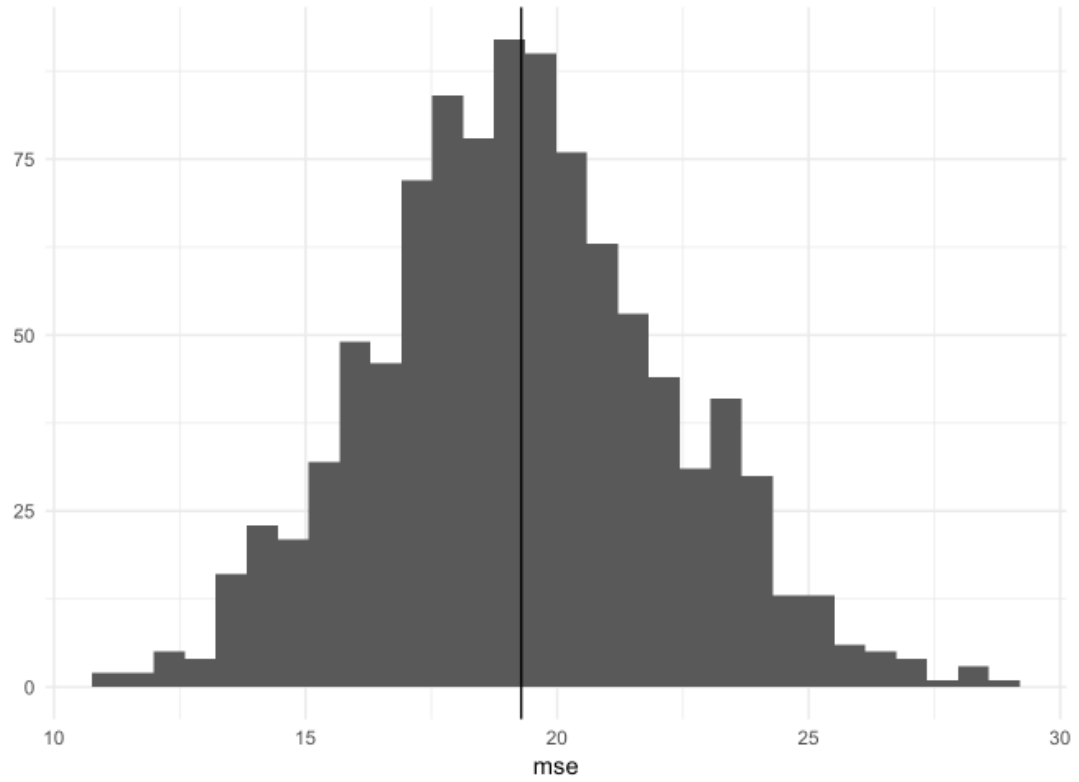
2

Benefits/drawbacks to decision trees

- Decision trees are an entirely different method of estimating functional forms as compared to linear regression. There are some benefits to trees:
  - They are easy to explain. Most people, even if they lack statistical training, can understand decision trees.
  - They are easily presented as visualizations, and pretty interpretable.
  - Qualitative predictors are easily handled without the need to create a long series of dummy variables.
- However there are also drawbacks to trees:
  - Their accuracy rates are generally lower than other regression and classification approaches.
  - Trees can be non-robust. That is, a small change in the data or inclusion/exclusion of a handful of observations can dramatically alter the final estimated tree.

Example

- let's estimate a decision tree for the highway mileage example (N=392) by splitting the data into a training/test set (70/30%) and estimating the test MSE, and repeat this process 1000 times using random combinations of training/test sets:



The Basics of Decision Trees

- Regression Trees
- Classification Trees
- Pruning Trees
- Trees vs. Linear Models
- Advantages and Disadvantages of Trees

- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification

Partitioning Up the Predictor Space

- One way to make predictions in a regression problem is to divide the predictor space (i.e. all the possible values for $X_1, X_2, \ldots, X_p$) into distinct regions, say $R_1, R_2, \ldots, R_p$
- Then for every X that falls in a particular region (say R_j) we make the same prediction,

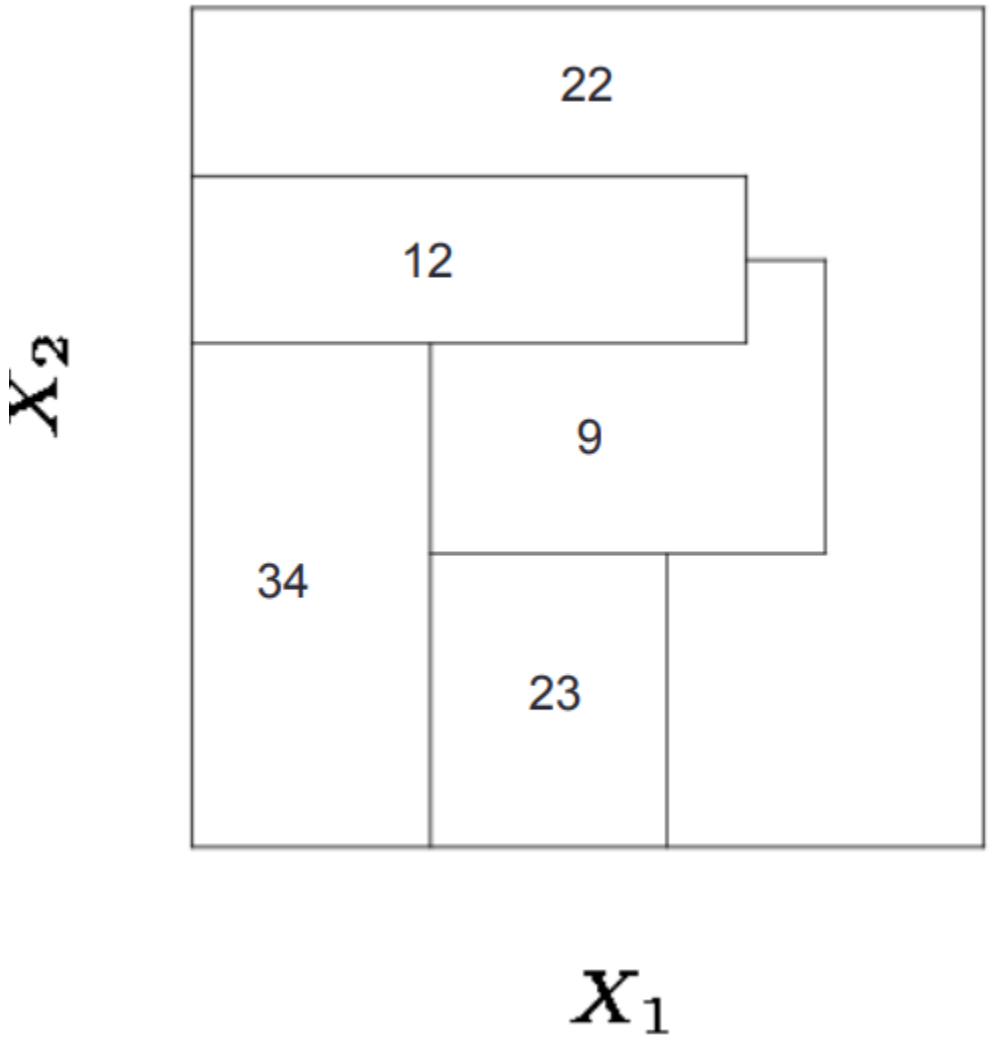
REGRESSION TREES

- Suppose for example we have two regions R_1 and R_2 with $\hat{Y}_1 = 10, \hat{Y}_2 = 20$

- Then for any value of X such that $X \in R_1$ we would predict 10, otherwise if we would predict 20.

The General View

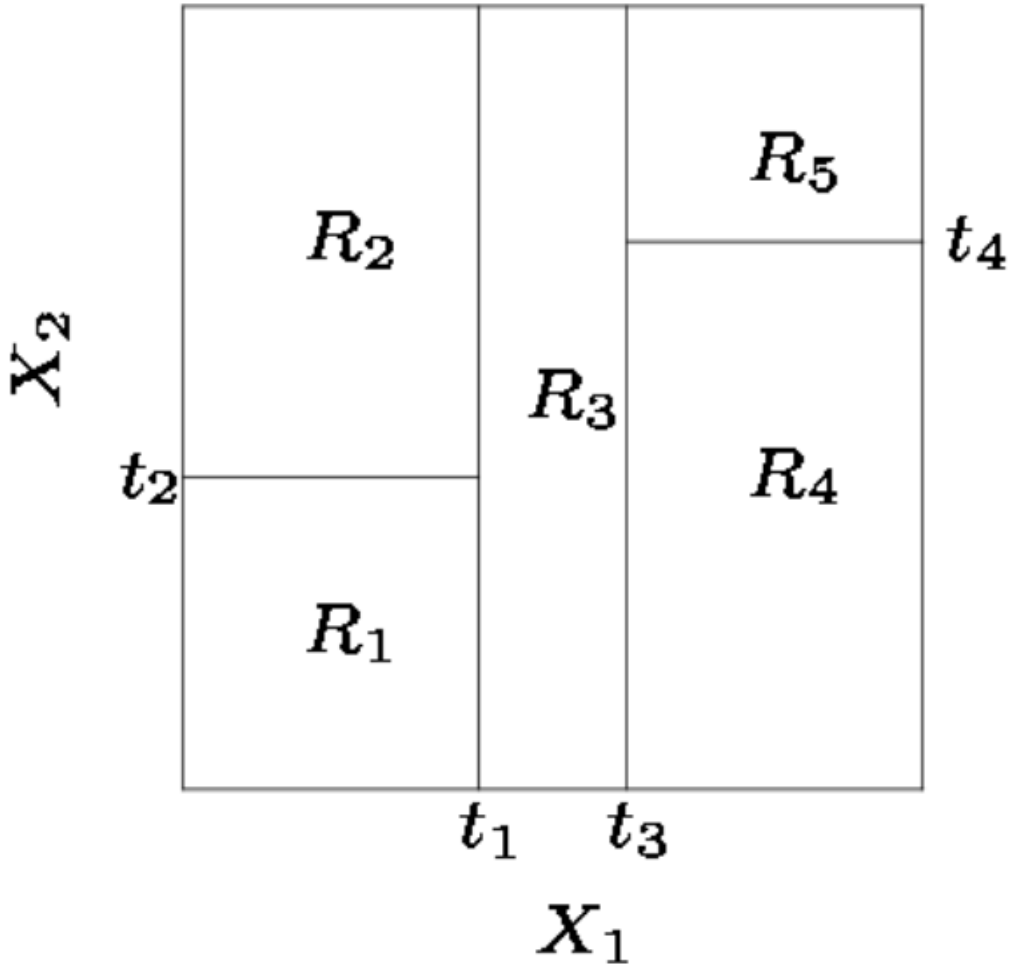
- Here we have two predictors and five distinct regions
- Depending on which region our new X comes from we would make one of five possible predictions for Y.



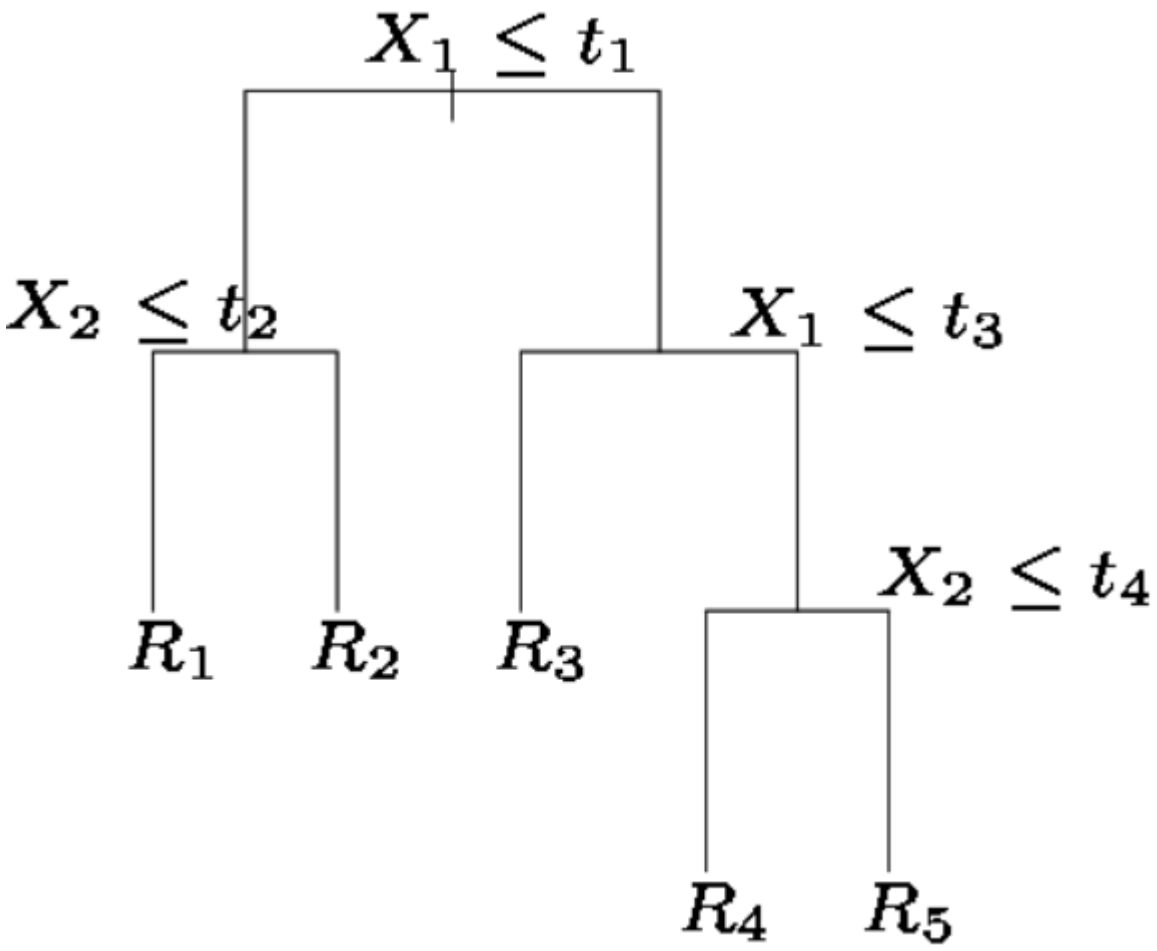
Splitting the X Variables

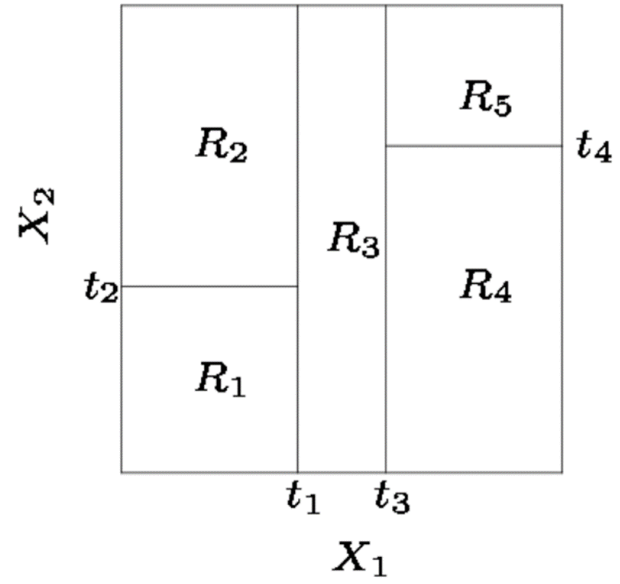
- Generally we create the partitions by iteratively splitting one of the X variables into two regions

