

1. The **number of trees** B
2. The maximum number of **splits/depth** d each tree is allow to have. (to make each tree small)
3. A **shrinkage parameter or learning rate** λ (to make each tree "different" from one another)



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:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \quad (8.10)$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \quad (8.11)$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \quad (8.12)$$

8.2.3 Boosting

See StatQuest with Josh Starmer 16 min animation.

8.2.3 Boosting

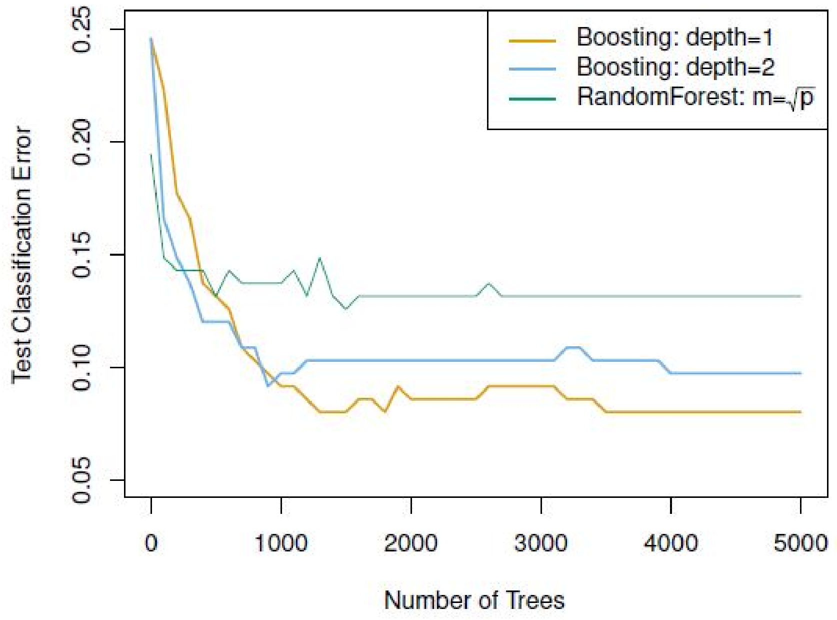


FIGURE 8.11. *Results from performing boosting and random forests on the 15-class gene expression data set in order to predict* cancer *versus* normal*. The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant. The test error rate for a single tree is 24 %.*

8.2.4 Bayesian Additive Regression Trees (BART)

For simplicity, only BART for regression are described in the book.

- In bagging, each tree is built on a **random set of training data**.
- In random forest, each tree also uses **a random set of predictors**.
- In boosting, each tree is built to capture **signal that is not yet accounted** for by the current set of trees

BART combines all these elements together.

8.2.4 Bayesian Additive Regression Trees (BART)

The book define some notation

- K be the total **number of regression trees** used to create the model (later by adding them together)
- B be the **number of iterations** the BART algorithm will run to improve the additive regression trees.
- $\hat{f}^b_k(x)$ be the **prediction** of an observation x for the kth regression tree used in the bth iteration of the BART algorithm, where $1 \leq k \leq K$ and $1 \leq b \leq B$
- L, where $L < B$ be the number of **burn-in** iterations.

Like boosting, the tree model is created by summing all regression trees at the end of each iteration.

$$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}^b_k(x)$$

for $b = 1, \ldots, B$

8.2.4 Bayesian Additive Regression Trees (BART)

In the first iteration of the BART algorithm, all K trees are initialized to to predict only **one** value: $\hat{f}_k^1(x) = \frac{1}{nK}\sum_{i=1}^n y_i$, the **mean** of the response values divided by the total number of trees.

Thus

$$\hat{f}^1(x) = \sum_{k=1}^K \hat{f}_k^1(x) = \sum_{k=1}^K \frac{1}{nK}\sum_{i=1}^n y_i = \frac{1}{n}\sum_{i=1}^n y_i$$

8.2.4 Bayesian Additive Regression Trees (BART)

In subsequent b iteration, where $2 \leq b \leq B$

Given the kth tree, we calculate for each observation i where, $1 \leq i \leq n$, a **partial residual** r_i is created by subtracting the response y_i with the predictions made from the **other** $K-1$ trees denoted as (k').

