BIOST 546: Machine Learning for Biomedical Big Data

Ali Shojaie

Lecture 6: Tree-Based Methods for Regression and Classification
Spring 2017

Recap

- Classification in high dimensions:
  - Logistic regression, and penalized logistic regression
  - LDA & QDA
  - KNN classifier
  - SVM
- Batch effects and pitfalls of classification

Today's Class

- Tree-based methods for classification and regression
    - classification and regression trees
    - a digression: the bootstrap
    - bagging and boosting
    - random forests

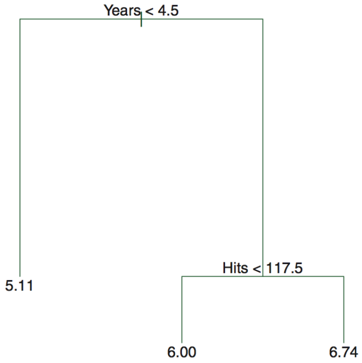
Tree-Based Methods

The main steps of tree-based methods are as follows:

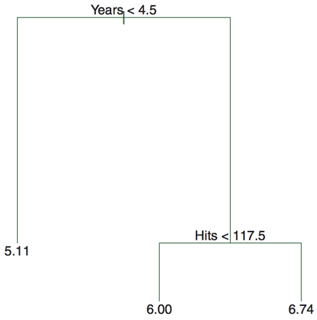
- *stratify* (or segment) the predictor space into small and simple regions
- for each observation, determine which segment it belongs to; the stratification of space can be captured by a tree (hence the name)
- predict the outcome by the mean (regression) or mode (classification) of outcomes in that segment

Toy Example: predicting salaries of baseball players I

- Outcome: Salary (in $\$1000$ and $\log_e$ scale) of baseball players
- Two predictors: years of experience in MLB (`Years`) and number of hits made in the previous year (`Hits`)



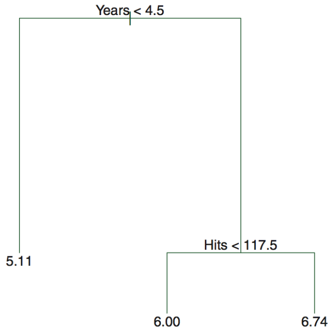
Toy Example: predicting salaries of baseball players II

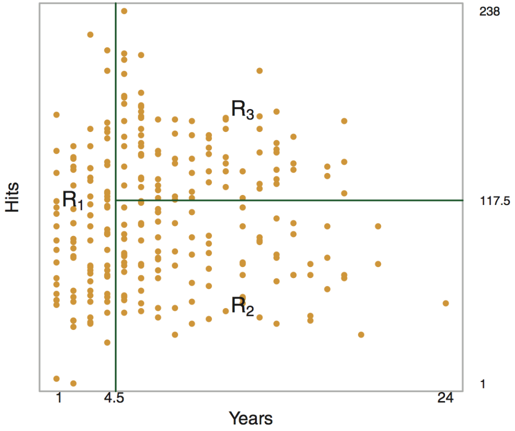


- The top split assigns observations having `Years` < 4.5 to the left branch
  - The predicted salary for players with < 4.5 years of experience is $\$1,000 \times e^{5.11}$, which is the mean salary for such players in the training data
- Players with > 4.5 `Years` of experience are further divided into 2 groups based on their number of `Hits` in the previous year:
  - For those with < 117.5 `Hits`, the predicted salary is $\$1,000 \times e^{6.00}$
  - For those with > 117.5 `Hits`, the predicted salary is $\$1,000 \times e^{6.74}$

Toy Example: predicting salaries of baseball players III

The tree divides the predictor space into 3 regions





$$R_1 = \{X \mid Years < 4.5\}$$
$$R_2 = \{X \mid Years \geq 4.5, Hits < 117.5\}$$
$$R_3 = \{X \mid Years \geq 4.5, Hits \geq 117.5\}$$

Decision Trees: the general idea

- The *regression* tree in the above example is likely an over-simplification of the true relationship between `Hits`, `Years`, and `Salary`.
- However, it is very easy to interpret, and has a nice graphical representation: you can easily explain it to a non-statistician, and don't need a computer (or even a calculator) to get an estimate!

Decision Trees: the general idea

- The general steps for building a regression (or classification) tree is quite simple:
  1) Divide the predictor space, i.e. 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) Use the mean (regression) or mode (classification) of the response values for the training observations in region R_j as the predicted response for that region.

Decision Trees: the general idea

- The general steps for building a regression (or classification) tree is quite simple:
  1) Divide the predictor space, i.e. 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) Use the mean (regression) or mode (classification) of the response values for the training observations in region R_j as the predicted response for that region.
- The main question: how should we construct the regions $R_1, \ldots, R_J$, and how many regions should we use?

Partitioning the Predictor Space

For now assume J is known. How to construct the regions $R_1, \ldots, R_J$?

Partitioning the Predictor Space

For now assume J is known. How to construct the regions $R_1, \ldots, R_J$?

- Divide the predictor space into high-dimensional rectangles, or boxes (in theory any shape can be used, but this is simpler)

Partitioning the Predictor Space

For now assume J is known. How to construct the regions $R_1, \ldots, R_J$?

- Divide the predictor space into high-dimensional rectangles, or boxes (in theory any shape can be used, but this is simpler)
- Find boxes $R_1, \ldots, R_J$ that minimize the RSS

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

Partitioning the Predictor Space

For now assume J is known. How to construct the regions $R_1, \ldots, R_J$?

- Divide the predictor space into high-dimensional rectangles, or boxes (in theory any shape can be used, but this is simpler)
- Find boxes $R_1, \ldots, R_J$ that minimize the RSS

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

- Unfortunately, it is not possible to consider every possible partition of the space into J boxes
- We consider instead a *top-down*, *greedy* approach called recursive binary splitting

Partitioning the Predictor Space

For now assume J is known. How to construct the regions $R_1, \ldots, R_J$?

