Tree-Based Methods and Model Aggregation

June, 2019
Columbia University

Tree-based methods

Many statistical methods are theoretically motivated

and have a rich mathematical history.

Tree-based methods are not theoretically motivated, and have basically no associated mathematical theory.

But they can work really really well in practice; especially with aggregation methods.

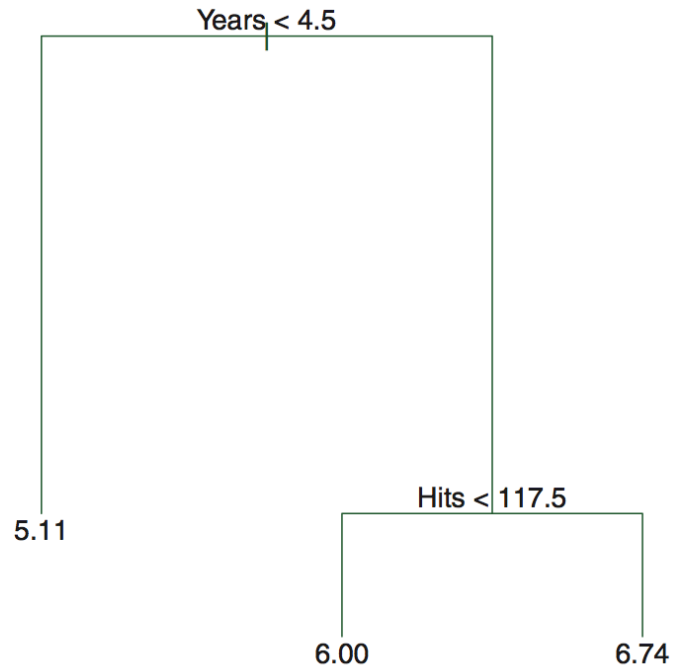
Tree-Based Methods

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

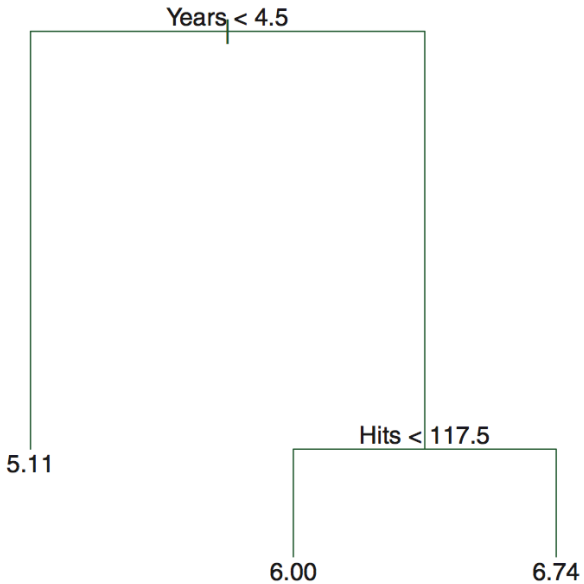
- *stratify* (or segment) the predictor space into small and simple regions
- for each observation, determine which segment it belongs to; the stratification of space can be expressed as a tree (hence the name)
- predict the outcome by the mean (regression) or mode (classification) of outcomes in that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



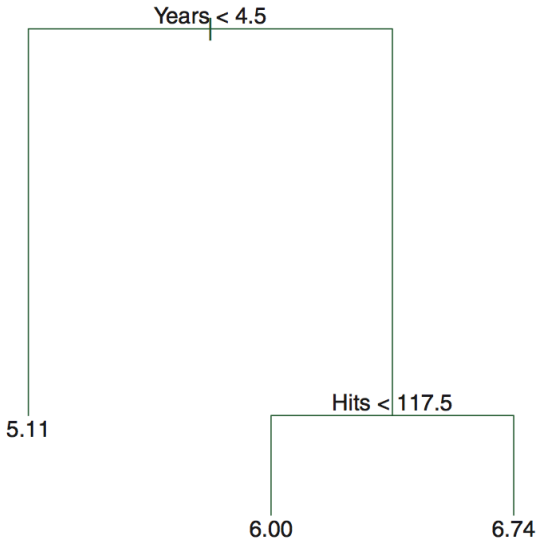
Toy Example: predicting salaries of baseball players II

