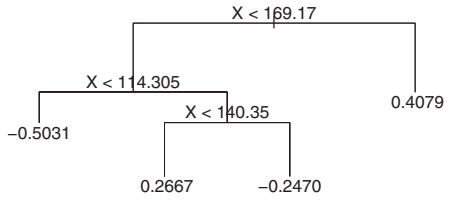
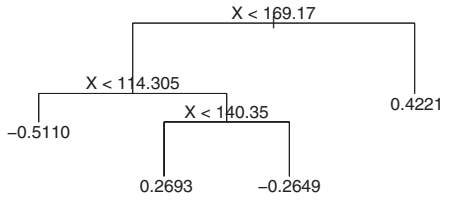
for the ith observation, $i = 1, ..., n$.

- Rather than fitting a fresh tree to this partial residual, BART randomly chooses a perturbation to the tree from the previous iteration (f_k^{b-1}) from a set of possible perturbations, favoring ones that improve the fit to the partial residual. There are two components to this perturbation:
  1. We may change the structure of the tree by adding or pruning branches.
  2. We may change the prediction in each terminal node of the tree.

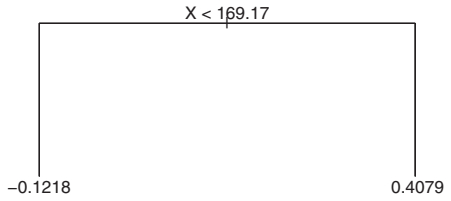
(a): $\hat{f}_k^{b-1}(X)$



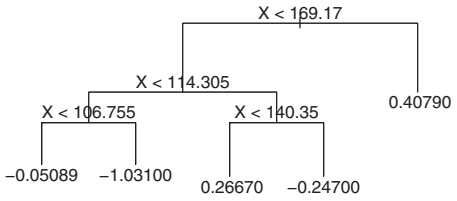
(b): Possibility #1 for $\hat{f}_k^b(X)$



(c): Possibility #2 for $\hat{f}_k^b(X)$



(d): Possibility #3 for $\hat{f}_k^b(X)$



A schematic of perturbed trees from the BART algorithm. (a): The kth 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 $\hat{f}_k^{b-1}(X)$.

- The output of BART is a collection of prediction models,

$$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x), \quad \text{for} \quad b = 1, ..., B.$$

- We typically throw away the first few of these prediction models, since models obtained in the earlier iterations — known as the burn-in period — tend not to provide very good results.

- We can let L denote the number of burn-in iterations; for instance, we might take $L = 200$. Then, to obtain a single prediction, we simply take the average after the burn-in iterations

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

Bayesian Additive Regression Trees Algorithm

1. Let $\hat{f}_1^1(x) = \hat{f}_2^1(x) = \ldots = \hat{f}_K^1(x) = \frac{1}{nK} \sum_{i=1}^{n} y_i$.
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

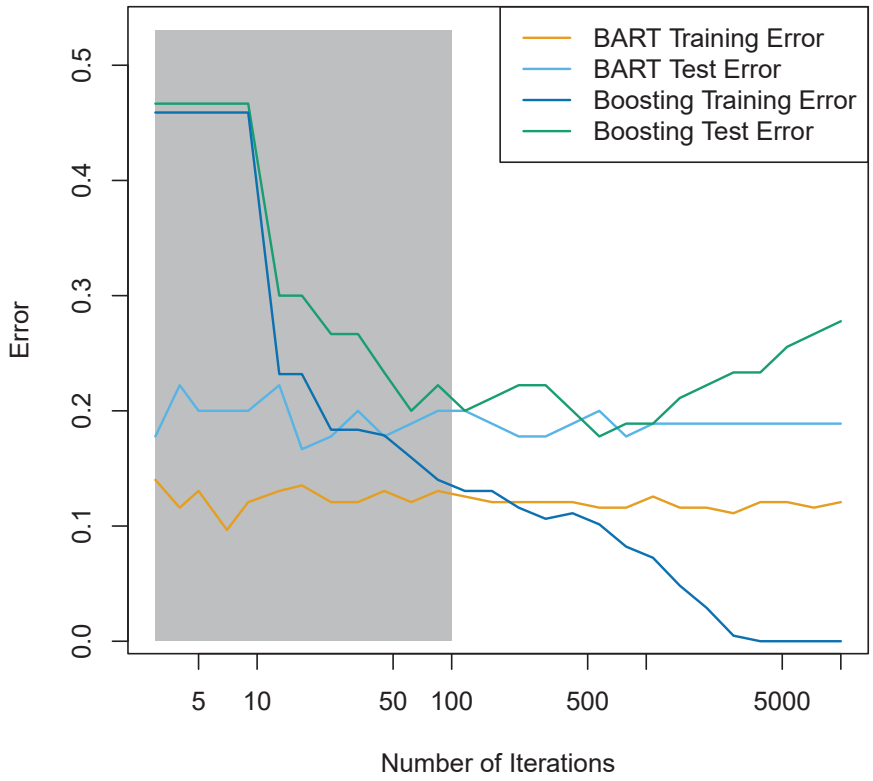
$$r_i = y_i - \sum_{k'<k} \hat{f}_{k'}^b(x_i) - \sum_{k'>k} \hat{f}_{k'}^{b-1}(x_i).$$