- Divide the predictor space into high-dimensional rectangles, or boxes (in theory any shape can be used, but this is simpler)
- Find boxes $R_1, \ldots, R_J$ that minimize the RSS

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

- Unfortunately, it is not possible to consider every possible partition of the space into J boxes
- We consider instead a *top-down*, *greedy* approach called recursive binary splitting
    - top-down: similar to hierarchical clustering
    - greedy: the best split is determined at that particular step, instead of considering the best global step (which is computationally difficult)

Partitioning the Predictor Space

For now assume J is known. How to construct the regions $R_1, \ldots, R_J$?

- Divide the predictor space into high-dimensional rectangles, or boxes (in theory any shape can be used, but this is simpler)
- Find boxes $R_1, \ldots, R_J$ that minimize the RSS

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

- Unfortunately, it is not possible to consider every possible partition of the space into J boxes
- We consider instead a *top-down*, *greedy* approach called recursive binary splitting
  - top-down: similar to hierarchical clustering
  - greedy: the best split is determined at that particular step, instead of considering the best global step (which is computationally difficult)

The above formulation is for a *regression* tree; for *classification*, we can use the 0-1 loss. We discuss the classification case in a few slides.

Recursive Binary Splitting

- Select a predictor X_j and a cut point t such that splitting the predictor space into regions

$$R_1(j,t) = \{X \mid X_j < t\}, \qquad R_2(j,t) = \{X \mid X_j \geq t\}$$

gives the largest decrease in RSS.

Recursive Binary Splitting

- Select a predictor X_j and a cut point t such that splitting the predictor space into regions

$$R_1(j,t) = \{X \mid X_j < t\}, \qquad R_2(j,t) = \{X \mid X_j \geq t\}$$

gives the largest decrease in RSS.

  - We consider *all predictors* and *all cut points* for each predictor!
  - We find the value of j and t that minimizes

$$\sum_{i:x_i \in R_1(j,t)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,t)} (y_i - \hat{y}_{R_2})^2$$

  - Finding *optimal* j and s can actually be done very quickly (as long as p is not too large)

Recursive Binary Splitting

- Select a predictor X_j and a cut point t such that splitting the predictor space into regions

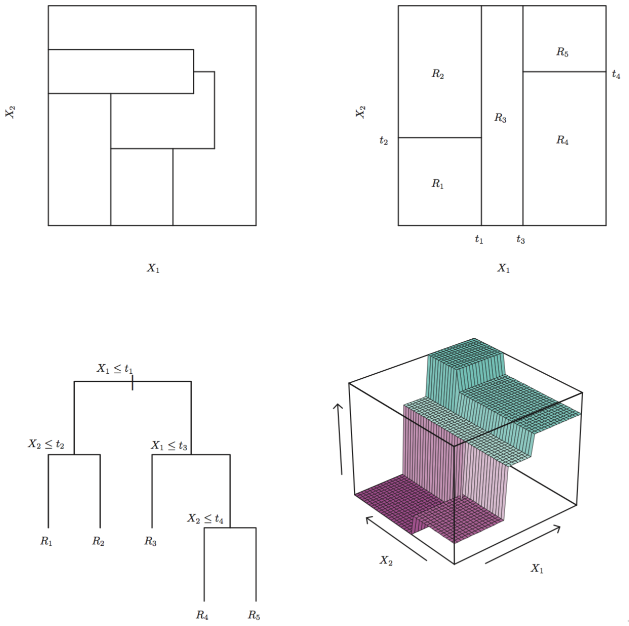
$$R_1(j,t) = \{X \mid X_j < t\}, \qquad R_2(j,t) = \{X \mid X_j \geq t\}$$

  gives the largest decrease in RSS.
  - We consider *all predictors* and *all cut points* for each predictor!
  - We find the value of j and t that minimizes

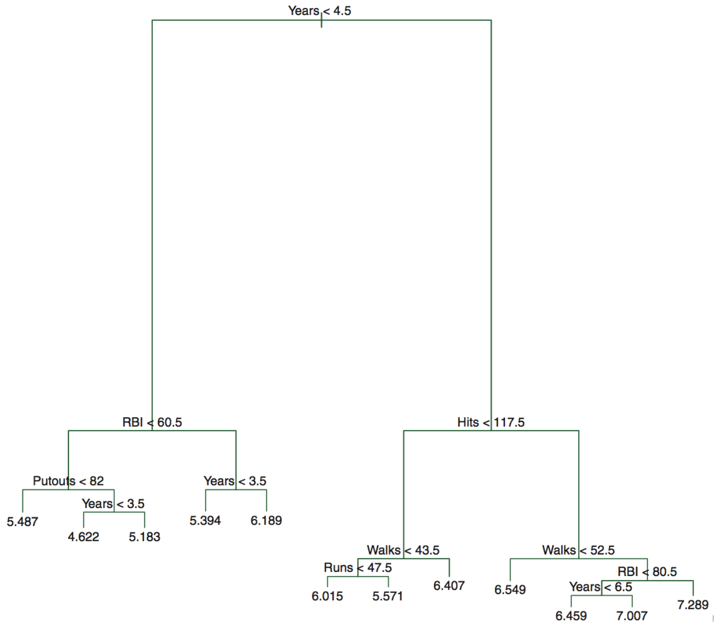
$$\sum_{i: x_i \in R_1(j,t)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,t)} (y_i - \hat{y}_{R_2})^2$$

  - Finding *optimal* j and s can actually be done very quickly (as long as p is not too large)
- Repeat the above process, splitting a subspace (rather than the whole space) in each step
  - Previously used variables can be considered again

Recursive Binary Splitting



Full Tree for the `Hitters` Data Set



Tree Pruning

- The complexity of the regression tree is determined by the number of regions J

Tree Pruning

- The complexity of the regression tree is determined by the number of regions J
- A tree with large J may work well in training data, but would be very bad on test data
- A smaller tree with fewer splits might have lower variance and better interpretation at the cost of a little bias