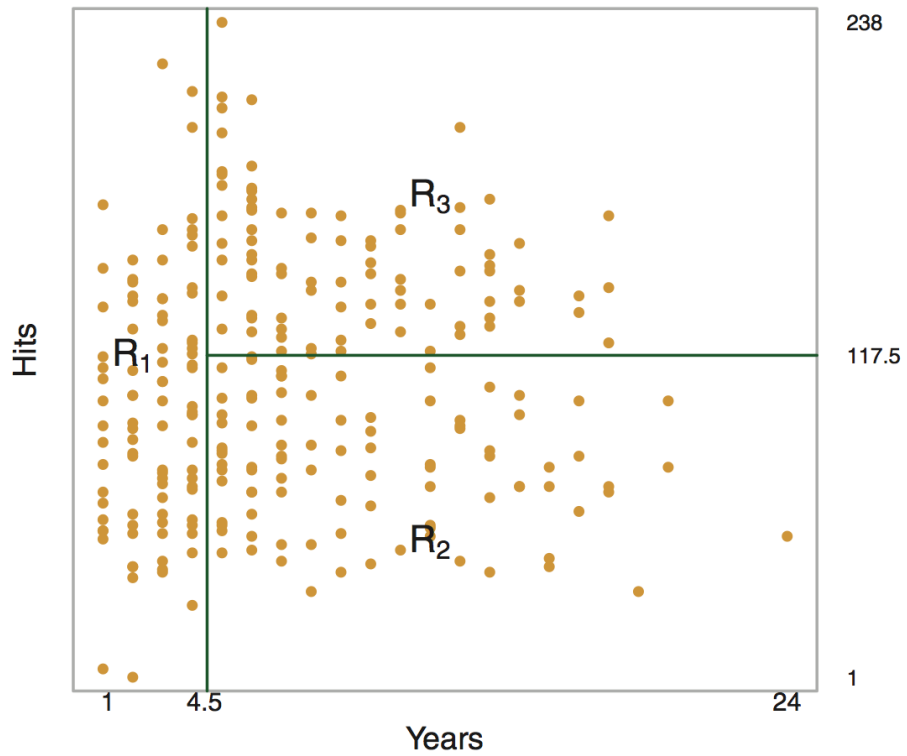


- The top split assigns observations having `Years` < 4.5 to the left branch
  - The predicted salary for players with < 4.5 years of experience is $\$1,000 \times e^{5.11}$, which is the mean salary for such players in the training data
- Players with > 4.5 `Years` experience are further divided based on number of `Hits` in 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.
- 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

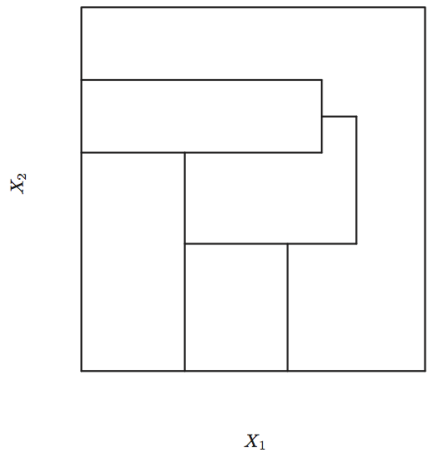
For fixed J. How to construct the regions $R_1, \ldots, R_J$?

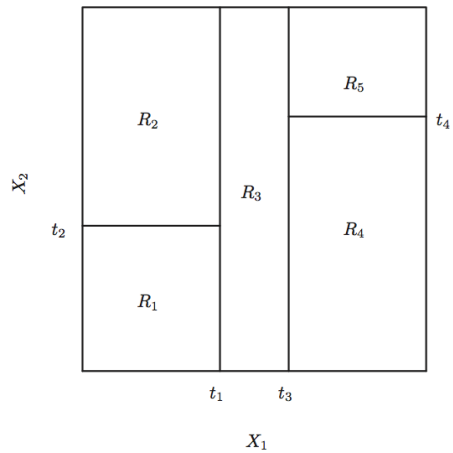
- Divide the predictor space into high-dimensional rectangles, or boxes (in theory any shape can be used, but this is simpler)
- Find boxes $R_1, \ldots, R_J$ that minimize the RSS

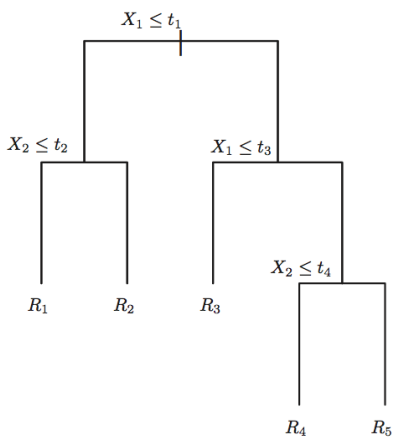
$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

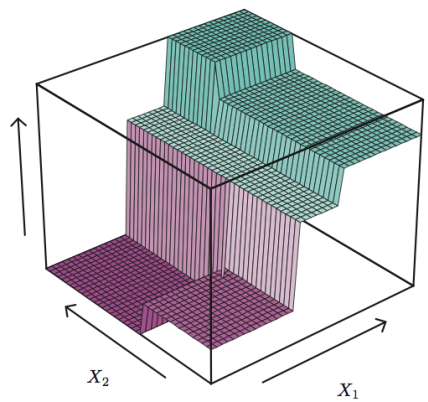
- Unfortunately, impractical to consider all possible partitions of space into J boxes
- We consider instead a *greedy* approach called recursive binary splitting
  - best split is determined at each given step, instead of the best global split