1. First split on $X_1 = t_1$
2. If $X_1 < t_1$ split on $X_2 = t_2$
3. If $X_1 > t_1$ split on $X_1 = t_3$
4. If $X_1 > t_3$ split on $X_2 = t_4$



Splitting the X Variable

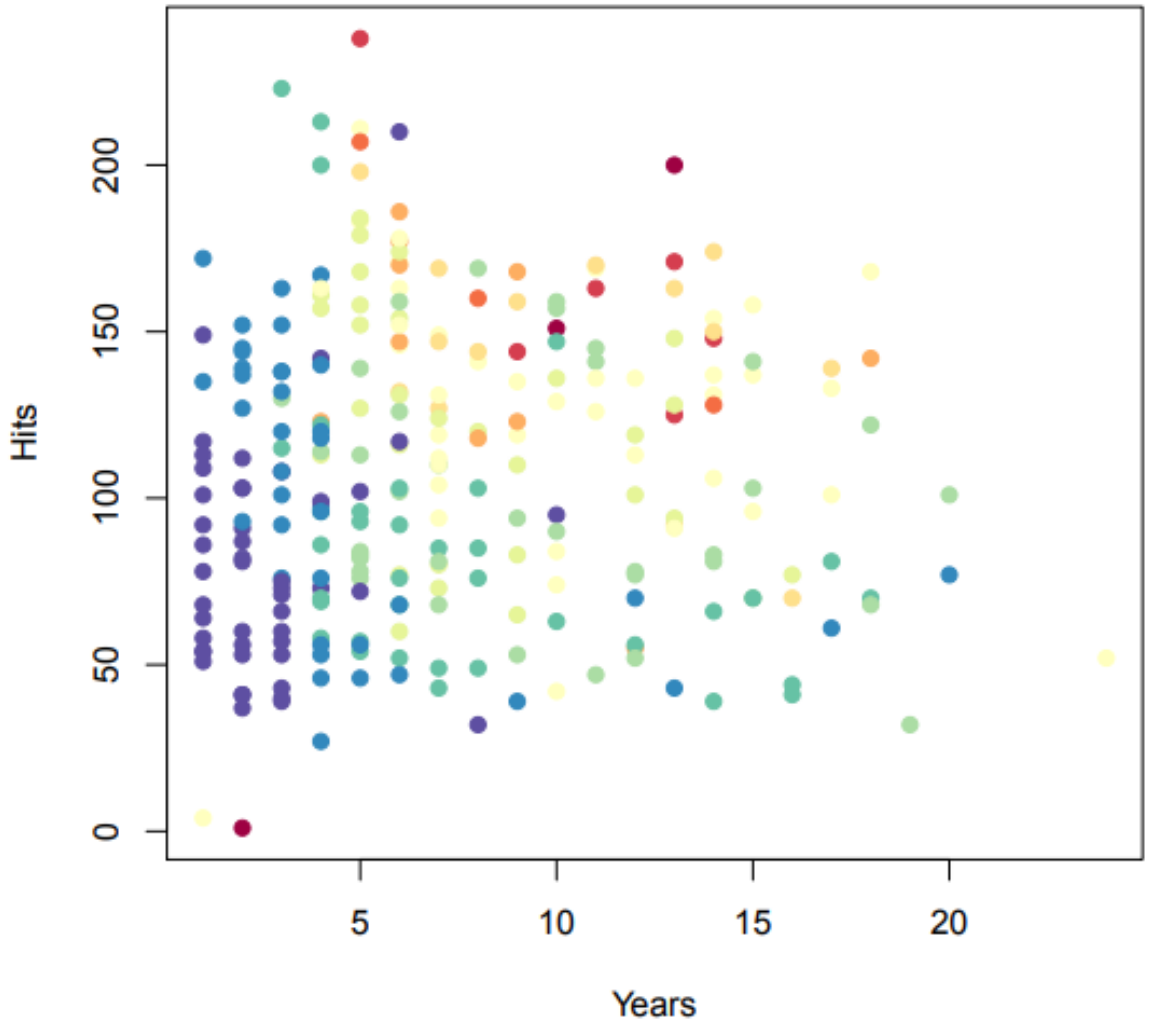




- When we create partitions this way we can always represent them using a tree structure.
- This provides a very simple way to explain the model to a non-expert i.e. your boss!

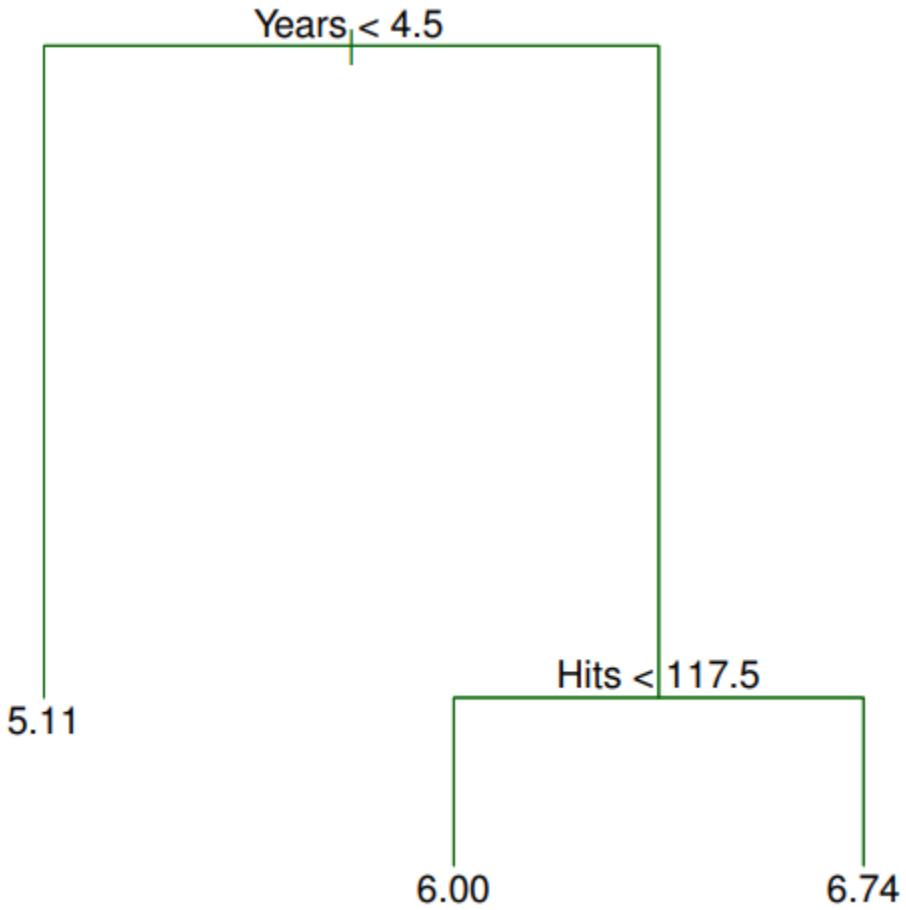
Baseball salary data: how would you stratify it?

- Baseball salary data: how would you stratify it? Salary is color-coded from low (blue, green) to high (yellow,red)



Decision tree for these data

- Another way of creating transformations of a variable — cut the variable into distinct regions.

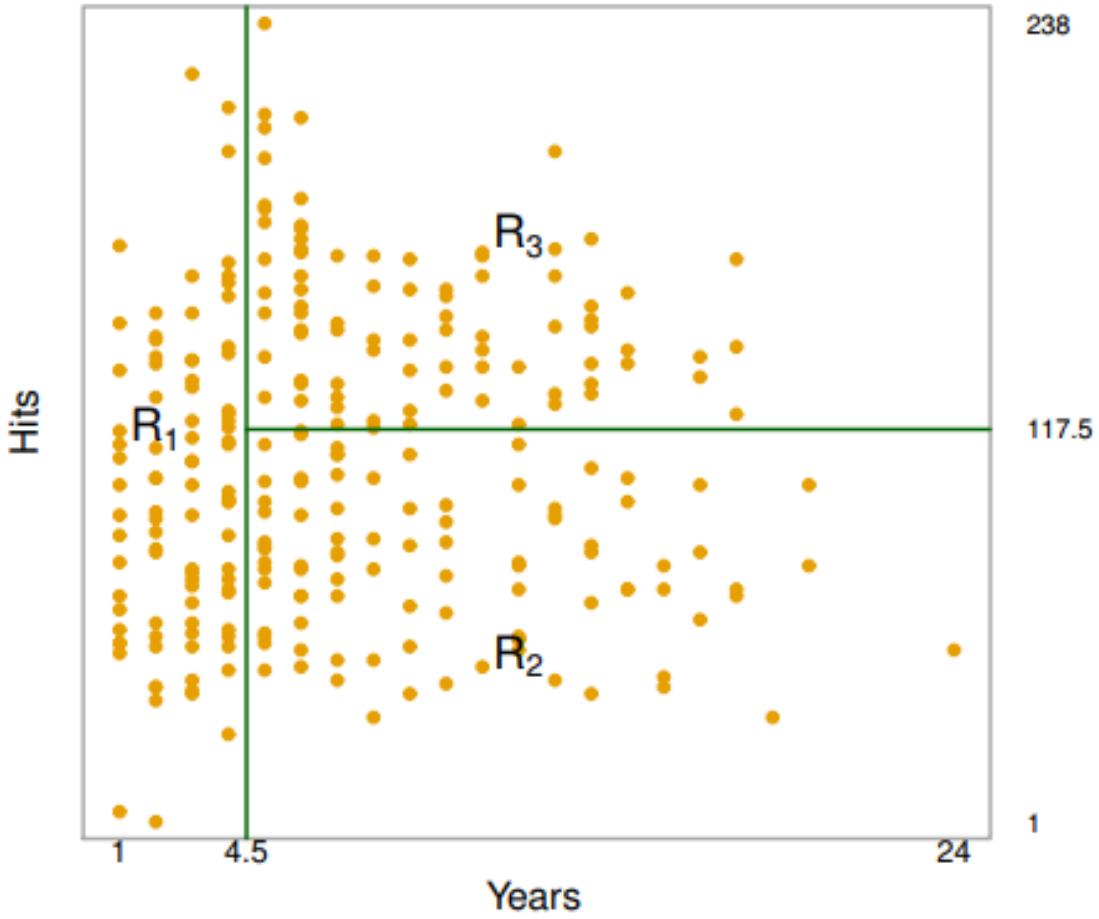


Details of previous figure

- For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to Years=4.5, and the right-hand branch corresponds to Years>=4.5.
- The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.

Results

- Overall, the tree stratifies or segments the players into three regions of predictor space: R_1={X | Years< 4.5}, R_2 ={X | Years>=4.5, Hits=4.5, Hits>=117.5}, and R_3 ={X | Years>=4.5, Hits>=117.5}.



Terminology for Trees

- In keeping with the tree analogy, the regions R_1, R_2, and R_3 are known as terminal nodes.
- Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as internal nodes
- In the hitters tree, the two internal nodes are indicated by the text Years < 4.5 and Hits < 117.5.