Tree Pruning

- The complexity of the regression tree is determined by the number of regions J
- A tree with large J may work well in training data, but would be very bad on test data
- A smaller tree with fewer splits might have lower variance and better interpretation at the cost of a little bias
- One option is to consider a split only if the drop in RSS is larger than some (high) threshold

Tree Pruning

- The complexity of the regression tree is determined by the number of regions J
- A tree with large J may work well in training data, but would be very bad on test data
- A smaller tree with fewer splits might have lower variance and better interpretation at the cost of a little bias
- One option is to consider a split only if the drop in RSS is larger than some (high) threshold
- However, this may not be a good strategy since a so-so split at step j may be followed by a great one at step $j+1$

Tree Pruning

- The complexity of the regression tree is determined by the number of regions J
- A tree with large J may work well in training data, but would be very bad on test data
- A smaller tree with fewer splits might have lower variance and better interpretation at the cost of a little bias
- One option is to consider a split only if the drop in RSS is larger than some (high) threshold
- However, this may not be a good strategy since a so-so split at step j may be followed by a great one at step $j+1$
- Instead, we first grow a large tree, e.g. until no region has > 5 observations, and then prune it to obtain a subtree

Tree Pruning

- Our goal is to select a subtree that leads to the lowest test error

Tree Pruning

- Our goal is to select a subtree that leads to the lowest test error
- We can use CV, however, it will be too computationally expensive to estimate the CV error for every possible subtree!

Tree Pruning

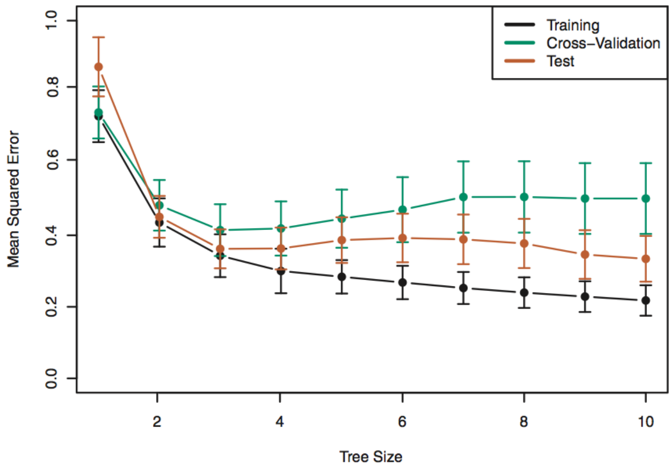
- Our goal is to select a subtree that leads to the lowest test error
- We can use CV, however, it will be too computationally expensive to estimate the CV error for every possible subtree!
- Instead we use a strategy called cost complexity pruning a.k.a. weakest link pruning

Tree Pruning

- Our goal is to select a subtree that leads to the lowest test error
- We can use CV, however, it will be too computationally expensive to estimate the CV error for every possible subtree!
- Instead we use a strategy called cost complexity pruning a.k.a. weakest link pruning
  - Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α
  - For each value of α, there exists a subtree $T \subset T_0$ such that

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

    is as small as possible ($|T|$ is the number of leaves of the tree)
  - α controls the tradeoff between complexity and fit (sound familiar?)
  - We can select α using validation set or CV approach


Training
Cross–Validation
Test
Mean Squared Error
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Tree Size

Putting it all together...

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. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

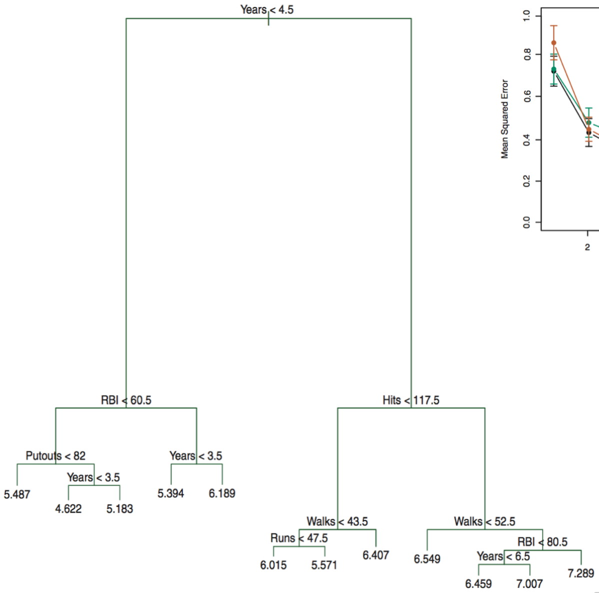
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

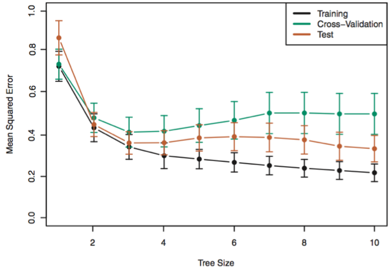
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

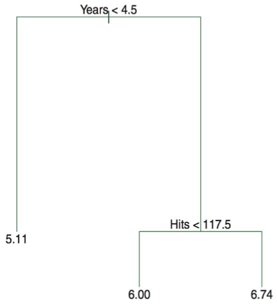
   Average the results for each value of α, and pick α to minimize the average error.

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

Putting it all together...