Recursive Binary Splitting

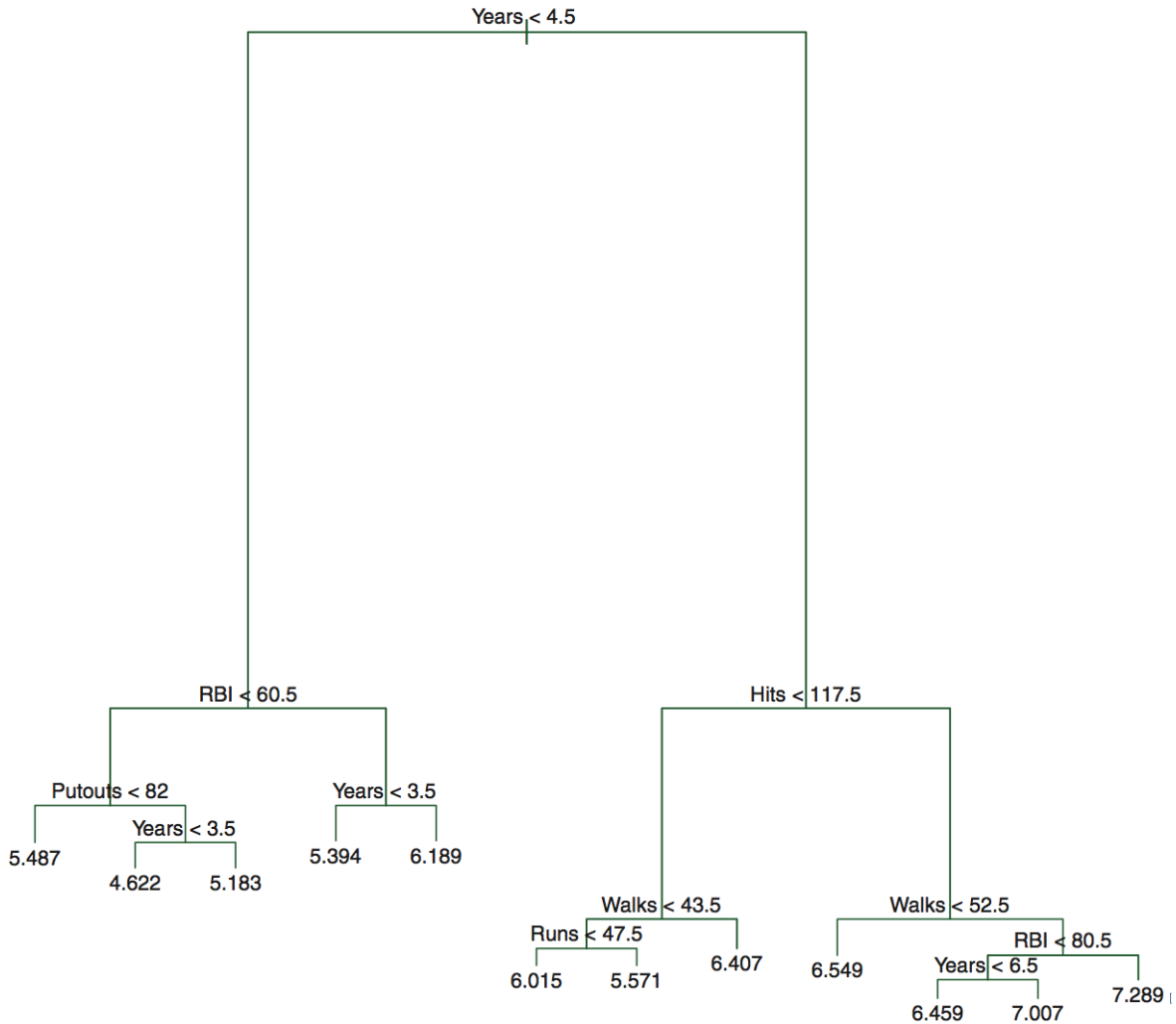








Full Tree for the `Hitters` Data Set

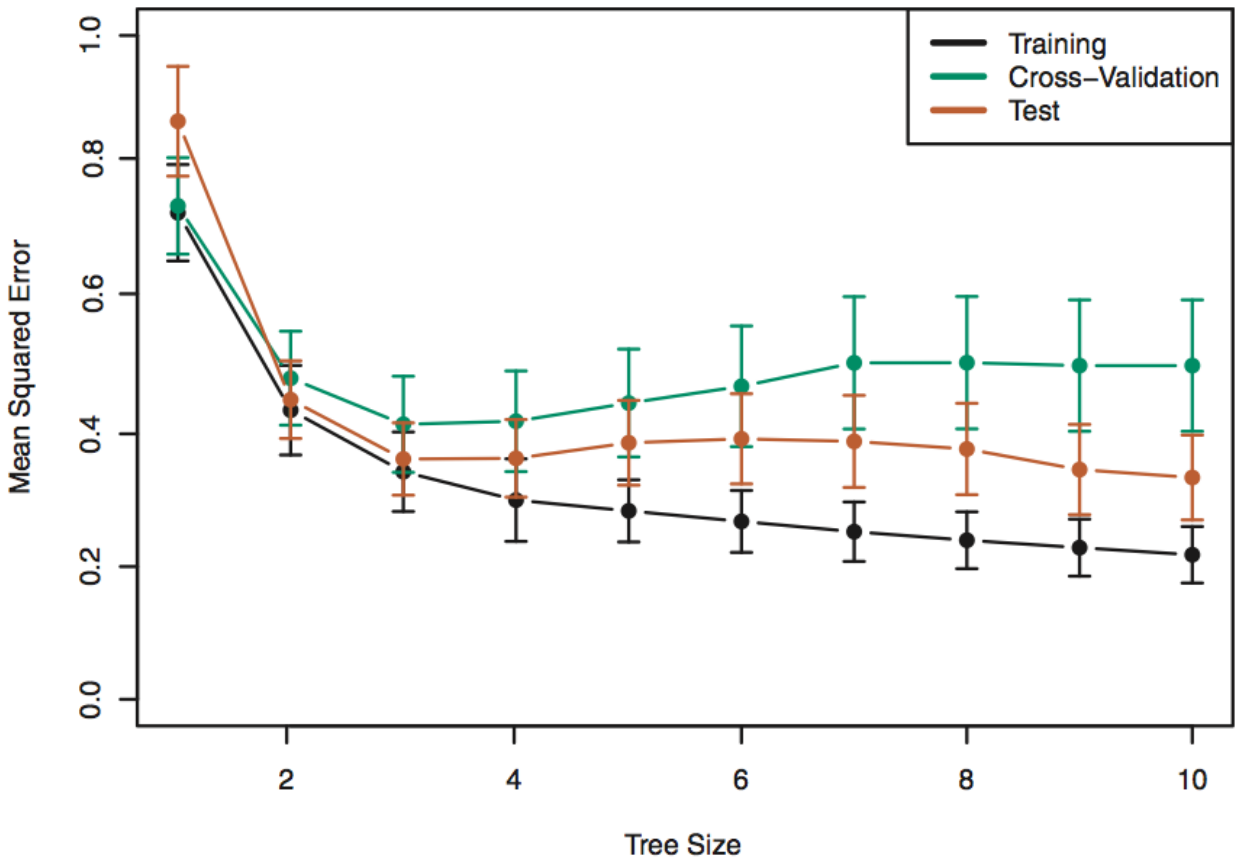