The other $K-1$ trees include those that have already been updated by BART $(k' < k)$ and those that will be updated by BART later $(k' > k)$.

$$r_i = y_i - \sum_{k'<k} \hat{f}^{\,b}_{k'}(x_i) - \sum_{k'>k} \hat{f}^{\,b-1}_{k'}(x_i)$$

8.2.4 Bayesian Additive Regression Trees (BART)

Next, BART will create **possible changes/pertubations** on the kth tree and assign a **probability** calculated by Bayesian methods (given the current kth tree terminal nodes or $\hat{f}_k^{b-1}(x)$ and partial residual r_i).

- Adding branches
- Pruning branches
- Change the prediction on terminal nodes
- Remain as they are

Changes that greatly improve the fit to the partial residual will be given a **higher probability**. However, penalty scores will be given when adding new branches to prevent the tree from growing too deep. Recall that unlike boosting, we do no have control on the depth

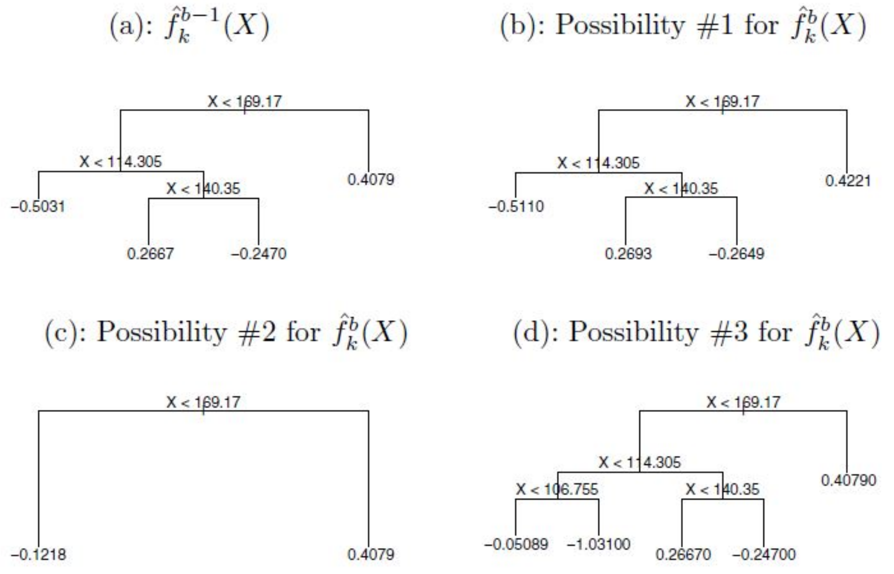


FIGURE 8.12. *A schematic of perturbed trees from the BART algorithm.* (a): *The* k*th tree at the* $(b-1)$*st iteration,* $\hat{f}_k^{b-1}(X)$*, is displayed. Panels* (b)–(d) *display three of many possibilities for* $\hat{f}_k^b(X)$*, given the form of* $\hat{f}_k^{b-1}(X)$. (b): *One possibility is that* $\hat{f}_k^b(X)$ *has the same structure as* $\hat{f}_k^{b-1}(X)$*, but with different predictions at the terminal nodes.* (c): *Another possibility is that* $\hat{f}_k^b(X)$ *results from pruning* $\hat{f}_k^{b-1}(X)$. (d): *Alternatively,* $\hat{f}_k^b(X)$ *may have more terminal nodes than* $\hat{f}_k^{b-1}(X)$.

8.2.4 Bayesian Additive Regression Trees (BART)

BART will then "roll a dice" to **choose** a perturbation on the kth tree as $\hat{f}_k^b(x)$. High probability does not mean it will get chosen.