      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,

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



BART and boosting results for the Heart data. Both training and test errors are displayed. After a burn-in period of 100 iterations (shown in gray), the error rates for BART settle down. Boosting begins to overfit after a few hundred iterations.

Note: When we apply BART, we must select the number of trees K, the number of iterations B, and the number of burn-in iterations L. We typically choose large values for B and K, and a moderate value for L: for instance, $K = 200$, $B = 1,000$, and $L = 100$ is a reasonable choice.

Performing BART in R

Here we use the BART package, and within it the gbart() function, to fit a Bayesian additive regression tree model to the Hitters data set. The gbart() function is designed for quantitative outcome variables. For binary outcomes, lbart() and pbart() are available.

To run the gbart() function, we must first create matrices of predictors for the training and test data. We run BART with default settings.

```
> library(BART)
> names(Hitters)
 [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
 [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
[13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
[19] "Salary"    "NewLeague"
> x <- Hitters[, -19]
> names(x)
 [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
 [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
[13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
[19] "NewLeague"
> y <- Log_Salary
> xtrain <- x[train, ]
> ytrain <- y[train]
> xtest <- x[-train, ]
> ytest <- y[-train]
> set.seed(1)
> bartfit <- gbart(xtrain, ytrain, x.test = xtest)
*****Calling gbart: type=1
*****Data:
data:n,p,np: 132, 22, 131
y1,yn: 1.158454, -1.028171
x1,x[n*p]: 475.000000, 0.000000
xp1,xp[np*p]: 315.000000, 0.000000
*****Number of Trees: 200
*****Number of Cut Points: 100 ... 1
*****burn,nd,thin: 100,1000,1
*****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.0634347,3,0.0876564,5.94815
*****sigma: 0.670821
```

```
*****w (weights): 1.000000 ... 1.000000
*****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,22,0
*****printevery: 100

MCMC
done 0 (out of 1100)
done 100 (out of 1100)
done 200 (out of 1100)
done 300 (out of 1100)
done 400 (out of 1100)
done 500 (out of 1100)
done 600 (out of 1100)
done 700 (out of 1100)
done 800 (out of 1100)
done 900 (out of 1100)
done 1000 (out of 1100)
time: 2s
trcnt,tecnt: 1000,1000
```

Next we compute the test error.

```
> names(bartfit)
 [1] "sigma"           "yhat.train"      "yhat.test"       "varcount"
 [5] "varprob"         "treedraws"       "proc.time"       "hostname"
 [9] "yhat.train.mean" "sigma.mean"      "LPML"            "yhat.test.mean"
[13] "ndpost"          "offset"          "varcount.mean"   "varprob.mean"
[17] "rm.const"
> yhat_bart <- bartfit$yhat.test.mean
> mean((ytest - yhat_bart)^2)
[1] 0.1944177
```

Summary

- Decision trees are simple and interpretable models for regression and classification. However, they are often not competitive with other methods in terms of prediction accuracy.
- Bagging, random forests, boosting, and BART are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- Random forests, boosting, and BART are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Support Vector Machines

The *support vector machine* (SVM) is a generalization of the *maximal margin classifier*, a simple and intuitive classifier, and its extension, the *support vector classifier*.