Tree Pruning

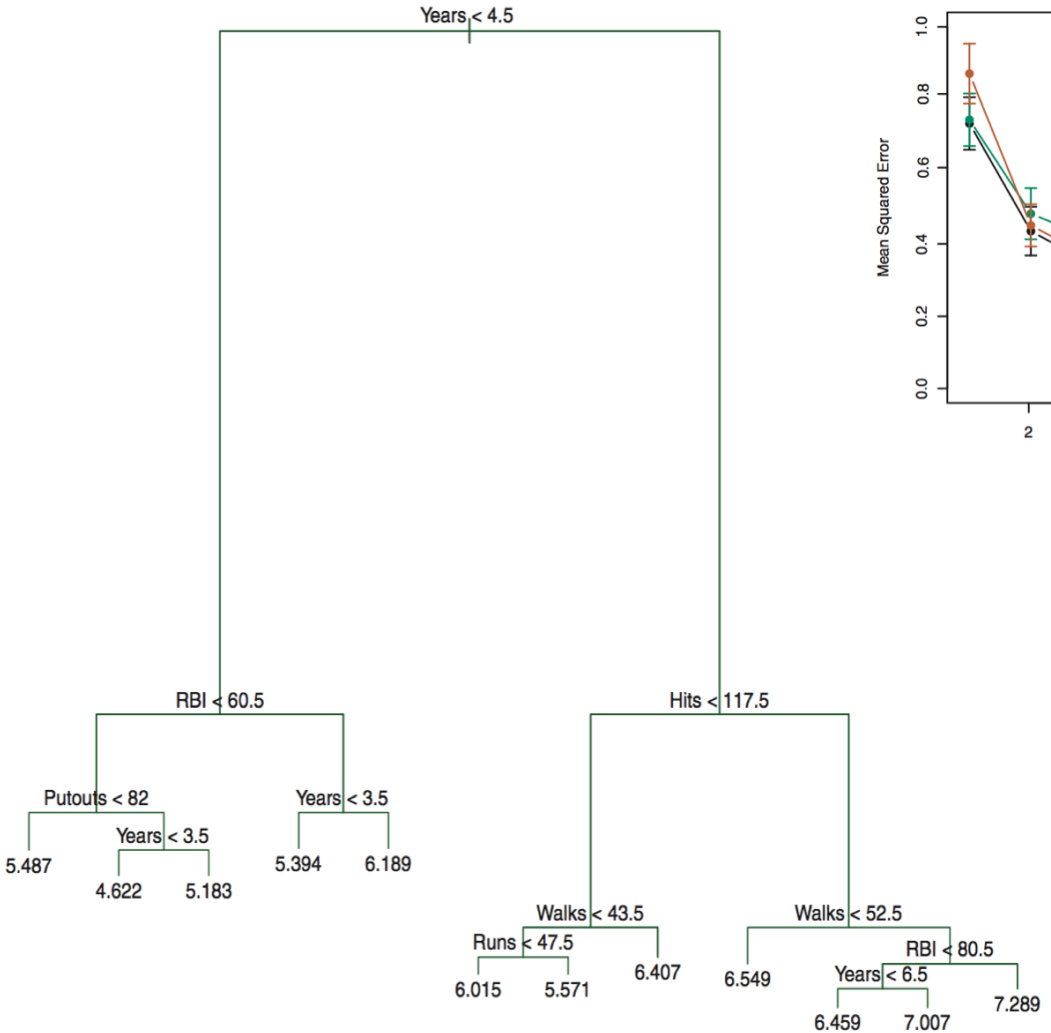
Number of splits/regions is the main tuning parameter to determine bias/variance tradeoff.