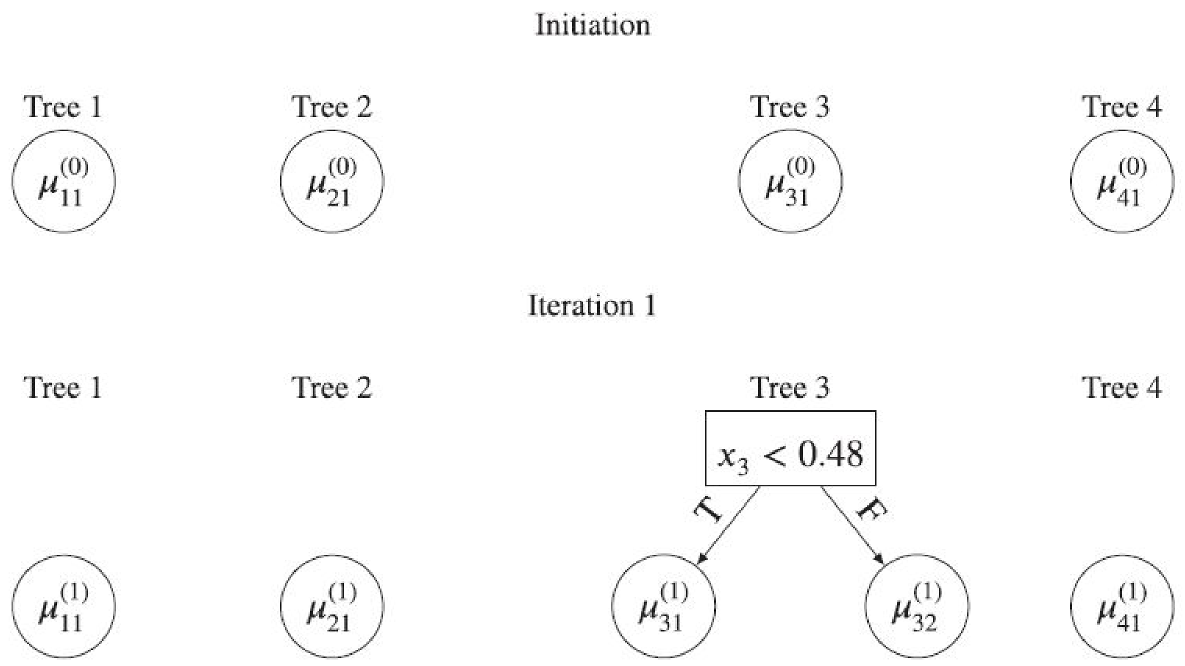
The **selected** tree $\hat{f}_k^b(x)$ will then we used to **fit** the partial residual r_i

This same process will be done for all K trees.

Adding up all the regression tree will giving the new tree model as $\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x)$ for the bth iteration.

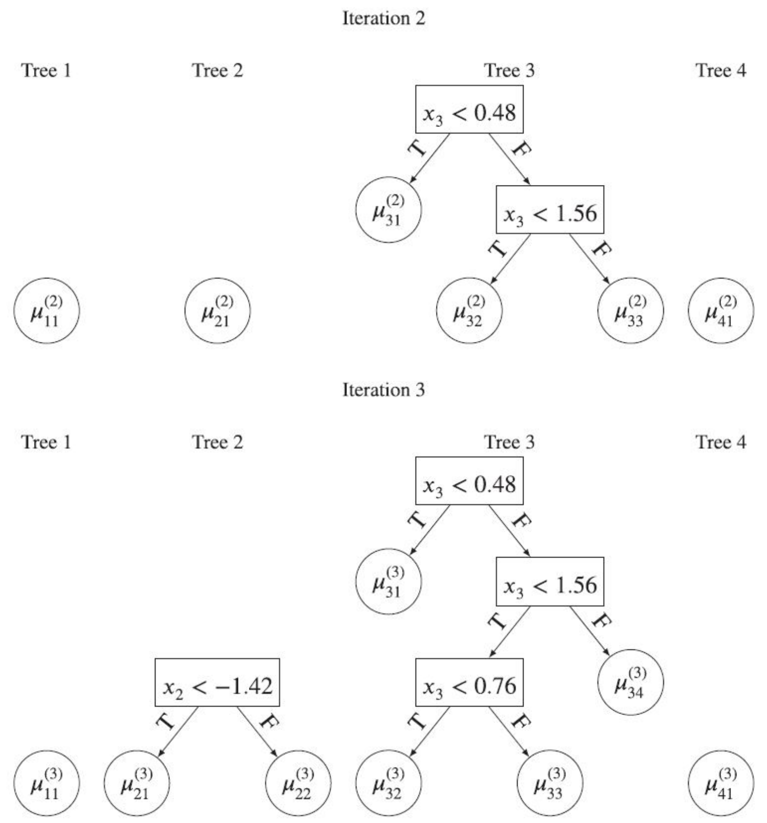
8.2.4 Bayesian Additive Regression Trees (BART)