Interpretation of Results

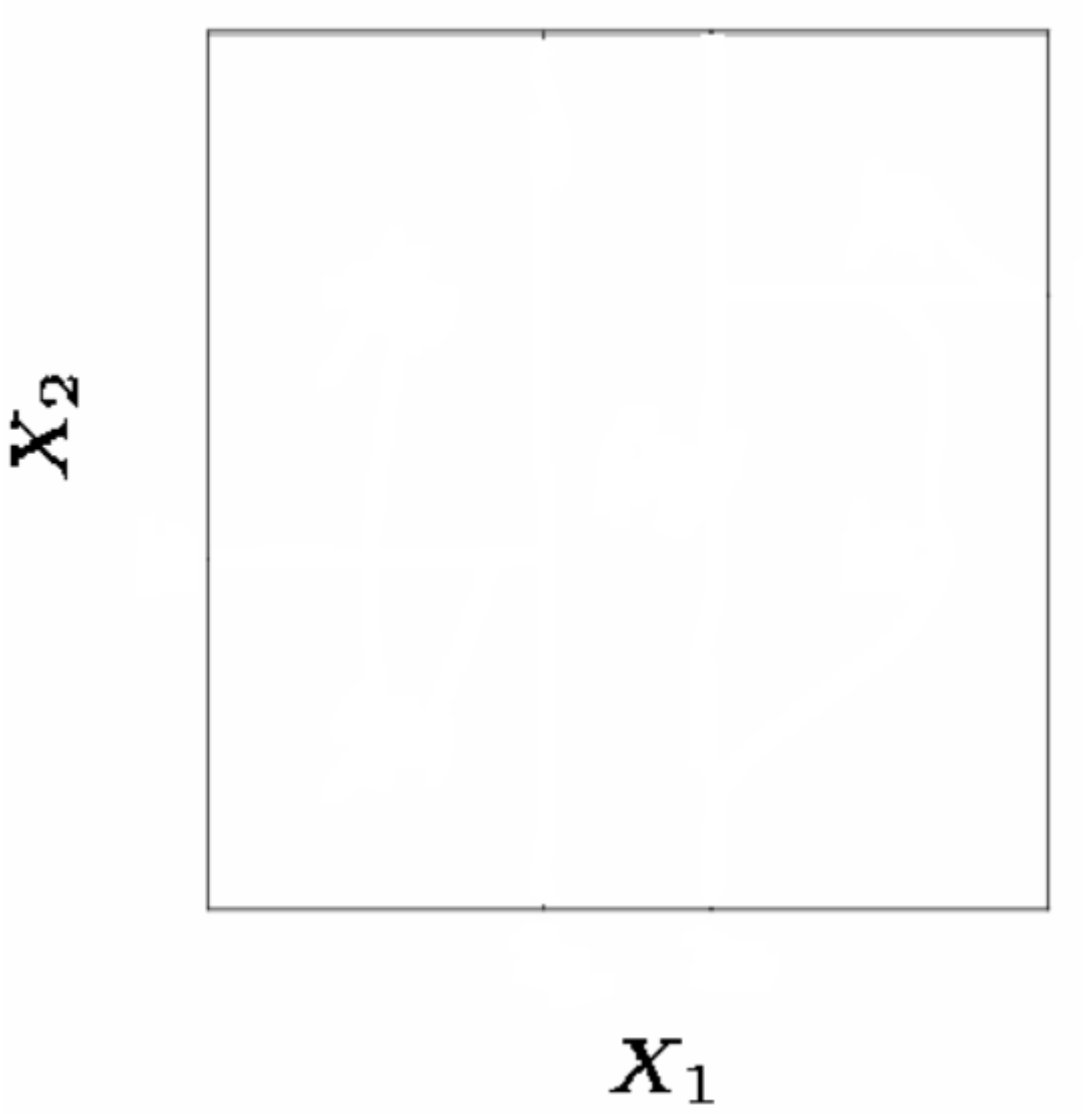
- Years is the most important factor in determining Salary, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in his Salary.
- But among players who have been in the major leagues for five or more years, the number of Hits made in the previous year does affect Salary, and players who made more Hits last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain

Details of the tree-building process

- Where to split? i.e. how do we decide on what regions to use i.e. R_1, R_2,…, R_p or equivalently what tree structure should we use?
- What values should we use for $\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_k$?
- Simple!
- For region R_j the best prediction is simply the average of all the responses from our training data that fell in region R_j.
- We divide the predictor space — that is, the set of possible values for $\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_p$— into J distinct and non-overlapping regions $\mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_j$. 2. For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

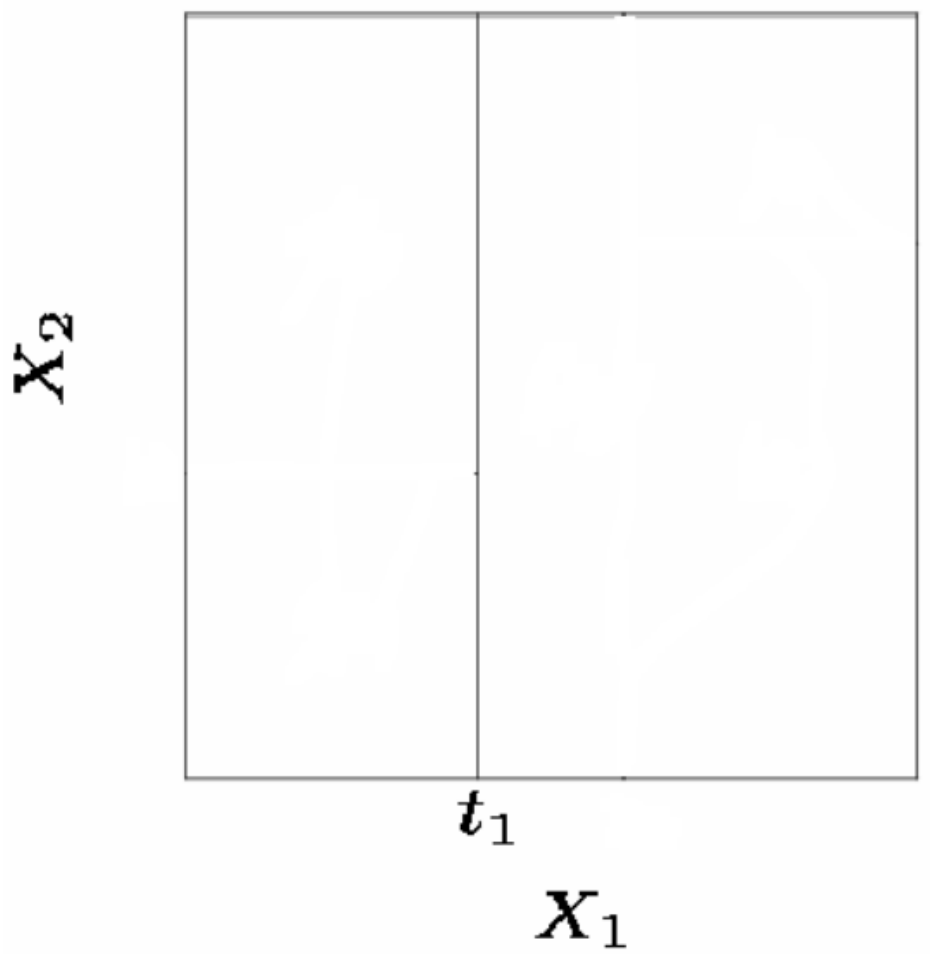
Where to Split?

- We consider splitting into two regions, $X_j > s$ and $X_j < s$ for all possible values of s and j.
- We then choose the s and j that results in the lowest MSE on the training data.



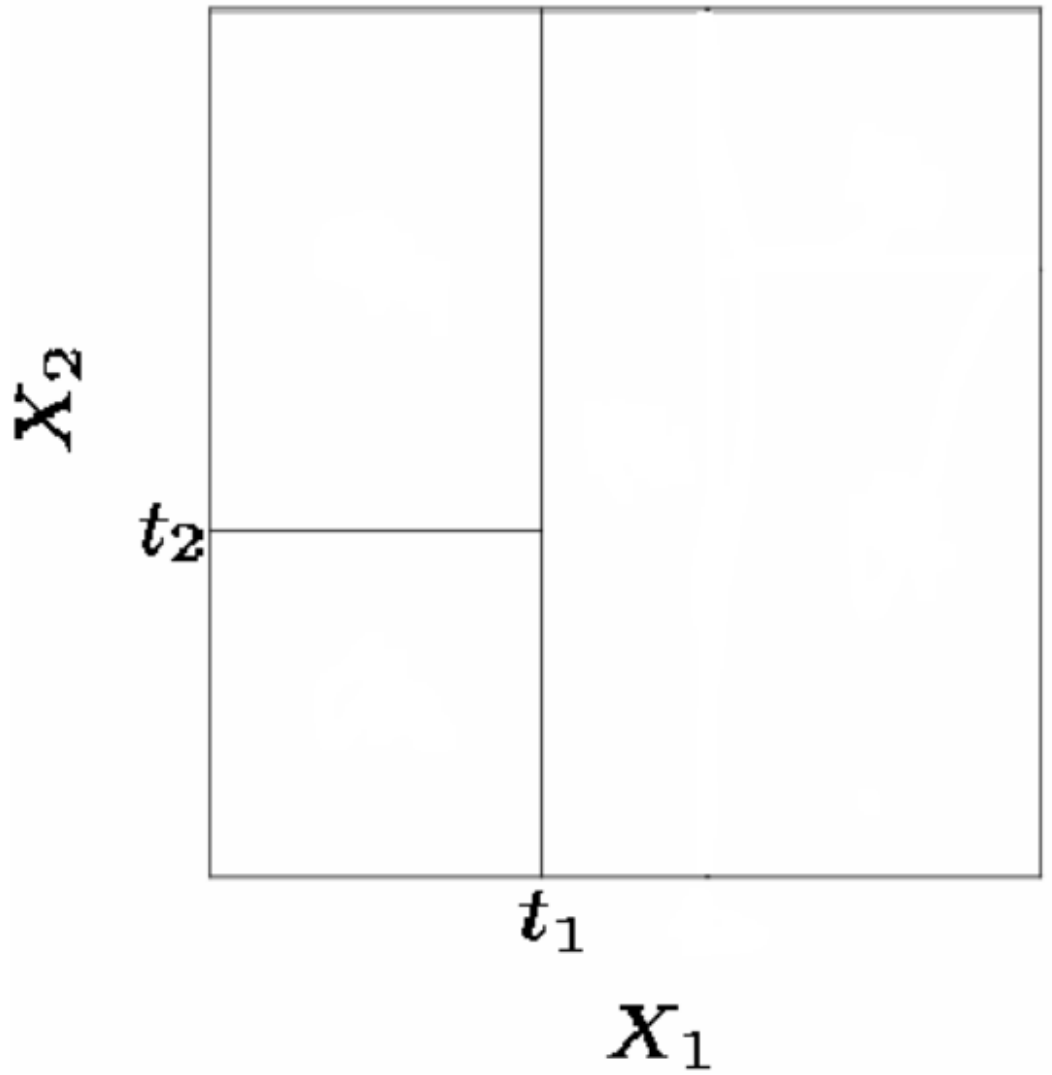
Where to Split?

- Here the optimal split was on X_1 at point t_1.
- Now we repeat the process looking for the next best split except that we must also consider whether to split the first region or the second region up.
- Again the criteria is smallest MSE.



Where to Split?

- Here the optimal split was the left region on X_2 at point 2.
- This process continues until our regions have too few observations to continue e.g. all regions have 5 or fewer points.



More details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_j$ that minimize the RSS, given by

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

- Where $\hat{y}_{R_j}$ is the mean response for the training observations within the *j*th box.

More details of the tree-building process

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a top-down, greedy approach that is known as recursive binary splitting.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Details— Continued

We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X | X_j < s \text{ s}\}$ and $\{X | X_j \geq s \text{ s}\}$ leads to the greatest possible reduction in RSS.

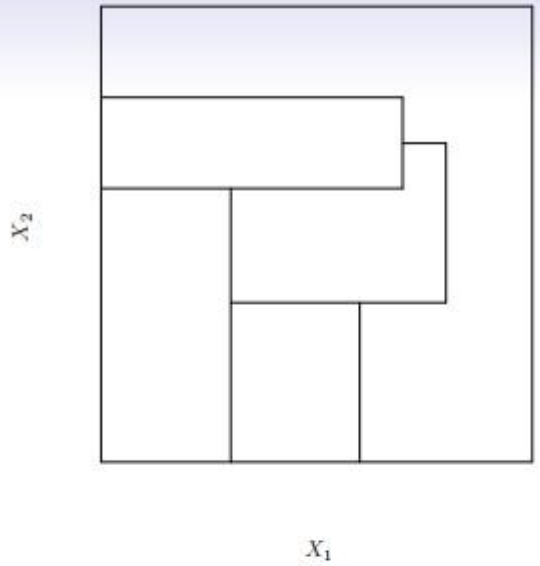
Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.

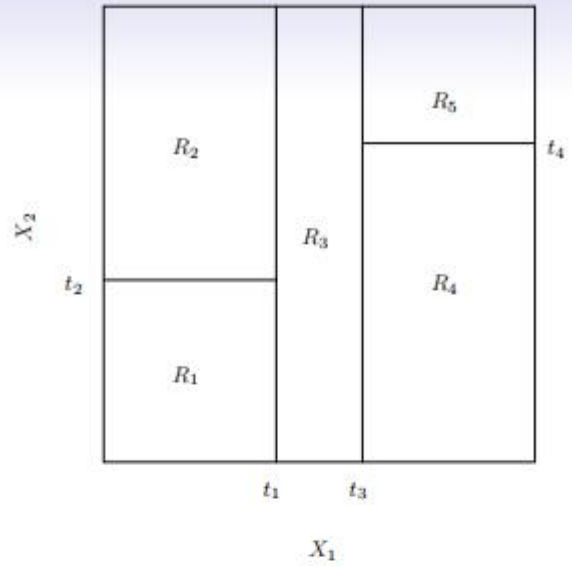
However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.