Classification Trees

Everything is the same as regression trees, except that

- we have a qualitative response, so use majority voting (mode) in each region, instead of mean

Classification Trees

Everything is the same as regression trees, except that

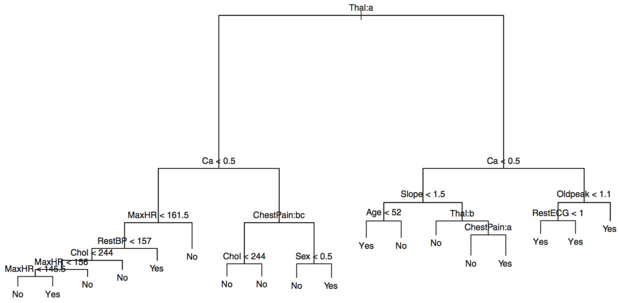
- we have a qualitative response, so use majority voting (mode) in each region, instead of mean
- the misclassification error rate is the natural choice to grow the tree, however, it turns out that the following work better
  - the Gini index
  $$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$
  - the cross-entropy
  $$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$
  - In the above, $\hat{p}_{mk}$ is the proportion of training observations in the mth region from the kth class

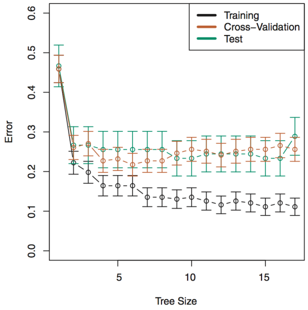
Classification Trees

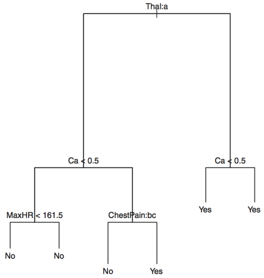
Everything is the same as regression trees, except that

- we have a qualitative response, so use majority voting (mode) in each region, instead of mean
- the misclassification error rate is the natural choice to grow the tree, however, it turns out that the following work better
  - the Gini index
    $$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$
  - the cross-entropy
    $$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$
  - In the above, $\hat{p}_{mk}$ is the proportion of training observations in the mth region from the kth class
- Both of these measure the purity of observations in each region, and have small values if all $\hat{p}_{mk}$'s are close to 0 or 1.

Classification Tree for the `Heart` Data

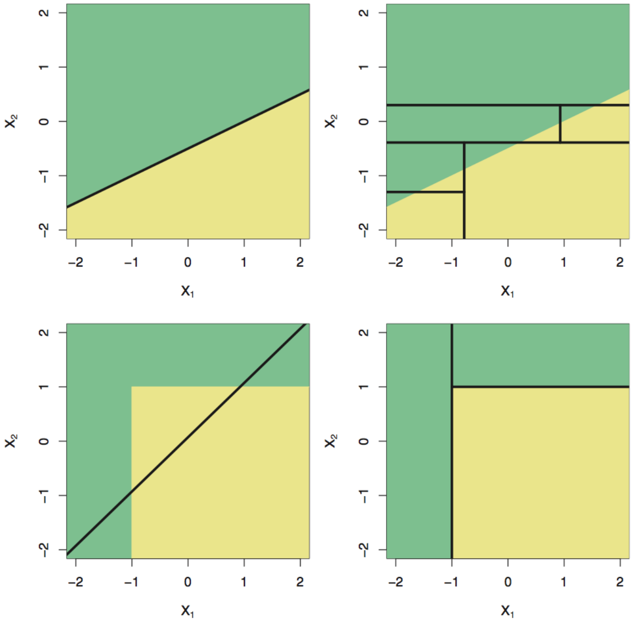






Trees vs Linear Models

Trees vs Linear Models



Trees: Pros and Cons

- Pros:
- very easy to interpret and explain to others
- can be easily displayed graphically (especially if they are small)
- can easily handle qualitative predictors without the need to create dummy variables
- Cons:
    - Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification methods we've discussed so far.
- By aggregating many decision trees, the predictive performance of trees can be substantially improved.

The Bootstrap

- Bagging, random forests, and boosting use trees as building blocks to construct more powerful prediction models (unfortunately, this comes at the cost of ease of interpretation!)
- All of these methods build many trees to improve the performance of tree-based methods. This is motivated by the fact that usual trees have high variance
- Both bagging and random forests build trees by sampling from the original data using the bootstrap
- Bootstrap is a powerful and general procedure that can be used to assess the variability of estimators. Here, we see that bootstrap can also be used to improve the performance of estimators.
- Before discussing bagging and random forests (and boosting), we talk a bit about the bootstrap

Confidence Intervals: a reminder I

For testing, want to know if $\mu = 0$
If we reject, would like to know a region wherein μ is likely.

R makes this easy:

```
> t.test(x)
One Sample t-test
data:  x
t = 5.7446, df = 9, p-value = 0.0002782
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 3.334149 7.665851
sample estimates:
mean of x
      5.5
```

Confidence Intervals: a reminder II

What is actually happening?

$$z = \frac{\bar{x} - \mu}{\sigma/\sqrt{n}} \dot{\sim} N(0,1)$$

So $z \in (-1.96, 1.96)$ with 0.95 probability, thus

$$\mu \in \left(\bar{x} - 1.96\,\sigma/\sqrt{n},\ \bar{x} + 1.96\,\sigma/\sqrt{n}\right)$$

with 0.95 probability.

Confidence Intervals: a reminder III

What is actually happening?

$$z = \frac{\bar{x} - \mu}{\sigma/\sqrt{n}} \mathrel{\dot\sim} N(0,1)$$

So $z \in (-1.96, 1.96)$ with 0.95 probability, thus

$$\mu \in \left(\bar{x} - 1.96\,\sigma/\sqrt{n},\ \bar{x} + 1.96\,\sigma/\sqrt{n}\right)$$

with 0.95 probability.

When can we do this?

- The data is normally distributed (which rarely happens in practice!)
- The sample size n is large enough that this holds asymptotically (CLT)

What if we don't know the asymptotic distribution? I

- Our data comes from $\boldsymbol{F}$
- We are interested in variance (SE) of a summary Θ (e.g. mean) of $\boldsymbol{F}$
- We don't know $\boldsymbol{F}$, nor do we know the (asymptotic) distribution of $\Theta(\boldsymbol{F})$!