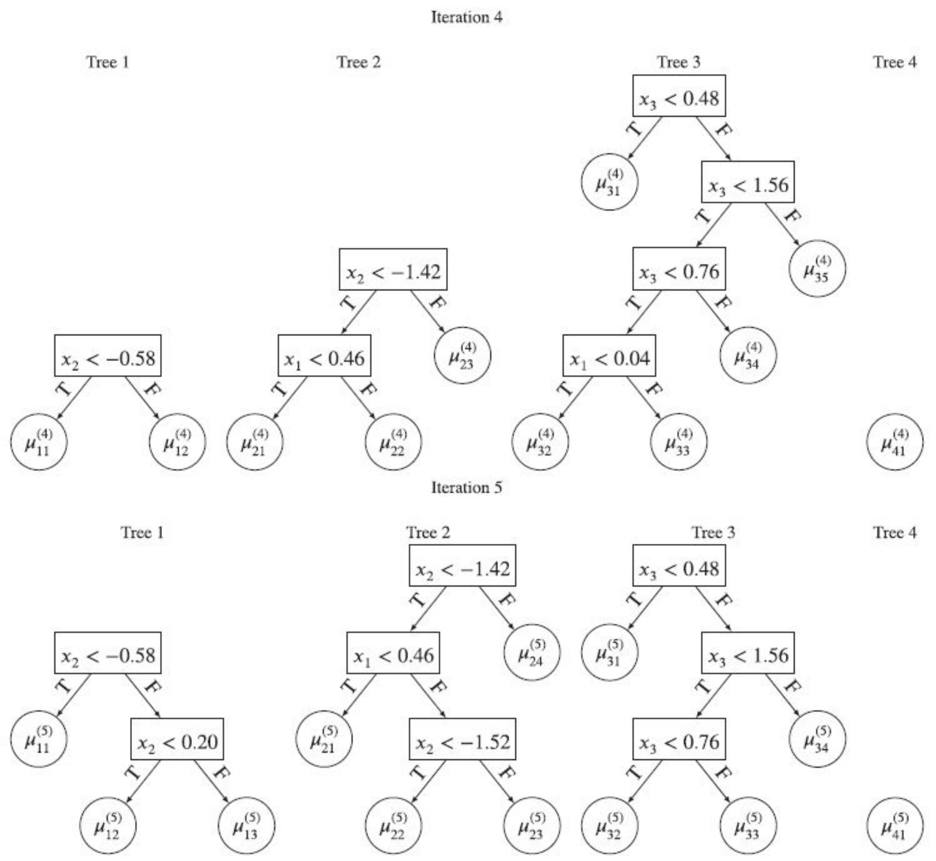
Figures from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6800811/