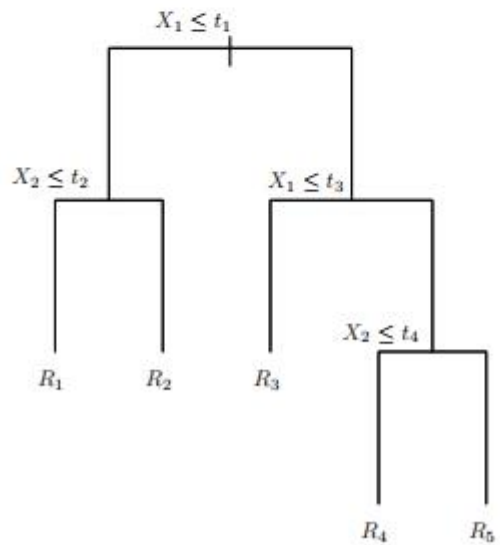
Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

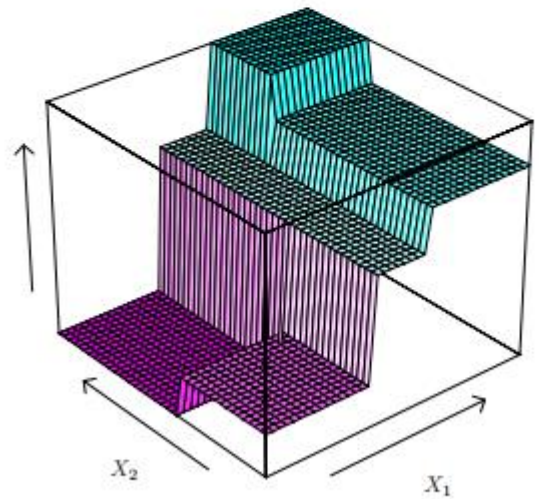
Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in the next slide.


X_2
X_1


R_1
R_2
R_3
R_4
R_5
t_1
t_2
t_3
t_4
X_2
X_1


$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
R_1
R_2
R_3
R_4
R_5


X_2
X_1

Details of previous figure

Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting.

Top Right: The output of recursive binary splitting on a two-dimensional example.

Bottom Left: A tree corresponding to the partition in the top right panel.

Bottom Right: A perspective plot of the prediction surface corresponding to that tree

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. Why?
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_j$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too short-sighted: a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in RSS later on.

Pruning a tree— continued

- A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree
- Cost complexity pruning — also known as weakest link pruning — is used to do this
- we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

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

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Choosing the best subtree

- How do we know how far back to prune the tree? We use **cross validation** to see which tree has the lowest error rate.
- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: tree algorithm

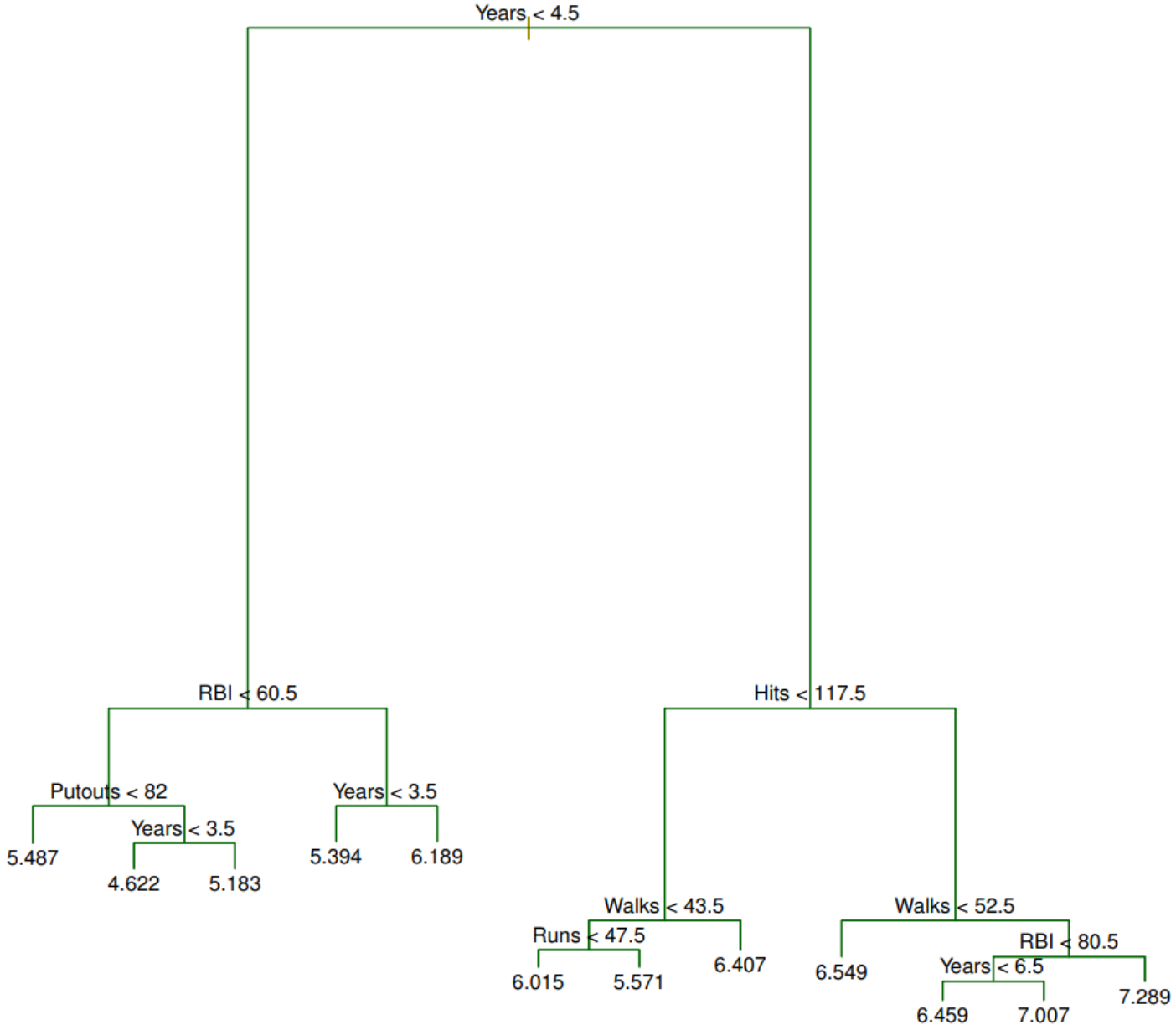
1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
    1. Repeat Steps 1 and 2 on the $\frac{K-1}{K}$ th fraction of the training data, excluding the kth fold.
    2. Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

    Average the results, and pick α to minimize the average error.
1. Return the subtree from Step 2 that corresponds to the chosen value of α.

Baseball example continued

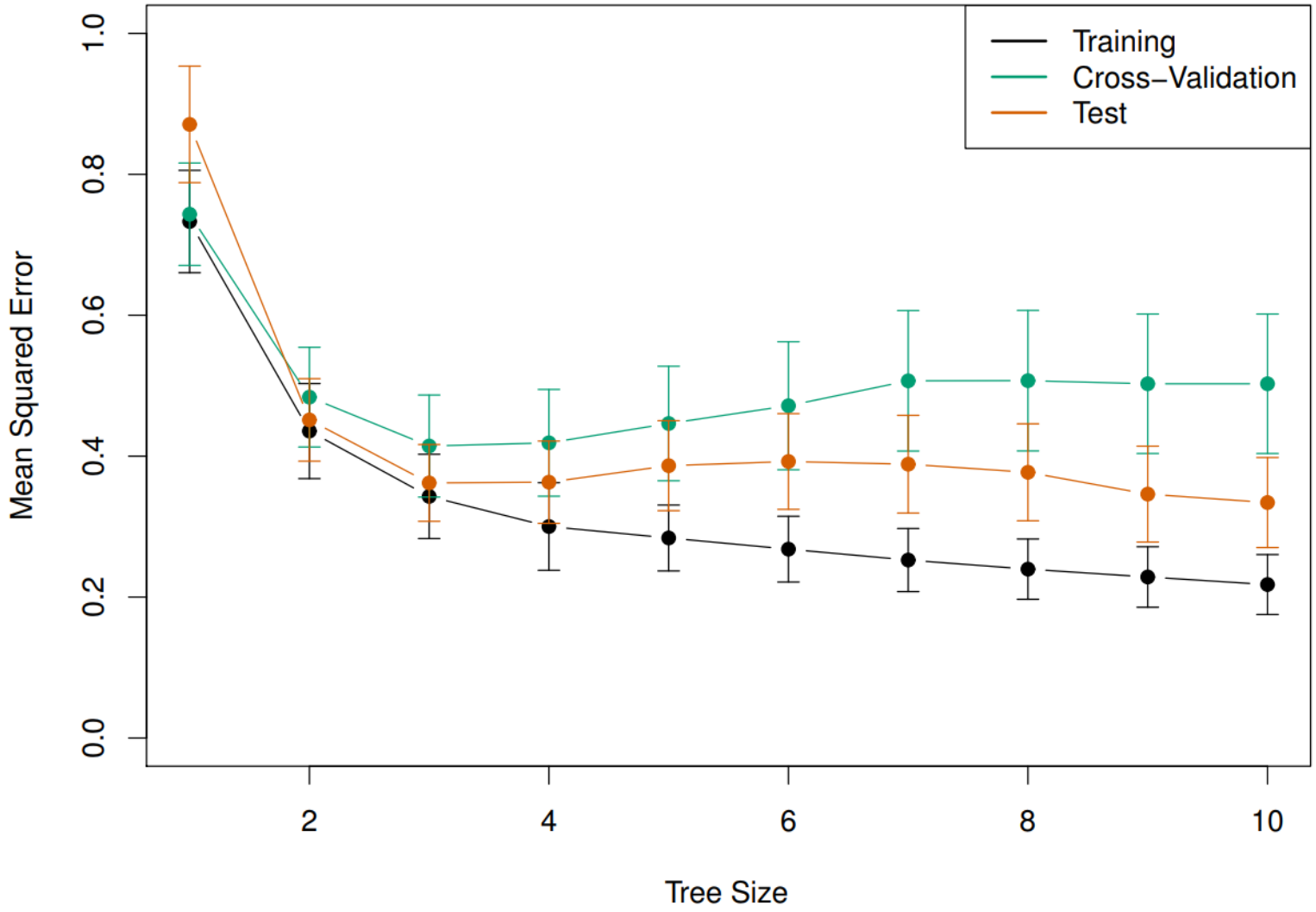
- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied α in in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