What if we don't know the asymptotic distribution? II

- Our data comes from $\boldsymbol{F}$
- We are interested in variance (SE) of a summary Θ (e.g. mean) of $\boldsymbol{F}$
- We don't know $\boldsymbol{F}$, nor do we know the (asymptotic) distribution of $\Theta(\boldsymbol{F})$!

For a moment, pretend we had access to unlimited data (as in simulation)!

- We can *sample* many data sets of size n (say $B = 10,000$ of them)
- From each data set, we can calculate one $\hat{\Theta}$
- We can then look at the distribution of $\hat{\Theta}$'s across B samples, and calculate its variance

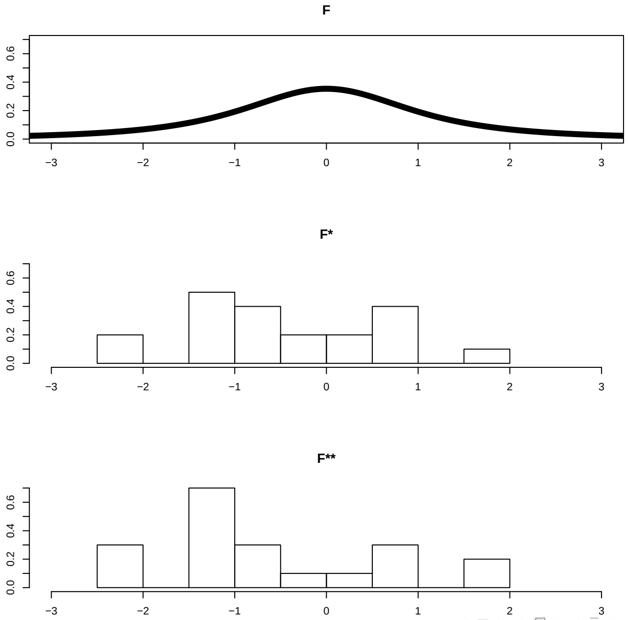
Unfortunately, in practice we only have a single sample of size n!

What if we don't know the asymptotic distribution? III

Bootstrap tries to *mimic* the previous (impossible) setting

- Our data comes from $\boldsymbol{F}$
- We are interested in Θ of $\boldsymbol{F}$
- We get a sample from $\boldsymbol{F}$, giving us an empirical distribution $\boldsymbol{F}^*$
- We could sample from $\boldsymbol{F}^*$ giving us $\boldsymbol{F}^{**}$

What if we don't know the asymptotic distribution? IV



A simple (non-omics!) example from ISL I

Minimizing risk of investment

- We want to invest a fixed amount of money in two stocks with returns X and Y
- This is equivalent to investing a fraction α of our money in X, and the remaining $1-\alpha$ in Y
- Investment risk is measured by variance, so we want to minimize $\mathrm{Var}(\alpha X + (1-\alpha)Y)$
- To do this, we need

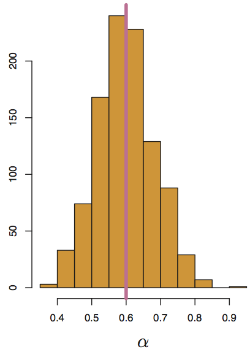
$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

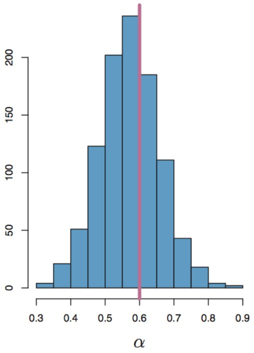
- How can we measure the variability of the optimal α (e.g. if we want to test whether $\alpha = 0.5$)?

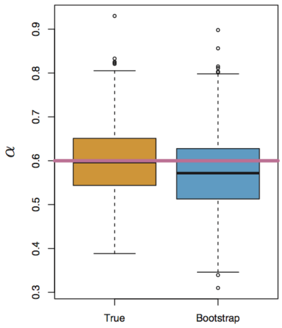
A simple (non-omics!) example from ISL II

Minimizing risk of investment

First let's see how bootstrap compares to simulating data from the actual distribution...



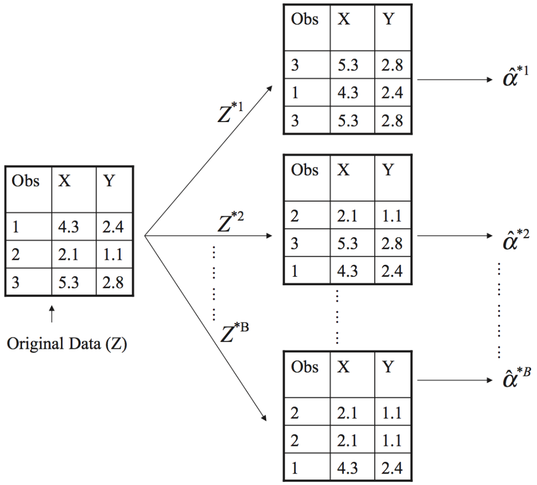




A simple (non-omics!) example from ISL III

Minimizing risk of investment

A closer look at what is going on...



Bootstrap: The Main Idea

Assume that $\boldsymbol{F} \leftrightarrow \boldsymbol{F}^*$, is mirrored by $\boldsymbol{F}^* \leftrightarrow \boldsymbol{F}^{**}$.