8.2.4 Bayesian Additive Regression Trees (BART)

Figures from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6800811/





8.2.4 Bayesian Additive Regression Trees (BART)

In total there will be a total of B tree models

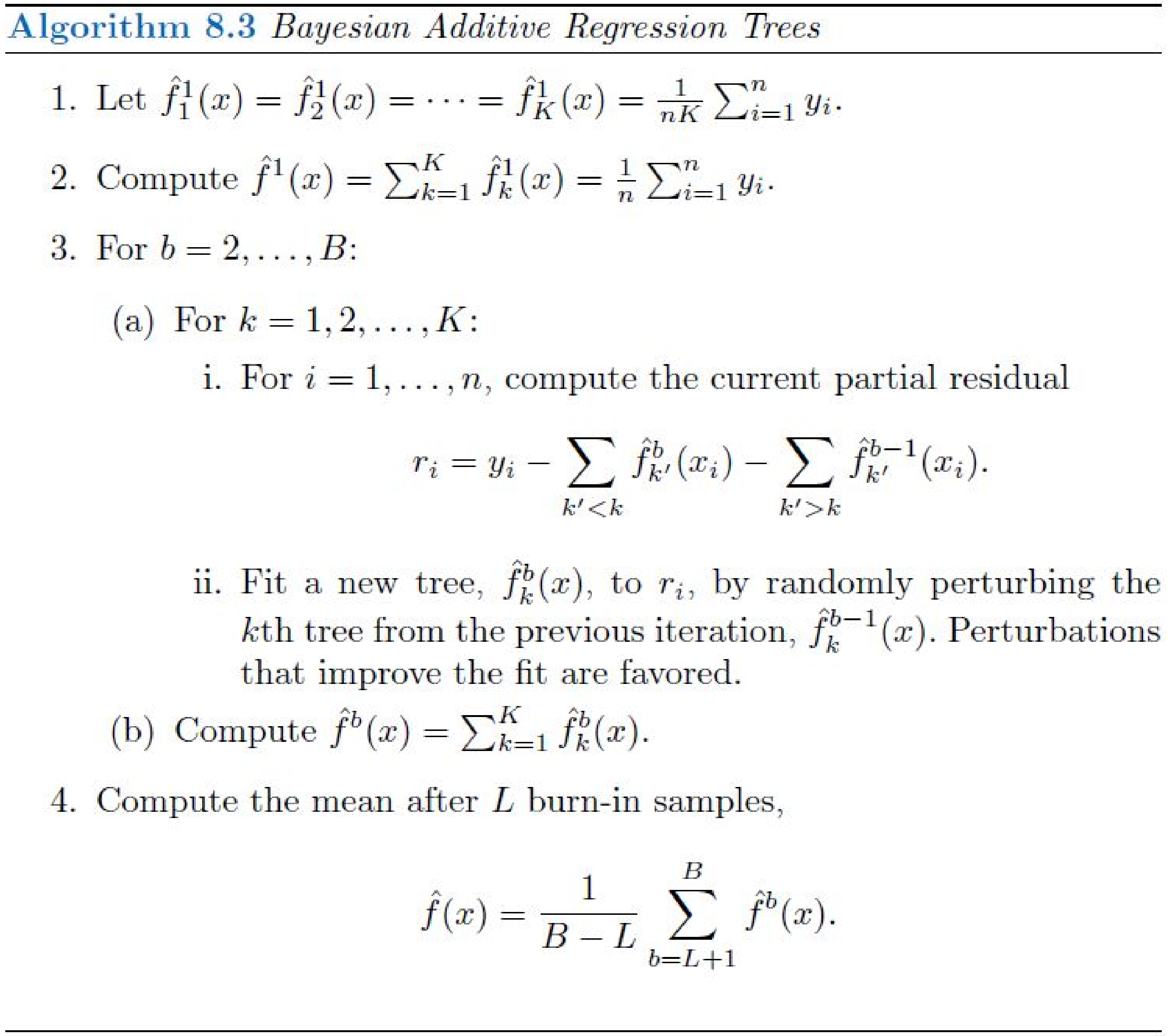
$$\hat{f}^1(x), \ldots, \hat{f}^B(x)$$

However, the first few models usually gives a poor prediction. Hence it is better to **get rid** of the first few L models.

To obtain a single prediction, the final tree model involves taking the **average** predicted value of the **remaining** $B - L$ tree models.

$$\hat{f}(x) = \frac{1}{B - L} \sum_{b=L+1}^{B} \hat{f}^b(x)$$

8.2.4 Bayesian Additive Regression Trees (BART)



Algorithm 8.3 *Bayesian Additive Regression Trees*

```
1. Let $\hat{f}_1^1(x) = \hat{f}_2^1(x) = \cdots = \hat{f}_K^1(x) = \frac{1}{nK}\sum_{i=1}^n y_i$.
2. Compute $\hat{f}^1(x) = \sum_{k=1}^K \hat{f}_k^1(x) = \frac{1}{n}\sum_{i=1}^n y_i$.
3. For $b = 2, \ldots, B$:
   (a) For $k = 1, 2, \ldots, K$:
       i. For $i = 1, \ldots, n$, compute the current partial residual
          $$r_i = y_i - \sum_{k'<k} \hat{f}_{k'}^b(x_i) - \sum_{k'>k} \hat{f}_{k'}^{b-1}(x_i).$$
       ii. Fit a new tree, $\hat{f}_k^b(x)$, to $r_i$, by randomly perturbing the
           kth tree from the previous iteration, $\hat{f}_k^{b-1}(x)$. Perturbations
           that improve the fit are favored.
   (b) Compute $\hat{f}^b(x) = \sum_{k=1}^K \hat{f}_k^b(x)$.
4. Compute the mean after $L$ burn-in samples,
   $$\hat{f}(x) = \frac{1}{B-L}\sum_{b=L+1}^{B} \hat{f}^b(x).$$
```