Baseball example continued



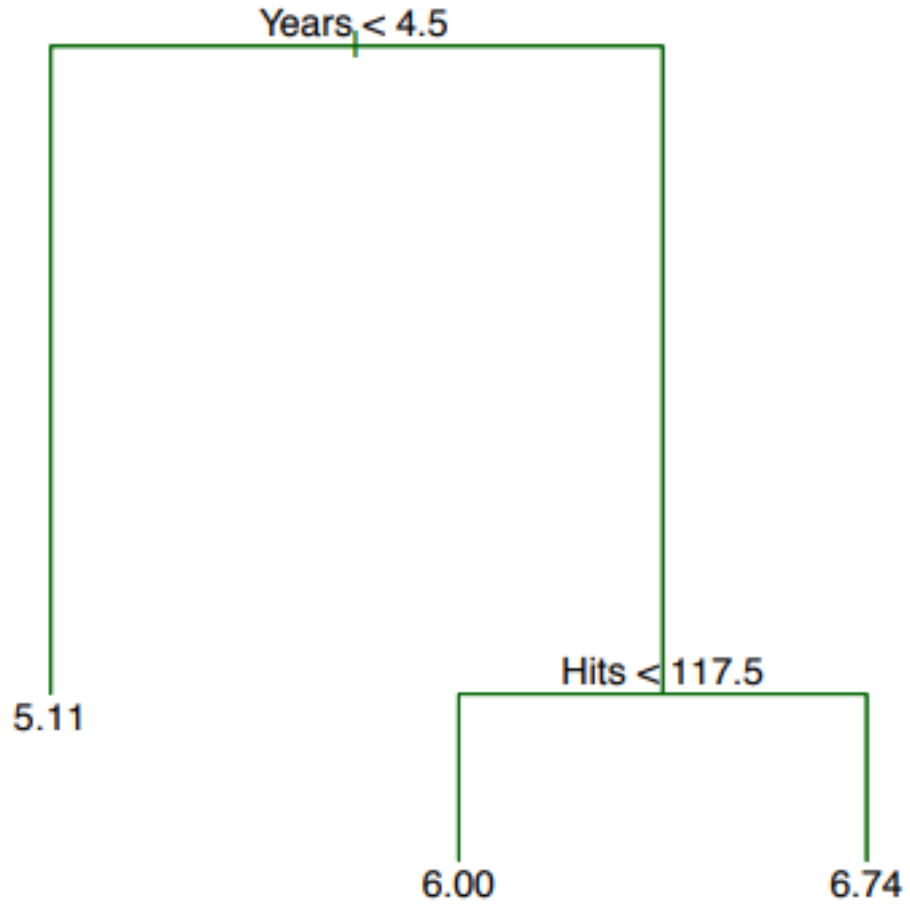
Baseball example continued

The minimum cross validation error occurs at a tree size of 3

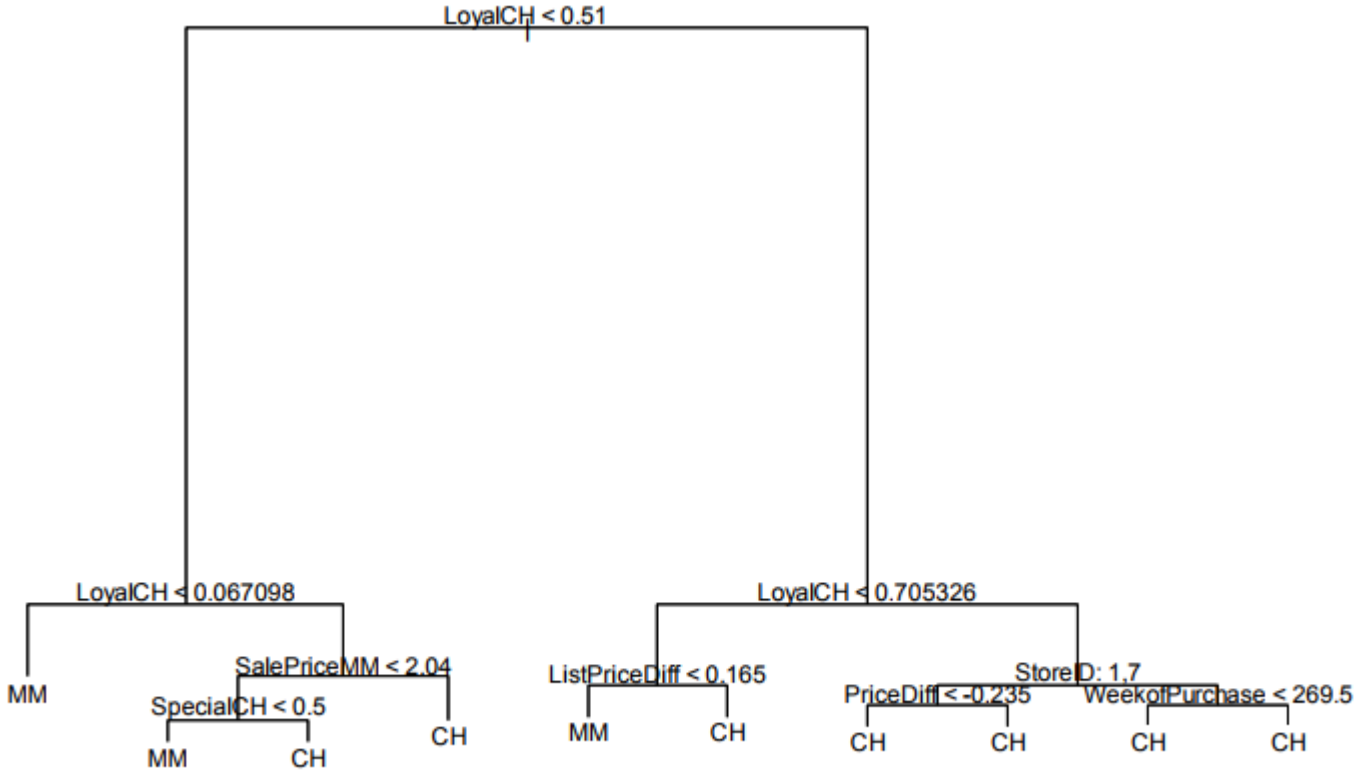


Baseball example continued

- Cross Validation indicated that the minimum MSE is when the tree size is three (i.e. the number of leaf nodes is 3)

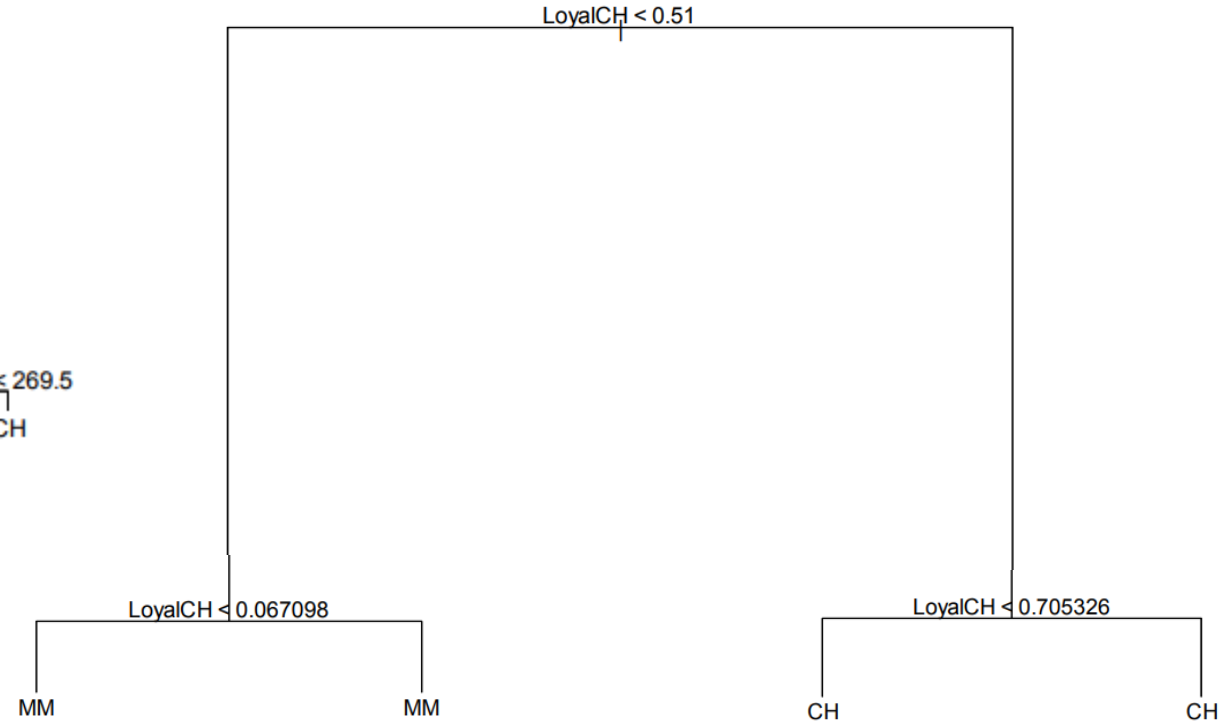


Example: Orange Juice Preference



Full Tree Training
Error Rate = 14.75%

Pruned Tree
CV Tree Error Rate = 22.5%



Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.
- The tree is grown (i.e. the splits are chosen) in exactly the same way as with a regression tree except that minimizing MSE no longer makes sense.
- There are several possible different criteria to use such as the “gini index” and “cross-entropy” but the easiest one to think about is to minimize the error rate.

Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the classification error rate. this is simply the fraction of the training observations in that region that do not belong to the most common class:

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

Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.

- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Gini index and Deviance

- The Gini index is defined by

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

a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.

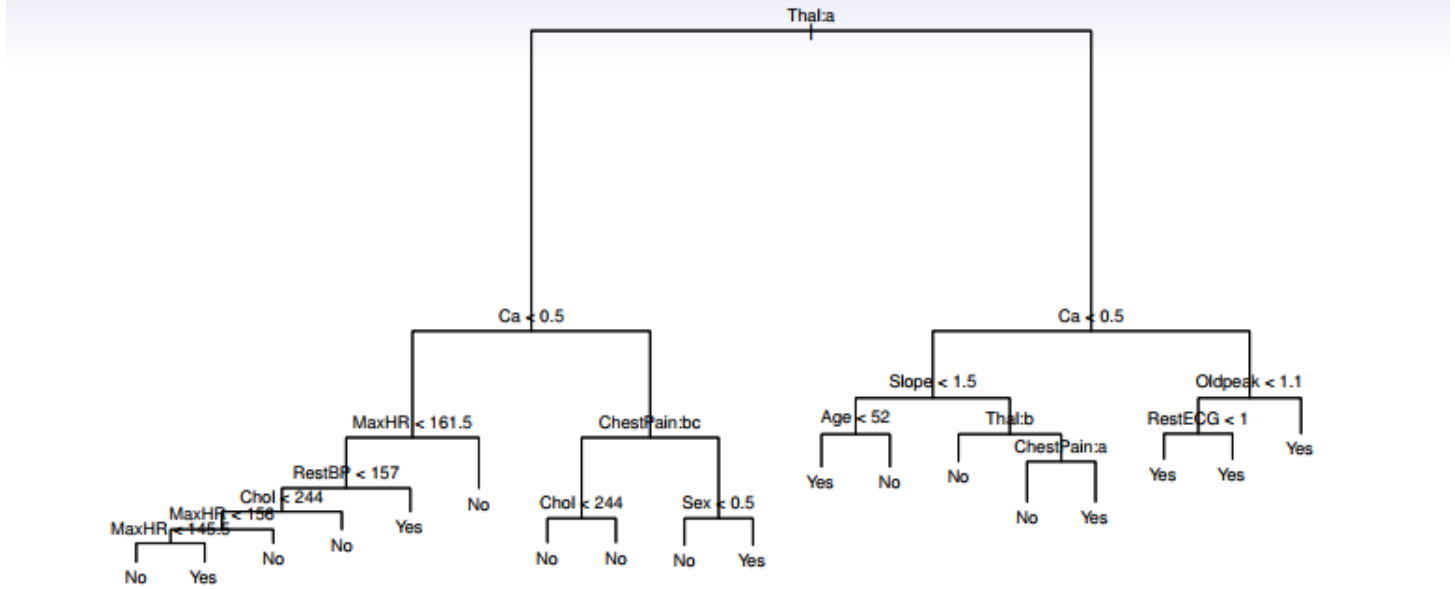
- For this reason the Gini index is referred to as a measure of node purity — a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is cross-entropy, given by

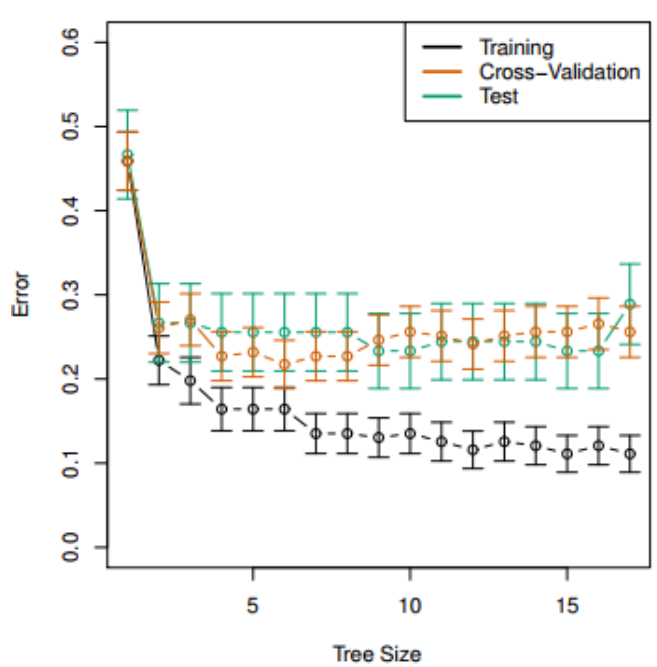
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

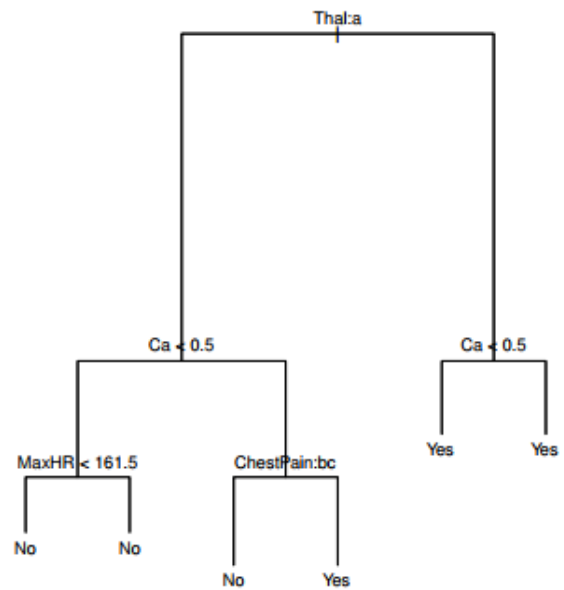
- It turns out that the Gini index and the cross-entropy are very similar numerically.