We find an interval $I = [L, U]$ with

$$P\left[\Theta\left(\boldsymbol{F}^*\right) - \Theta\left(\boldsymbol{F}^{**}\right) \in I\right] = 0.95$$

and pretend

$$P\left[\Theta\left(\boldsymbol{F}\right) - \Theta\left(\boldsymbol{F}^*\right) \in I\right] \approx 0.95$$

This gives us

$$\Theta\left(\boldsymbol{F}\right) \in \left[\Theta\left(\boldsymbol{F}^*\right) + L, \Theta\left(\boldsymbol{F}^*\right) + U\right]$$

with ~ 0.95 probability

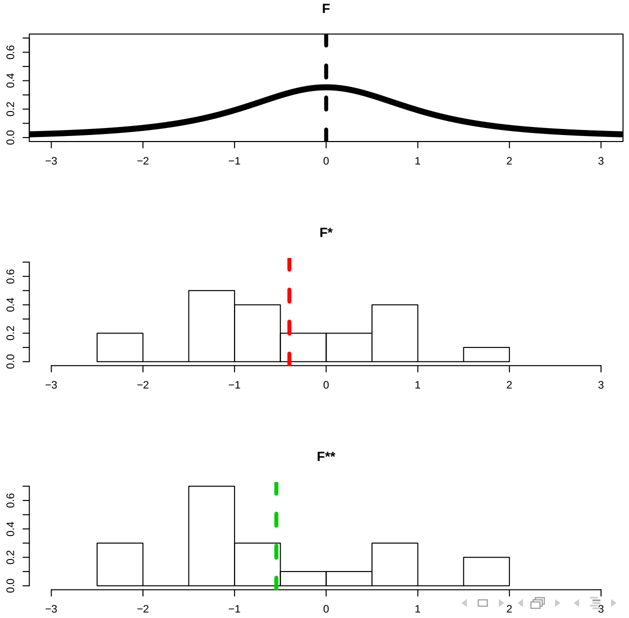
How should we sample from $\boldsymbol{F}^*$?

The Bootstrap in Pictures

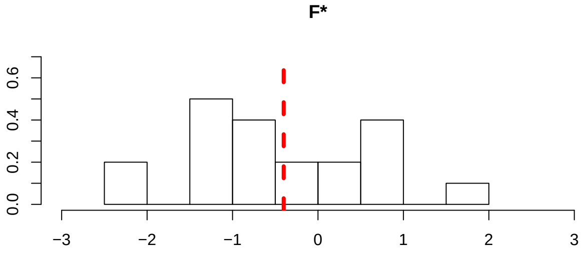
A confidence interval for the mean!

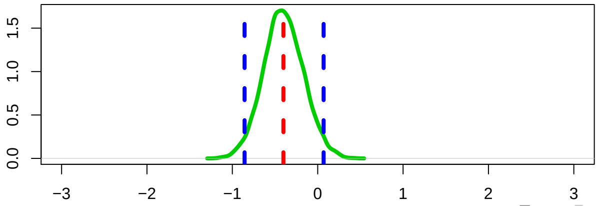
- $\Theta(F) =$ Population Mean
- $\Theta(F^*) =$ Sample Mean
- $\Theta(F^{**}) =$ resampled Mean

The Bootstrap in Pictures



The Bootstrap in Pictures





The Bootstrap in Practice

```
1. Calculate $\Theta^*$ on your sample $X$
2. for $b = 1, \ldots, B$
    1. Resample with replacement from $X$ to get $X^b$
    2. Calculate $\Theta^{**b}$ on $X^b$
3. Find the $0.025$ and $0.975$ quantile of $\Theta^* - \Theta^{**b}$ for $L$ and $U$
4. Report $[\Theta^* + L, \Theta^* + U]$ as the CI.
```

The Bootstrap in R

The simple bootstrap is straightforward

```
calc_statistic <- function(x){
  return(mean(x))
}

mu_star <- mean(X)

mu_star_star <- replicate(10000, calc_statistic(sample(X, replace=TRUE)))

U <- quantile(mu_star - mu_star_star, 0.975)
L <- quantile(mu_star - mu_star_star, 0.025)

CI <- c(mu_star + L, mu_star + U)
```

Even Simpler Bootstrap

The Percentile Bootstrap

```
calc_statistic <- function(x){
  return(mean(x))
}

mu_star <- mean(X)

mu_star_star <- replicate(10000, calc_statistic(sample(X, replace=TRUE)))

U <- quantile(mu_star_star, 0.975)
L <- quantile(mu_star_star, 0.025)

CI <- c(L,U)
```

This can be a *disaster* for non-symmetric sampling distributions

Bootstrap: additional comments

Bootstrap: additional comments

- When does the bootstrap (provably) work?

Bootstrap: additional comments

- When does the bootstrap (provably) work?
  - When the statistic, suitably centered and scaled, has a reasonable asymptotic distribution.

Bootstrap: additional comments

- When does the bootstrap (provably) work?
  - When the statistic, suitably centered and scaled, has a reasonable asymptotic distribution.
- When does the bootstrap not work?

Bootstrap: additional comments

- When does the bootstrap (provably) work?
  - When the statistic, suitably centered and scaled, has a reasonable asymptotic distribution.
- When does the bootstrap not work?
  - When the statistic is weird (not a smooth function of the data), or when n is really small! (Bootstrap is an asymptotic procedure too!)

Bootstrap: additional comments

- When does the bootstrap (provably) work?
  - When the statistic, suitably centered and scaled, has a reasonable asymptotic distribution.
- When does the bootstrap not work?
  - When the statistic is weird (not a smooth function of the data), or when n is really small! (Bootstrap is an asymptotic procedure too!)
- Other thoughts

Bootstrap: additional comments

- When does the bootstrap (provably) work?
  - When the statistic, suitably centered and scaled, has a reasonable asymptotic distribution.
- When does the bootstrap not work?
  - When the statistic is weird (not a smooth function of the data), or when n is really small! (Bootstrap is an asymptotic procedure too!)
- Other thoughts
  - Using "fancier" bootstraps (ABC, BCA, "t", etc...) can lead to more correct CI (more correct than using asymptotic normality).
  - Bootstrap can also be used to estimate the test error, but this is not straightforward
  - We don't have to take a sample of size n, we can take a smaller sample of size m (especially when the original bootstrap does not work), but the sampling should always be with replacement

Bagging

- The usual decision trees (regression or classification) have high variance:
  - if we split the training data into two parts, we get very different trees
  - Methods with low variance (e.g. linear regression with small p) give similar models in this case
- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method