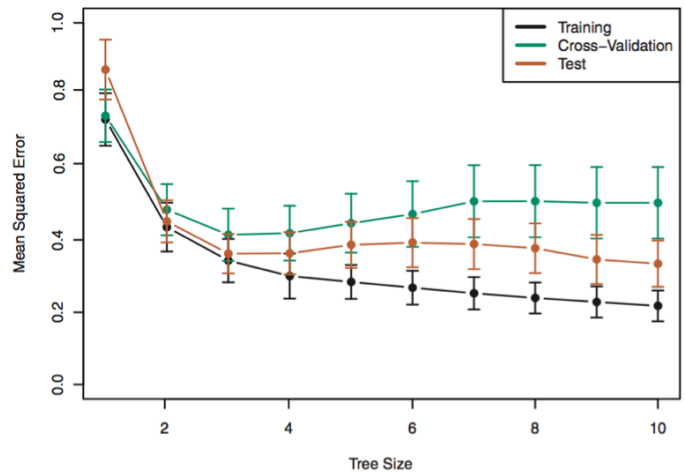
Trees sometimes use a slightly more convoluted parameter related to so-called pruning

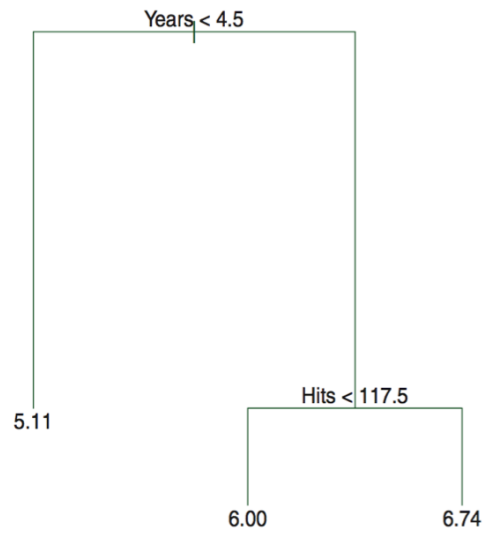
Essentially we use cross validation to select number of splits.


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...







COLUMBIA | MAILMAN SCHOOL OF PUBLIC HEALTH | COLUMBIA UNIVERSITY IRVING INSTITUTE FOR CLINICAL AND TRANSLATIONAL RESEARCH

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 RSS no longer makes sense! The misclassification error rate is the natural choice for growing the tree, however, it turns out that better measures exist, namely, the Gini Index and the Cross-Entropy
  - 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 Trees, Part I

- Suppose Y is qualitative with values in $1, 2, \ldots, K$.
- Need to adjust criteria for splitting nodes and pruning tree.
- For the mth node, representing the region R_m containing N_m observations, define

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

  the proportion of class k observations in node m.
- Classify the observations in node m to the majority class,

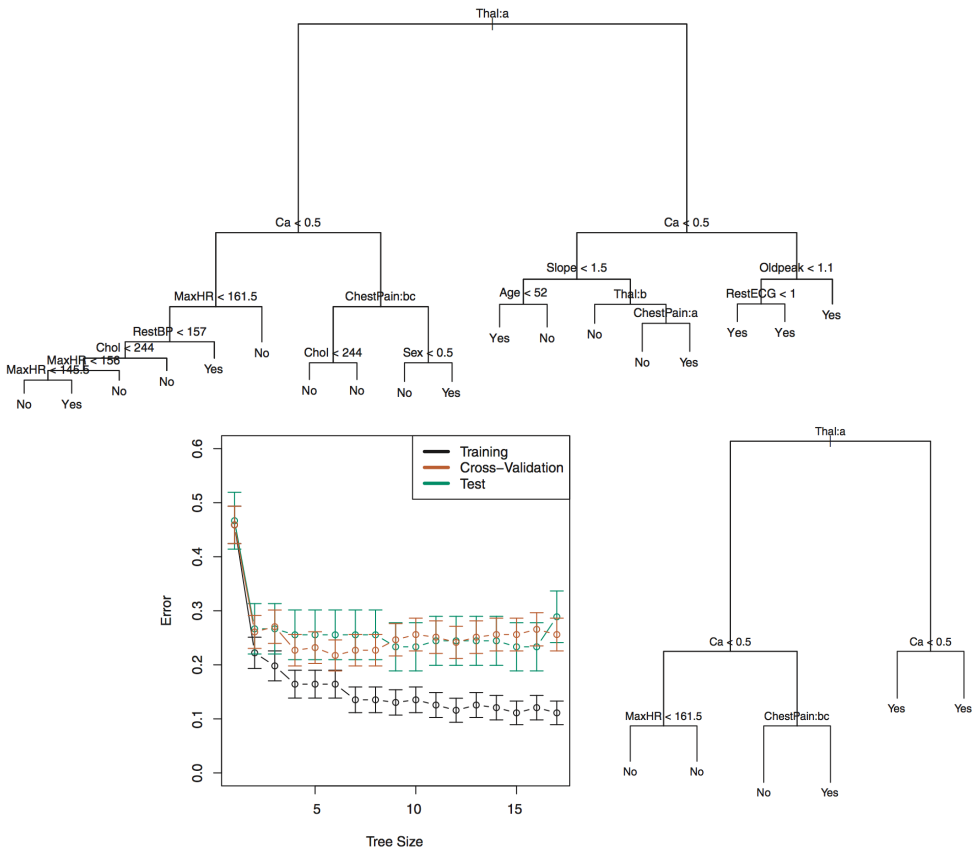
$$k(m) = \text{argmax}_k \hat{p}_{mk}.$$