Example: heart data

- These data contain a binary outcome HD for 303 patients who presented with chest pain.
- An outcome value of Yes indicates the presence of heart disease based on an angiographic test, while No means no heart disease.
- There are 13 predictors including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Slope < 1.5
Oldpeak < 1.1
RestBP < 157
Chol < 244
Chol < 244
Sex < 0.5
Age < 52
Thal:b
RestECG < 1
Yes
MaxHR < 156
MaxHR < 145.5
ChestPain:a
No
No
No
No
No
No
No
No
No
No
No
No
No
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes


Training
Cross−Validation
Test
Error
0.0
0.1
0.2
0.3
0.4
0.5
0.6
5
10
15
Tree Size

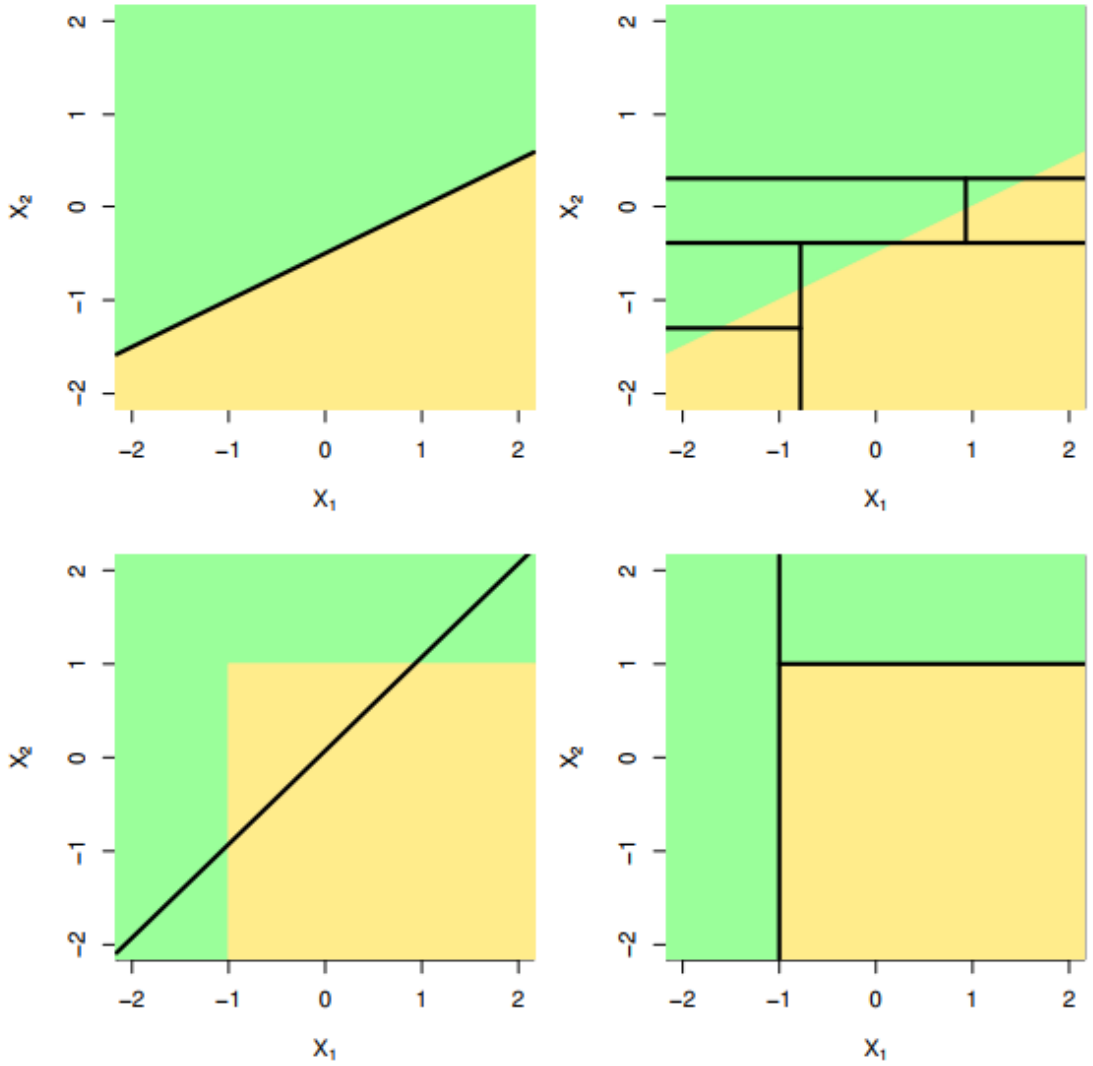

Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Yes
Yes
No
No
No
Yes

Trees vs. Linear Models

- Which model is better?
- If the relationship between the predictors and response is linear, then classical linear models such as linear regression would outperform regression trees
- On the other hand, if the relationship between the predictors is nonlinear, then decision trees would outperform classical approaches

Trees Versus Linear Models

- Top Row: the true decision boundary is linear;
  - Left: linear model (good)
  - Right: decision tree
- Bottom row: the true decision boundary is non-linear.
  - Left: linear model;
  - Right: decision tree (good)



Advantages and Disadvantages of Trees

- Pros:
  - Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
  - Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
  - Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
  - Trees 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 approaches seen in this book.
- However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

Aggregating Trees

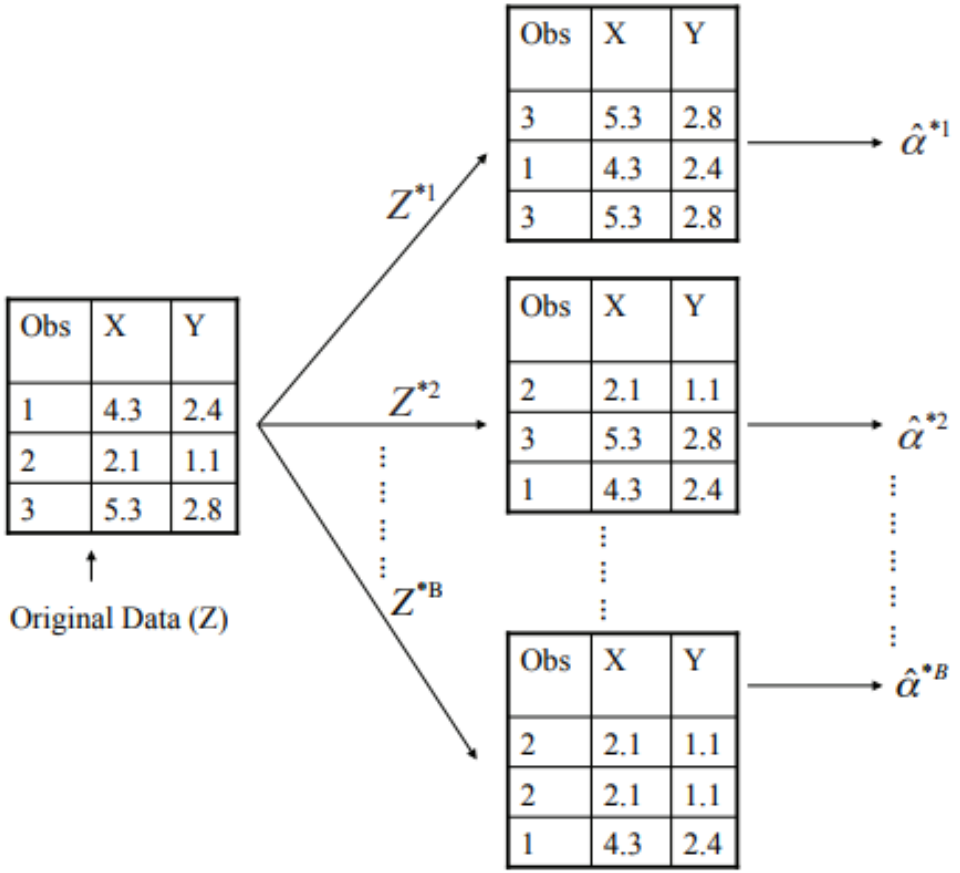
- Bagging
  - Bootstrapping
  - Bagging for Regression Trees
  - Bagging for Classification Trees
  - Out-of-Bag Error Estimation
  - Variable Importance: Relative Influence Plots
- Random Forests

Problem!

- Decision trees discussed earlier suffer from high variance!
  - If we randomly split the training data into 2 parts, and fit decision trees on both parts, the results could be quite different
- We would like to have models with low variance
- To solve this problem, we can use bagging (**b**ootstrap **agg**regat**ing**)

Bootstrapping is simple!

- Resampling of the observed dataset (and of equal size to the observed dataset), each of which is obtained by random sampling with replacement from the original dataset.



Bagging

- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of *n* independent observations $Z_1, \ldots, Z_n$, each with variance σ^2 , the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n
- In other words, averaging a set of observations reduces variance. Of course, this is not practical because we generally do not have access to multiple training sets.

How does bagging work?

- Generate B different bootstrapped training datasets
- Train the statistical learning method on each of the B training datasets, and obtain the prediction
- For prediction:
  - Regression: average all predictions from all B trees
  - Classification: majority vote among all B trees

Bagging— continued

- We can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x. We then average all the predictions to obtain

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

- This is called bagging.

Bagging for Regression Trees

- Construct B regression trees using B bootstrapped training datasets
- Average the resulting predictions
- Note: These trees are not pruned, so each individual tree has high variance but low bias. Averaging these trees reduces variance, and thus we end up lowering both variance and bias

Bagging classification trees

- Construct B regression trees using B bootstrapped training datasets
- For prediction, there are two approaches:
    1. Record the class that each bootstrapped data set predicts and provide an overall prediction to the most commonly occurring one (majority vote).
    2. If our classifier produces probability estimates we can just average the probabilities and then predict to the class with the highest probability.
- Both methods work well.

Out-of-bag estimates

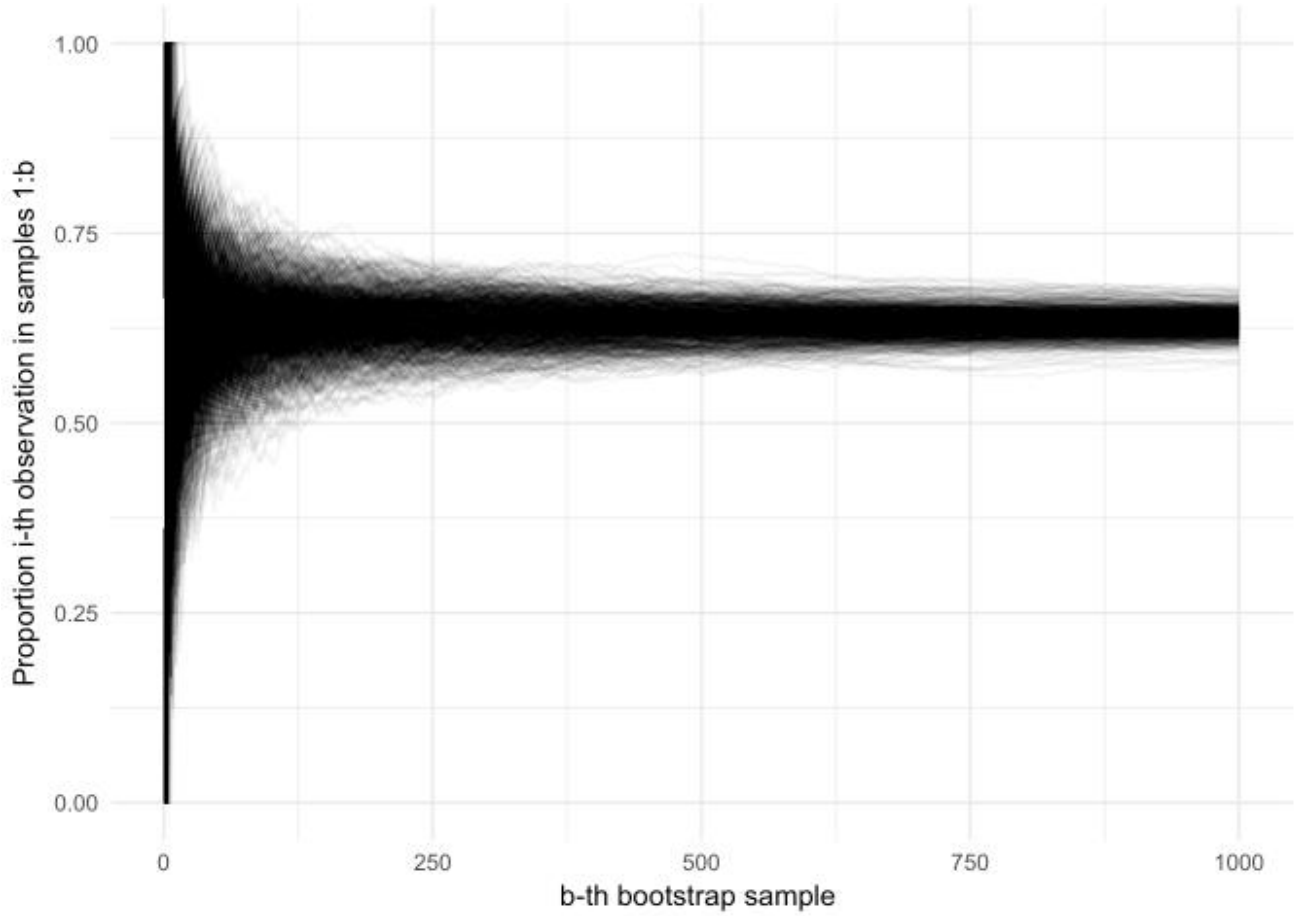
- Fortunately using a bagged approach also allows us to avoid using any type of resampling method to calculate the test MSE or error rate.
- This is because we have a natural test set as a result of the bootstrapping process.
- Recall that in a bootstrap sampling process, we **sample with replacement**. This means that in some bootstrap samples, an observation may never be drawn.
- On average, each bagged tree uses approximately two-thirds of the original observations.
- observations not appearing in a given bag are considered **out-of-bag observations** (OOB).