Bagging: The Main Idea

- We know that averaging reduces the variance:
  - Specifically, if we take the average of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, $\mathrm{Var}(\bar{Z}) = \sigma^2/n$

Bagging: The Main Idea

- We know that averaging reduces the variance:
  - Specifically, if we take the average of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, $\mathrm{Var}(\bar{Z}) = \sigma^2/n$
- So to reduce the variance of a learning procedure, we could take many (say B) training samples, build a separate prediction model from each, and average the resulting predictions

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

Bagging: The Main Idea

- We know that averaging reduces the variance:
  - Specifically, if we take the average of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, $\mathrm{Var}(\bar{Z}) = \sigma^2/n$
- So to reduce the variance of a learning procedure, we could take many (say B) training samples, build a separate prediction model from each, and average the resulting predictions

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

- As we discussed before, this is not possible in practice, so in bagging, we use bootstrap samples

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

Bagging: Some Remarks

- For classification trees, majority voting takes the place of averaging
- When using bagging with trees, each tree is grown deep and is not pruned.
- Thus, each individual tree has high variance, but low bias. Averaging these B trees reduces the variance.
- The value of B is not critical:
  - using a very large value of B will not lead to overfitting
  - as B increases, computation becomes more expensive
  - in practice $B = 100$ to $B = 1000$ works pretty well

Random Forests: The Motivation

- Suppose that there is one very strong predictor in the data set, along with a number of moderately strong ones
- Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split, and they all look somewhat similar
- This means that predictions from the bagged trees can be highly correlated
- Averaging many highly correlated predictors does not lead to as large of a reduction in variance as uncorrelated predictors
- In such a case, bagging may not give significance reduction in variance compared to a single tree

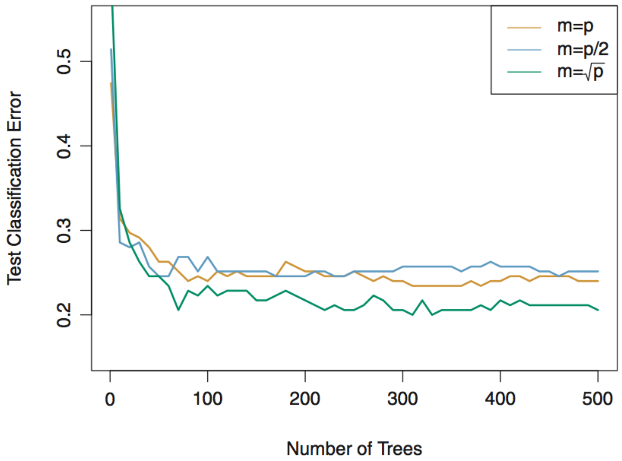
Random Forests: The Idea

- In random forests, only m predictors, chosen randomly, are available for the tree from each bootstrap sample
  - Often m chosen to be small relative to p, e.g. $m = \sqrt{p}$
  - Thus, each of the trees would not even have access to the majority of predictors
  - When $m = p$, we get the usual bagging

Random Forests: The Idea

- In random forests, only m predictors, chosen randomly, are available for the tree from each bootstrap sample
  - Often m chosen to be small relative to p, e.g. $m = \sqrt{p}$
  - Thus, each of the trees would not even have access to the majority of predictors
  - When $m = p$, we get the usual bagging
- The random selection of variables in random forests "decorrelates" the trees, and averaging over these decorrelated trees results in a large reduction in variance
- This also makes random forests computationally more attractive (compared to e.g. bagging)
- However, each of the trees will have higher bias in this case, so need to again consider the bias-variance tradeoff

Application to gene expression microarrays



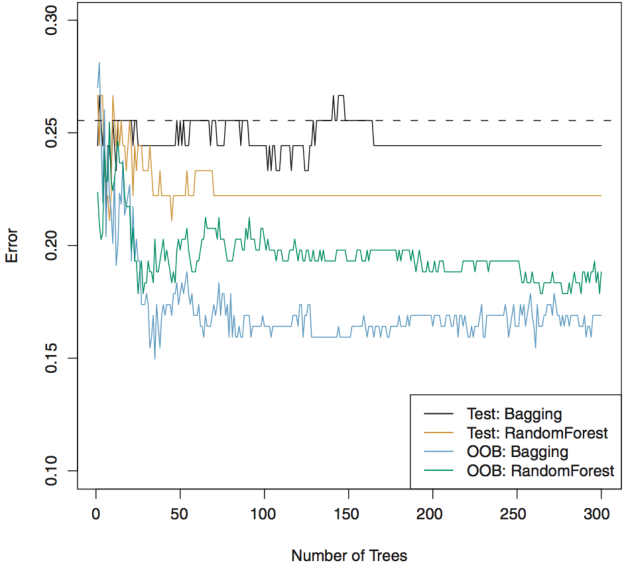
Out-of-Bag Error Estimation

- It turns out that we can easily estimate the test error in a *bagged model*
- In general, the chance of a single observation being included in a given bootstrap sample is about 2/3:

$$\begin{aligned} Pr\{\text{observation } i \in \text{bootstrap sample } b\} &= 1-(1-1/N)^N \\ &\approx 1-e^{-1} \\ &= 0.632. \end{aligned}$$

- The remaining one-third of the observations not used to fit a given bagged tree are called out-of-bag (OOB) observations
- These OOB observations can be used to estimate the test error

A Comparison



Interpreting The Results

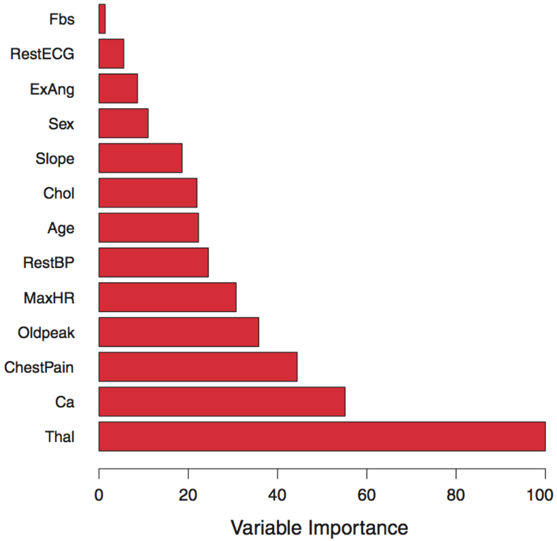
- Unfortunately, bagging and random forest result in models that are not easily interpretable:
  - it is unclear how each variable affects each of the tree models, and how we should interpret the final model