Classification Trees, Part II

- Three possible measures of node impurity:
  - Misclassification Error: $1 - \hat{p}_{mk(m)}$.
  - Gini Index: $\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$.
  - Cross-Entropy or Deviance: $-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$.
- CE and Gini are more sensitive to changes in node probabilities, and hence are a better choice than MCE for growing the tree.
- Misclassification error is typically used for tree pruning.

Classification Tree for the Heart Data

Predicting heart disease from 13 demo/clinical features.



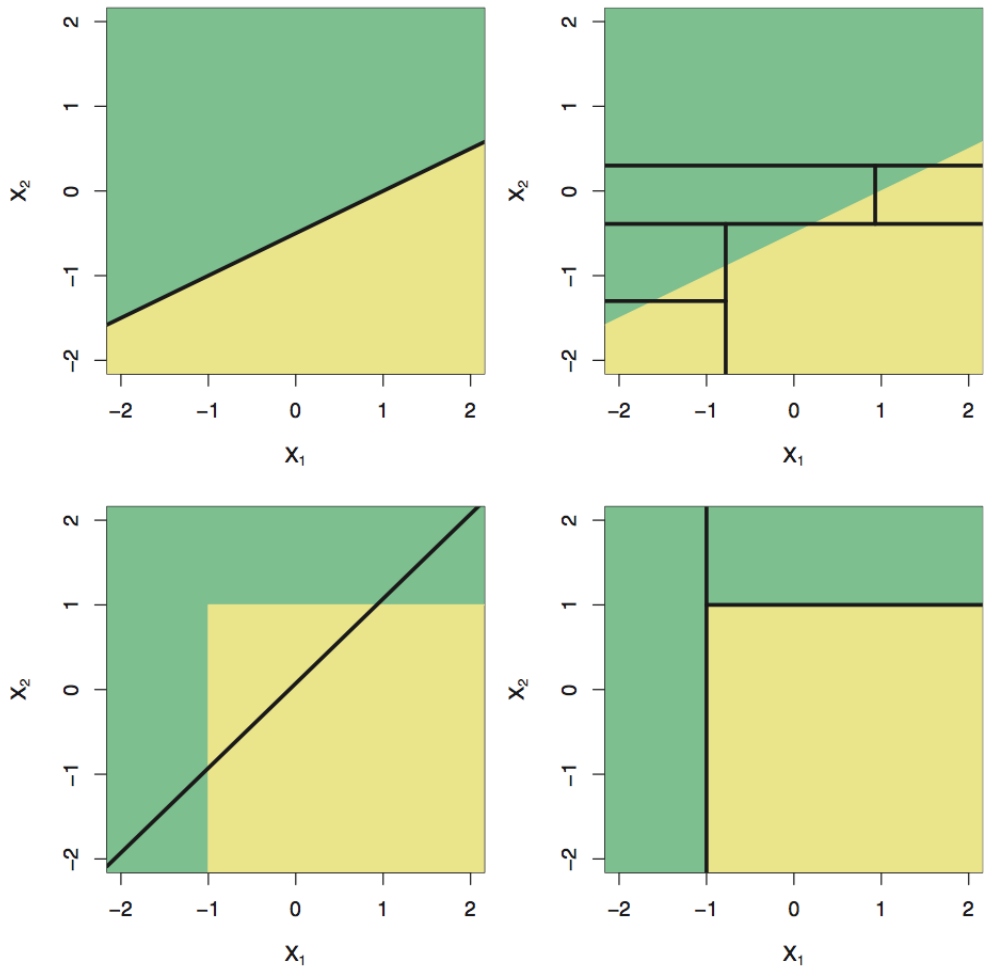
Survival Trees and beyond!

If we have a loss function to optimize, we can build a tree...

We greedily split the tree to minimize that loss function

For survival data, could use Cox loss.

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

There is a very useful tool in statistics known as the bootstrap

It is often used for statistical inference, however it can also be used in developing powerful predictive models.

Bootstrapping generally refers to the process of creating many new "artificial" datasets that resemble your original dataset (with added "noise").

Averaging Models

Suppose 10 different research groups were studying the same problem...

Imagine each group ran the same experiment: They collected an outcome y, and features x on many patients

Suppose each group built a predictive model, to predict y from x.

Now imagine we want to combine these models, to get an even stronger model. We could use

$$\hat{f}_{avg}(x) = \frac{1}{10}\sum_{b=1}^{10}\hat{f}^b(x)$$

Averaging Models

In practice, we do not have 10 datasets. However, we can use a procedure which simulates this process.