Out-of-bag error estimate

1. First we generate bagged predictions for each observation i using only its OOB estimates

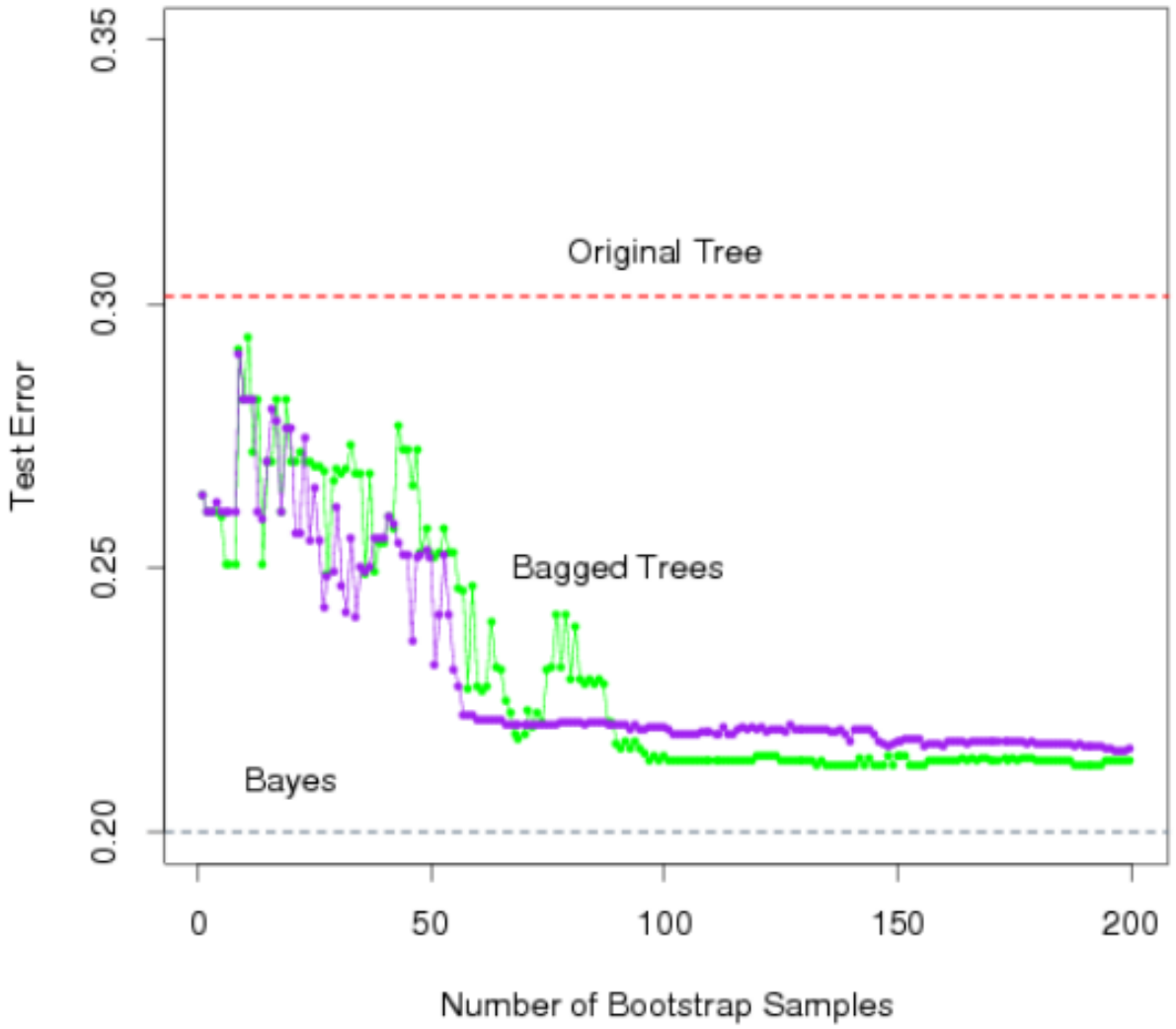
2. average across all i observations to get the OOB error estimate.

This is a valid estimate of the test error rate/MSE because it only uses observations that were not part of the training observations for a given bag b

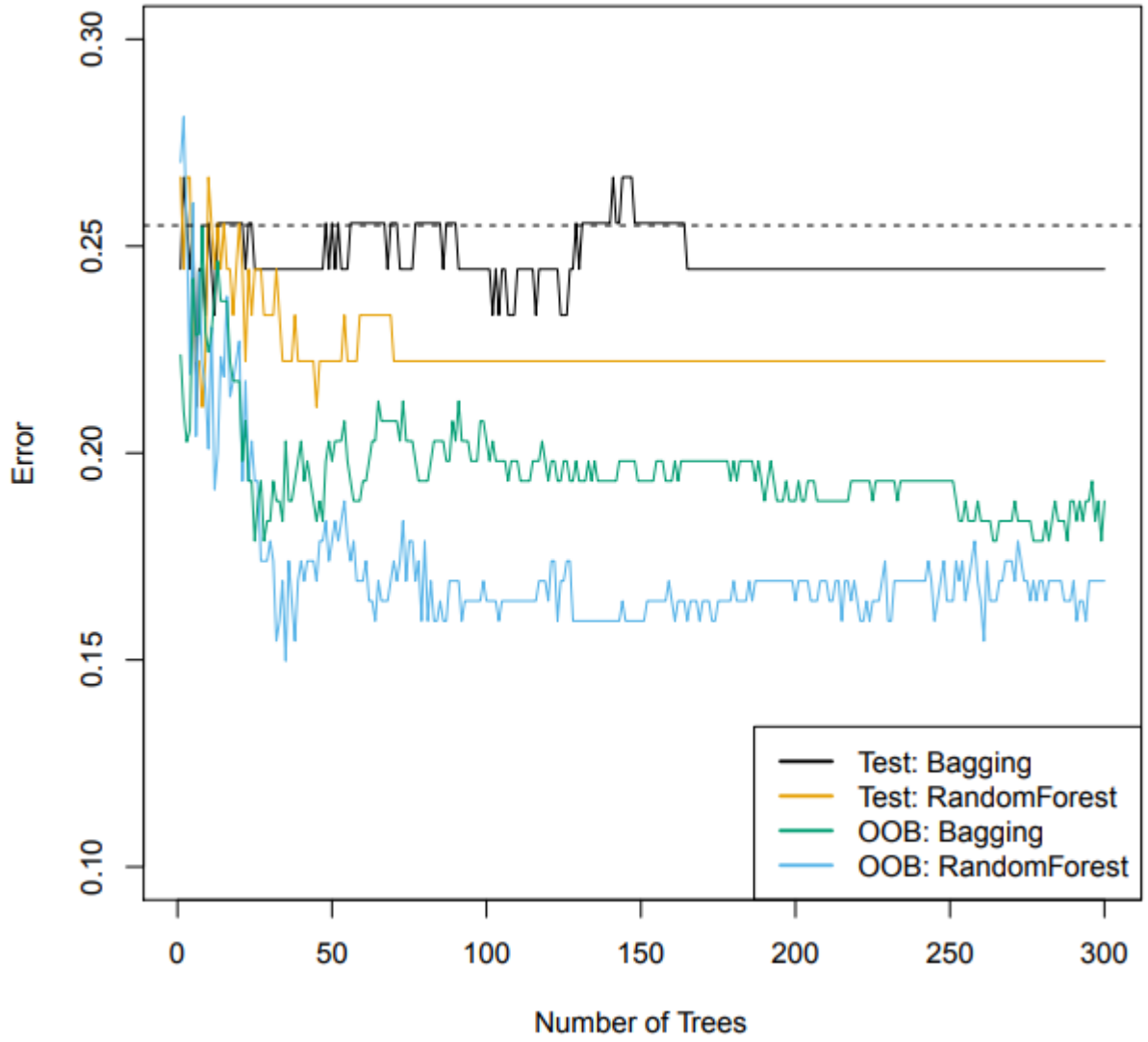


A Comparison of Error Rates

- Here the green line represents a simple majority vote approach
- The purple line corresponds to averaging the probability estimates.
- Both do far better than a single tree (dashed red) and get close to the Bayes error rate (dashed grey).



Bagging the heart data



Details of previous figure

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 considerably lower

Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the ith observation, which we average.
- This estimate is essentially the LOO cross-validation error for bagging, if B is large.

Variable Importance Measure

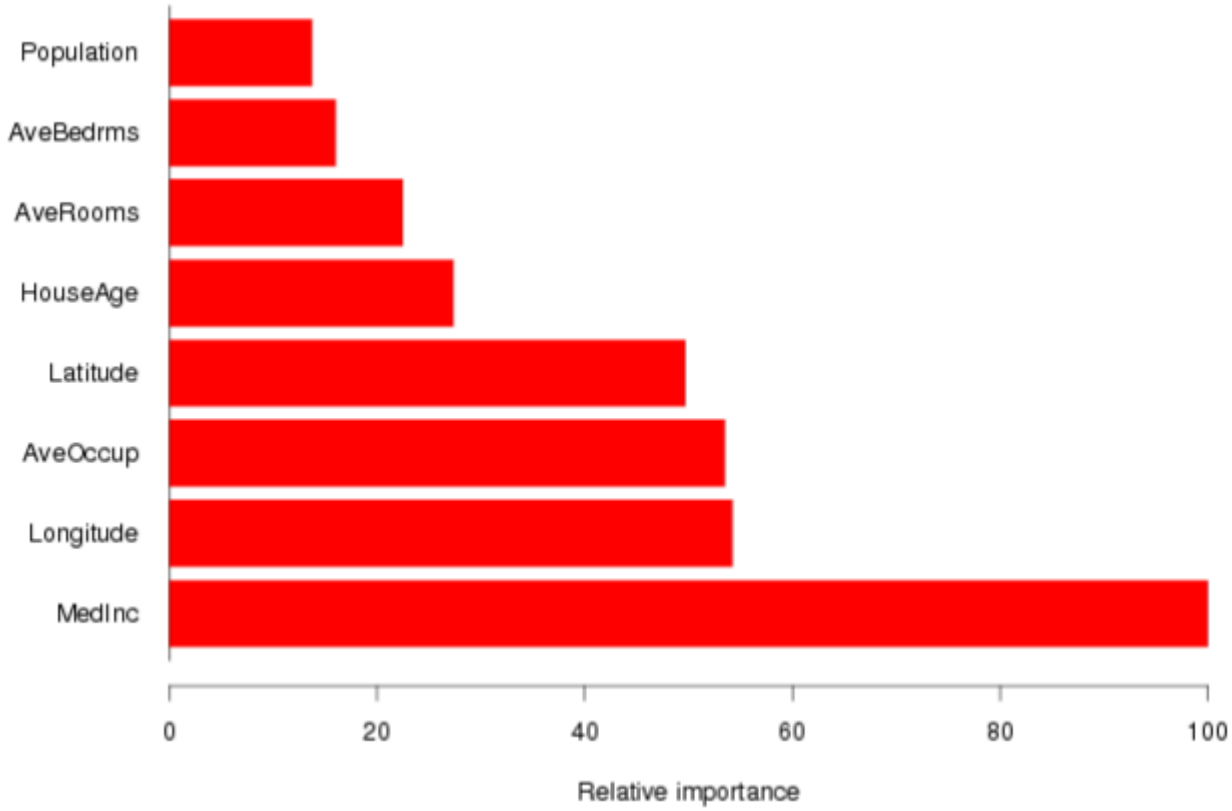
- Bagging typically improves the accuracy over prediction using a single tree, but it is now hard to interpret the model!
- We have hundreds of trees, and it is no longer clear which variables are most important to the procedure
- Thus bagging improves prediction accuracy at the expense of interpretability
- But, we can still get an overall summary of the importance of each predictor using Relative Influence Plots

Relative Influence Plots

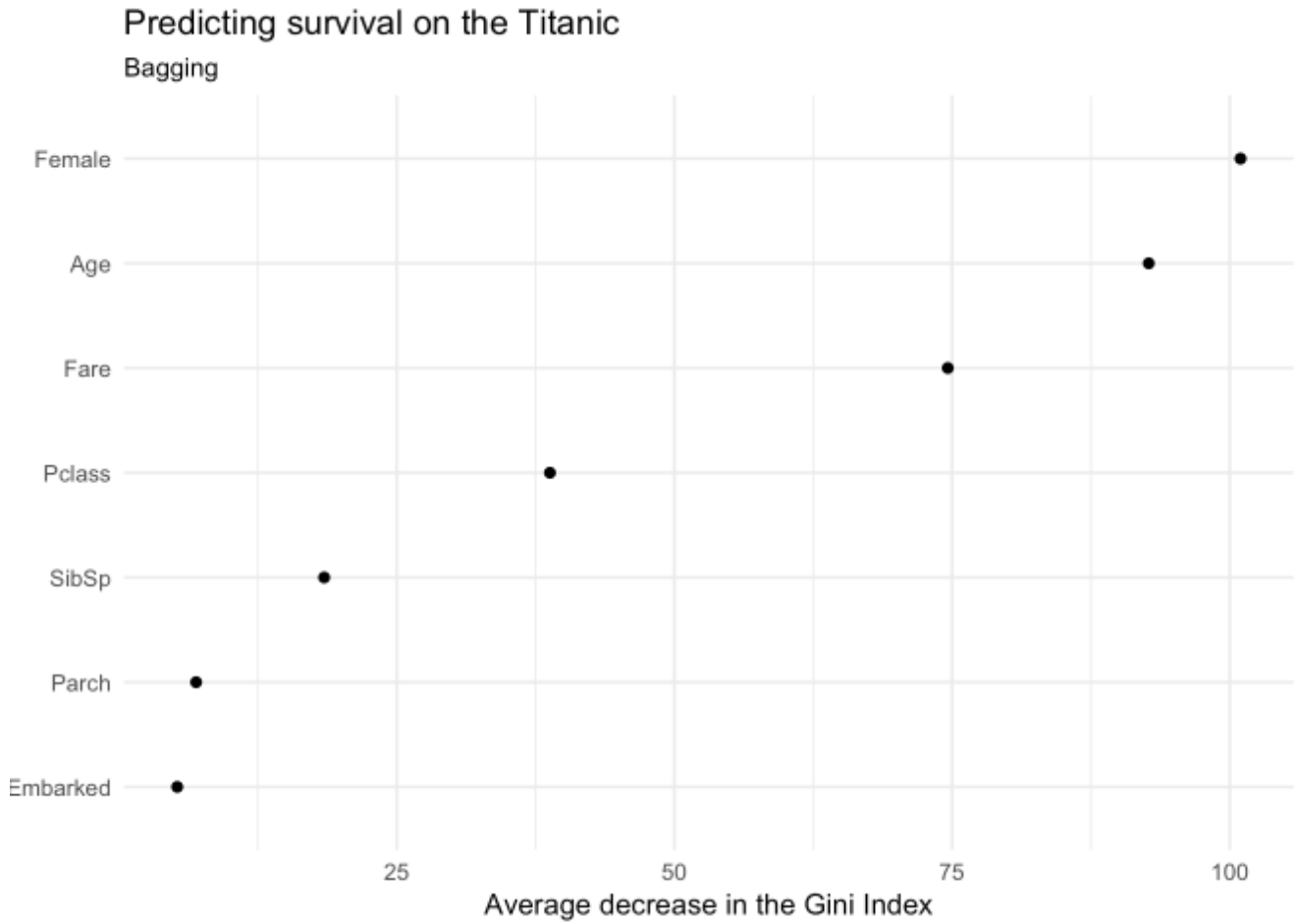
- How do we decide which variables are most useful in predicting the response?
  - We can compute something called relative influence plots.
  - These plots give a score for each variable.
  - These scores represents the decrease in MSE when splitting on a particular variable
  - A number close to zero indicates the variable is not important and could be dropped.
  - The larger the score the more influence the variable has.