Maximal Margin Classifier

Here we try to find a hyperplane that separates the feature space into two classes.

What Is a Hyperplane?

- In a p-dimensional space, a hyperplane is a flat affine subspace of dimension $p-1$. The word affine indicates that the subspace need not pass through the origin.
- In general, the equation for a hyperplane in a p-dimensional space has the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0 \qquad (*)$$

for parameters $\beta_0, \beta_1, \ldots, \beta_p$.

when we say that (*) defines the hyperplane, we mean that if a point $X = (X_1, \cdots, X_p)^T$ in p-dimensional space (i.e. a vector of length p) satisfies (*), then X lies on the hyperplane.

- For instance, in two dimensions, a hyperplane is defined by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

for parameters β_0, β_1, and β_2. This is simply the equation of a line.

In three dimension, a hyperplane is a flat two-dimensional subspace, that is, a plane.

In $p > 3$ dimension, it can be hard to visualize a hyperplane.

- Now, suppose that X does not satisfy the hypeplane; rather,

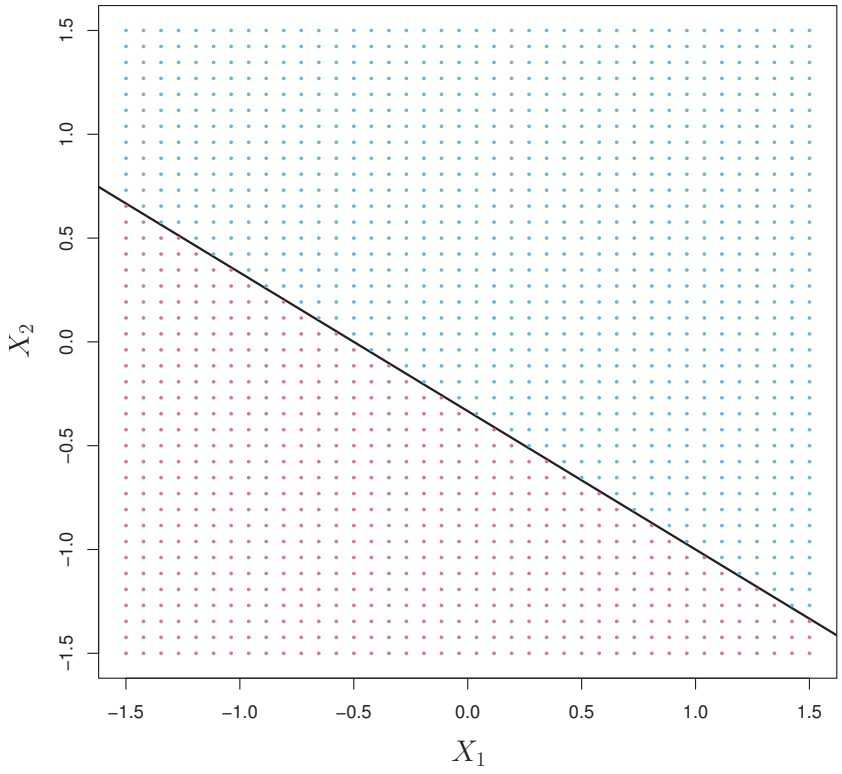
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0$$

  Then this tells us that X lies to one side of the hyperplane. On the other hand, if

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0$$

  then X lies on the other side of the hyperplane.

- So we can think of the hyperplane as dividing p-dimensional space into two halves.



The hyperplane $1 + 2X_1 + 3X_2 = 0$ is shown. The blue region is the set of points for which $1 + 2X_1 + 3X_2 > 0$, and the purple region is the set of points for which $1 + 2X_1 + 3X_2 < 0$.

Classification Using a Separating Hyperplane

- Now suppose that we have a $n \times p$ data matrix $\mathbf{X}$ that consists of n training observations in p-dimensional space,

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix},$$

  and that these observations fall into two classes—that is, $y_1, \ldots, y_n \in \{-1, 1\}$ where -1 represents one class and 1 the other class.

- We also have a test observation, a p-vector of observed features $x^* = (x_1^* \ldots x_p^*)^T$.