- We can take our original dataset (with n observations), and sample, with replacement n times from it.
- This will result in a new dataset with n observations (though some will be replicated).
- We can now build a predictive model on this new dataset.

If we do this many times, we will end up with many models, which we can then average.

Bagging

This is known as Bootstrap Aggregation or Bagging, and can substantially improve model performance.

It is most commonly used with tree-based models.

Bagging

Another (similar) motivation for 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

Bagging can be thought of as 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, $\mathrm{Var}(\bar{Z}) = \sigma^2/n$
- To reduce the variance, we could take many (B) training samples, build separate prediction model from each, and average resulting predictions

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

- Not possible in practice. In bagging, we use bootstrap samples

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

The Price of Bagging

Bagging trees can lead to improved predictions, but
...interpretability is lost!

Out Of Bag Samples

- No need to do cross-validation to estimate prediction error for bagging.
- Instead, use out of bag (OOB) samples to compute the error.
- That is, for each observation $z_i = (x_i, y_i)$, predict the response *by taking the average of the predictions corresponding to bootstrap samples in which z_i didn't appear*.
- Gives similar results to cross-validation.

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

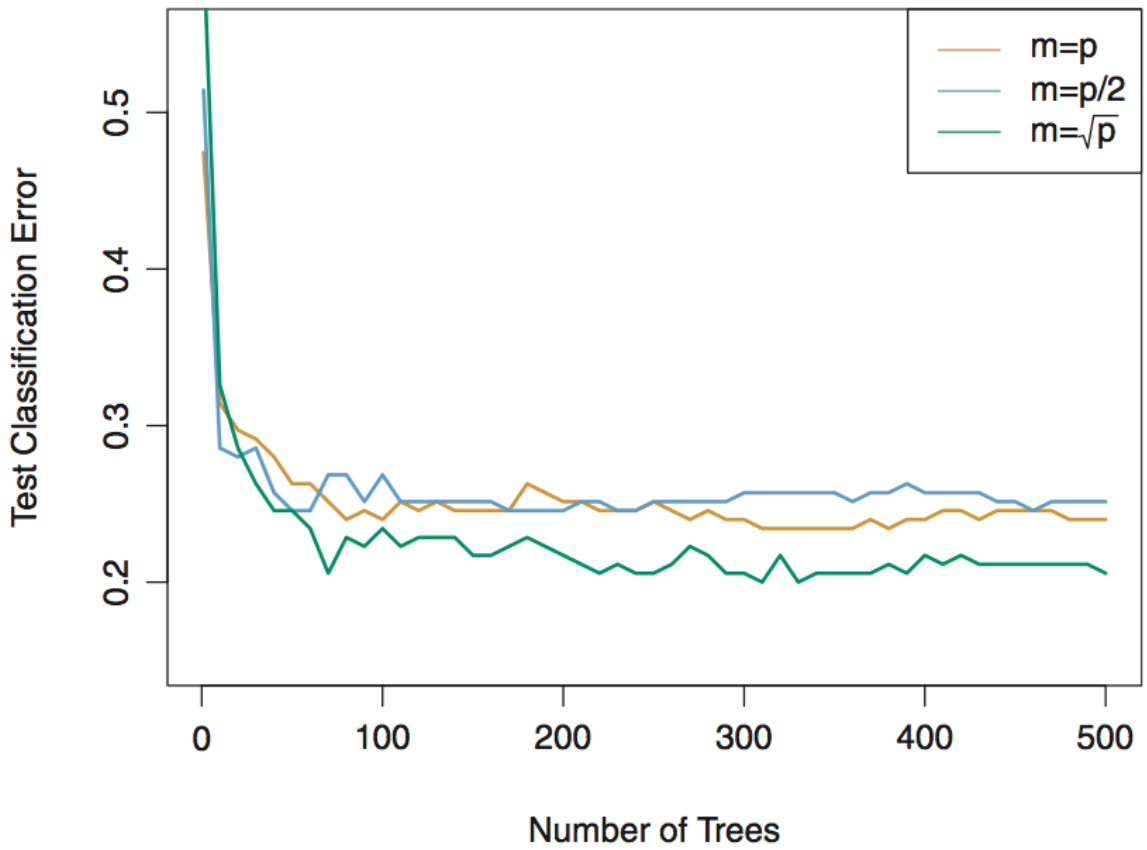
Random Forests: The Motivation

- 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 significant 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
- 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



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
- 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: As a local descent algorithm

Gradient descent logic for minimizing a function:

- Should step in a greedy, locally steepest, descent direction
- As we move, locally steepest direction will change
- Take small enough steps to ensure downhill traversal

Boosting logic

- Should adjust our fitted function, by choosing splits greedily (to most reduce RSS)
- As we adjust fitted function, best greedy split choice changes
- Add in only small amounts of each tree to always use nearly locally-optimal greedy splitting.