Interpreting The Results

- Unfortunately, bagging and random forest result in models that are not easily interpretable:
  - it is unclear how each variable affects each of the tree models, and how we should interpret the final model
- A very useful tool for gaining insight about individual variables is the variable importance plot
  - The plot shows the total amount of improvement in RSS/Gini Index/entropy resulting from splits over a given predictor, averaged over all B trees

Variable Importance Plot

For the `Heart` data



Boosting

- Boosting is another approach for improving the performance of tree-based methods

Boosting

- Boosting is another approach for improving the performance of tree-based methods
- Like bagging and random forests, boosting uses multiple trees. The main difference is that
  - bagging and random forests aggregate trees based on multiple copies of the data
  - boosting aggregates trees based on a modified version of the same data

Boosting

- Boosting is another approach for improving the performance of tree-based methods
- Like bagging and random forests, boosting uses multiple trees. The main difference is that
  - bagging and random forests aggregate trees based on multiple copies of the data
  - boosting aggregates trees based on a modified version of the same data
- Although we focused here on tree-based methods, bagging and boosting are general purpose methods that can improve the performance of any prediction method.

Boosting: Main Idea

- Boosting does not involve bootstrap sampling, and everything is based on a single data set

Boosting: Main Idea

- Boosting does not involve bootstrap sampling, and everything is based on a single data set
- In boosting, the trees are grown sequentially: each tree is grown using information from previously grown trees

Boosting: Main Idea

- Boosting does not involve bootstrap sampling, and everything is based on a single data set
- In boosting, the trees are grown sequentially: each tree is grown using information from previously grown trees
- Instead of fitting a single large decision tree, which can potentially overfit the data, in boosting we learn slowly
  - Each tree is small, with just a few terminal nodes
  - Given the current model, we fit a decision tree to the residuals from the previous model as the response
  - We then add this new decision tree into the fitted function and update the residuals

Boosting: Main Idea

- Boosting does not involve bootstrap sampling, and everything is based on a single data set
- In boosting, the trees are grown sequentially: each tree is grown using information from previously grown trees
- Instead of fitting a single large decision tree, which can potentially overfit the data, in boosting we learn slowly
    - Each tree is small, with just a few terminal nodes
    - Given the current model, we fit a decision tree to the residuals from the previous model as the response
    - We then add this new decision tree into the fitted function and update the residuals
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well

Boosting: The Algorithm

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,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Tuning Parameters

- Boosting has 3 tuning parameters:

Tuning Parameters

- Boosting has 3 tuning parameters:
  - The number of trees B: unlike bagging and random forests, boosting can overfit if B is too large, though this happens slowly. We use cross-validation to select B

Tuning Parameters

- Boosting has 3 tuning parameters:
  - The number of trees B: unlike bagging and random forests, boosting can overfit if B is too large, though this happens slowly. We use cross-validation to select B
  - The shrinkage parameter λ: a small positive number that controls the rate of learning (typically 0.01 or 0.001). λ slows down the learning process, allowing more and different shaped trees to attack the residuals. A very small λ may require a very large B to achieve good performance.

Tuning Parameters

- Boosting has 3 tuning parameters:
  - The number of trees B: unlike bagging and random forests, boosting can overfit if B is too large, though this happens slowly. We use cross-validation to select B
  - The shrinkage parameter λ: a small positive number that controls the rate of learning (typically 0.01 or 0.001). λ slows down the learning process, allowing more and different shaped trees to attack the residuals. A very small λ may require a very large B to achieve good performance.
  - The number of splits in each tree d: controls the complexity of the boosted ensemble. Often a single split $d = 1$ works well (each tree is a stump). When $d = 1$, we are fitting an additive model, and in general d is the interaction depth.

Boosting: Some Remarks

- Note that in each step, we fit the model using current residuals, rather than the original outcome

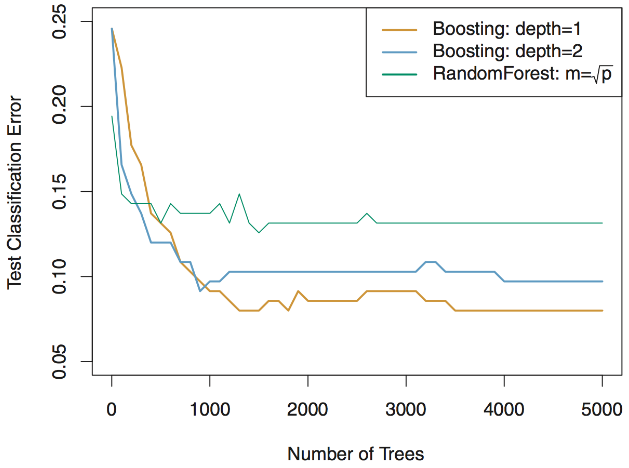
Boosting: Some Remarks

- Note that in each step, we fit the model using current residuals, rather than the original outcome
- The key in boosting is that each individual tree has to be small

Boosting: Some Remarks

- Note that in each step, we fit the model using current residuals, rather than the original outcome
- The key in boosting is that each individual tree has to be small
- A potential advantage of boosting over bagging and random forests is interpretability especially if we use small trees (using stumps gives an additive model)

Comparison on Gene Expression Data


Boosting: depth=1
Boosting: depth=2
RandomForest: m=$\sqrt{p}$
Test Classification Error
Number of Trees