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,

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

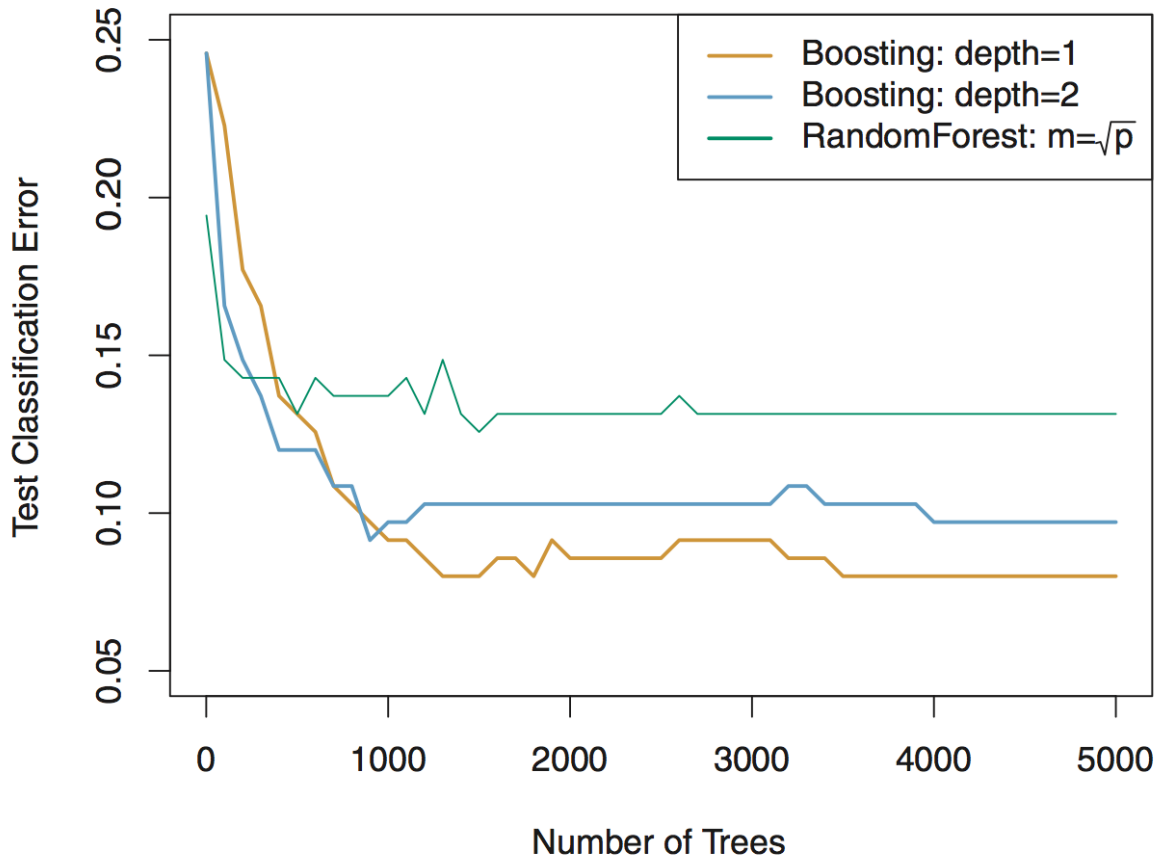
Tuning Parameters

- Boosting has 3 tuning parameters:
  - 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
- The key in boosting is that each individual tree has to be small
- Boosting tends to do very well with little tuning (can take a while to run though)

Comparison on Gene Expression Data



Aggregating Models

Boosting/Bagging/Random Forests are primarily used for aggregating similar models (eg. trees).

What if we want combine a boosted tree model, a linear-Lasso, and a sparse-additive model? One way to "combine" is just use

cross-validation to find the *best* model; and then use that model in the future.

However, a linear combination of the models may perform better than any single model.

Stacking is a method that uses cross-validation to find that combination.

Stacking

The stacking algorithm (using LOO CV):

1. For each modeling procedure, k, and each observation, i:
   1.1 Build a predictive model $\hat{f}_{k,-i}$ using all obs except i
   1.2 Evaluate the model on the left out obs $\hat{f}_{k,-i}(x_i)$
2. Solve the linear regression problem in α:

$$\hat{\alpha}_1, \ldots, \hat{\alpha}_j \leftarrow \text{argmin} \sum_i \left(y_i - \sum_j \alpha_j \hat{f}_{k,-i}(x_i) \right)^2$$

3. Refit our models to all the data, to obtain $\hat{f}_k$ for each k
4. In the future, use the model $\hat{f}(x) \leftarrow \sum_j \alpha_j \hat{f}_j(x)$

Illustrating the algorithm

Suppose we want to combine a boosted tree model, and a lasso model:

- For each observation i, we would fit our two models to the rest of the data.
- *fitting the two models* will require additional internal CV for tuning parameter selection (eg. λ in the lasso, tree-depth, etc. for boosting). For each i, the choices determined by CV may be different.
- After determining the optimal weights we refit our models (including any internal CV) on all the observations to determine the final models used in our mixture

The tuning parameter choices will likely differ at each stage of CV; and from choices for the final mixture.

Cross-validation in Stacking

- By splitting up the optimization into 2 stages, we avoid overfitting
- Can use k fold CV, or training-test split rather than LOO-CV.
- Can be very computationally intensive — still need to evaluate performance.