Example: Housing Data

- Median Income is by far the most important variable.
- Longitude, Latitude and Average occupancy are the next most important



Predicting survival on Titanic



For classification trees, larger values is better. Thus, gender, age, and fare are the most important predictors

Random Forests

- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- How does it work?
  - Build a number of decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
  - A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ -- that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).

Why are we considering a random sample of m predictors instead of all p predictors for splitting?

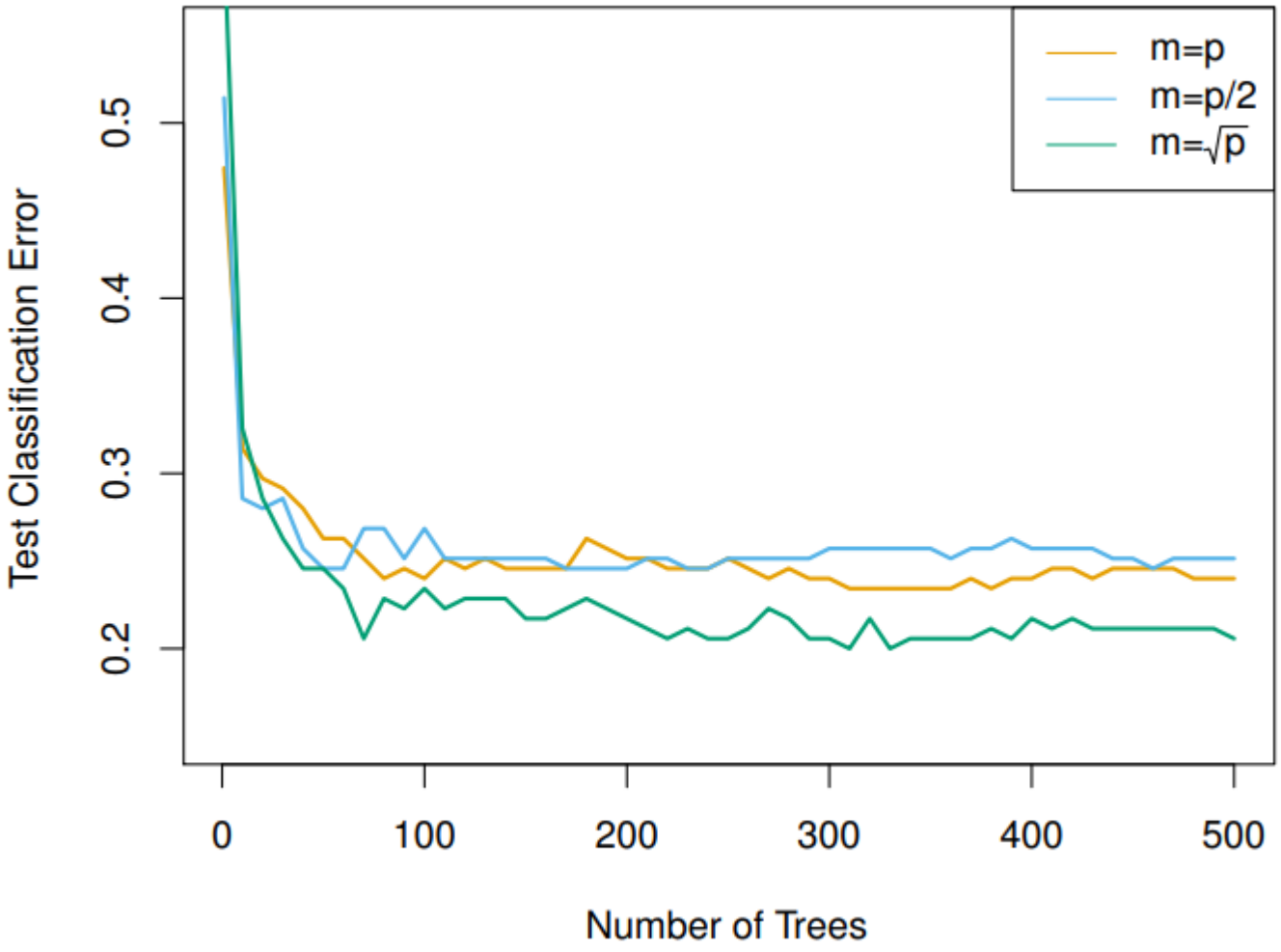
- Suppose that we have a very strong predictor in the data set along with a number of other moderately strong predictor, then in the collection of bagged trees, most or all of them will use the very strong predictor for the first split!
- All bagged trees will look similar. Hence all the predictions from the bagged trees will be highly correlated
- Averaging many highly correlated quantities does not lead to a large variance reduction, and thus random forests “de-correlates” the bagged trees leading to more reduction in variance

Example: gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Results: gene expression data

- Notice when random forests are built using $m = p$, then this amounts simply to bagging.



Details of previous figure

- Results from random forests for the fifteen-class gene expression data set with p = 500 predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees

Boosting algorithm 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:
   1. Fit a tree $\hat{f}^b$ with d splits (d + 1 terminal nodes) to the training data (X, r).
   2. Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   3. Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Output the boosted model,

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

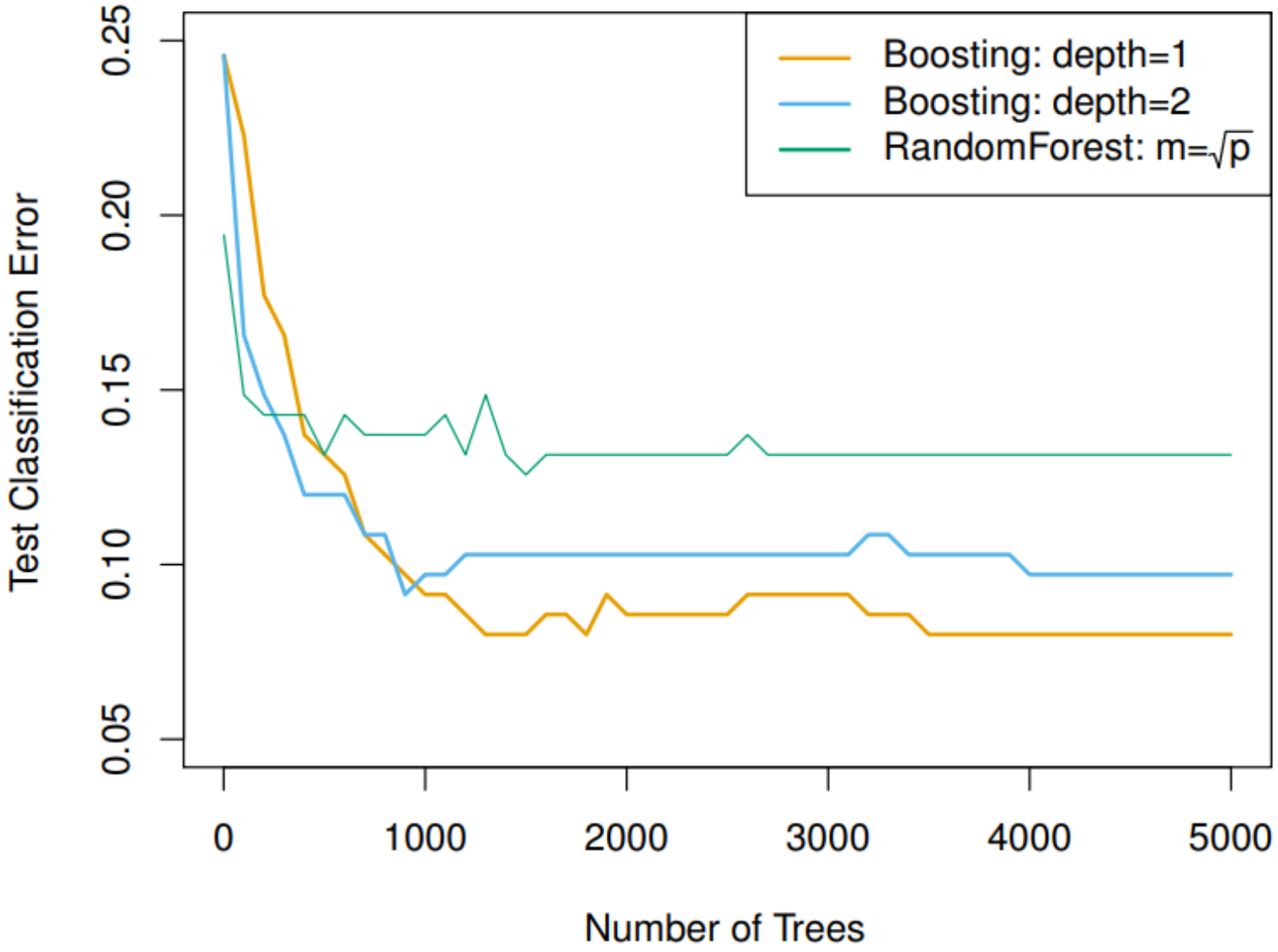
What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Boosting for classification

- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. We will not go into detail here, nor do we in the text book.
- The R package gbm (gradient boosted models) handles a variety of regression and classification problems.

Gene expression data continued



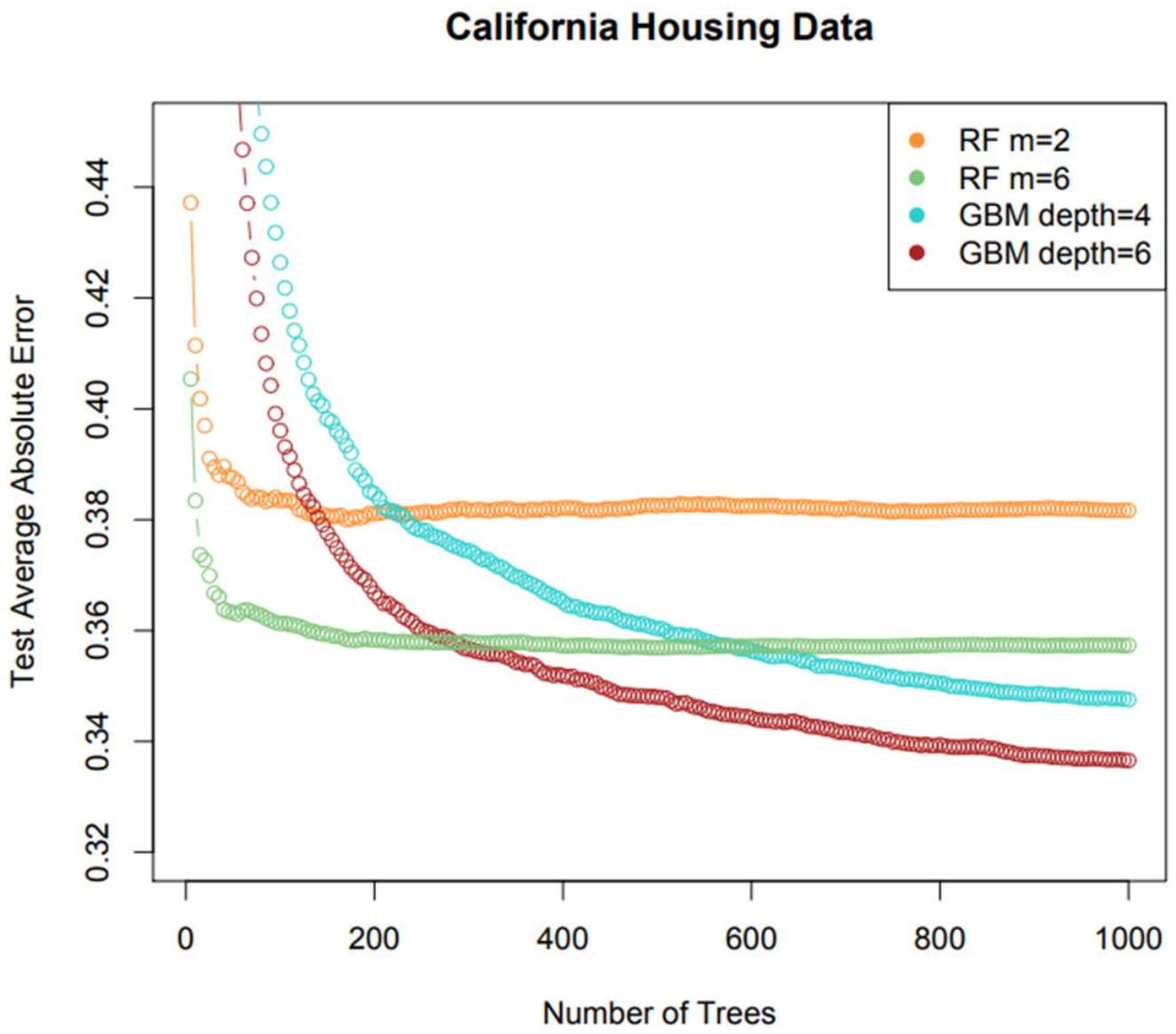
Details of previous figure

- Results from performing boosting and random forests on the fifteen-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%.

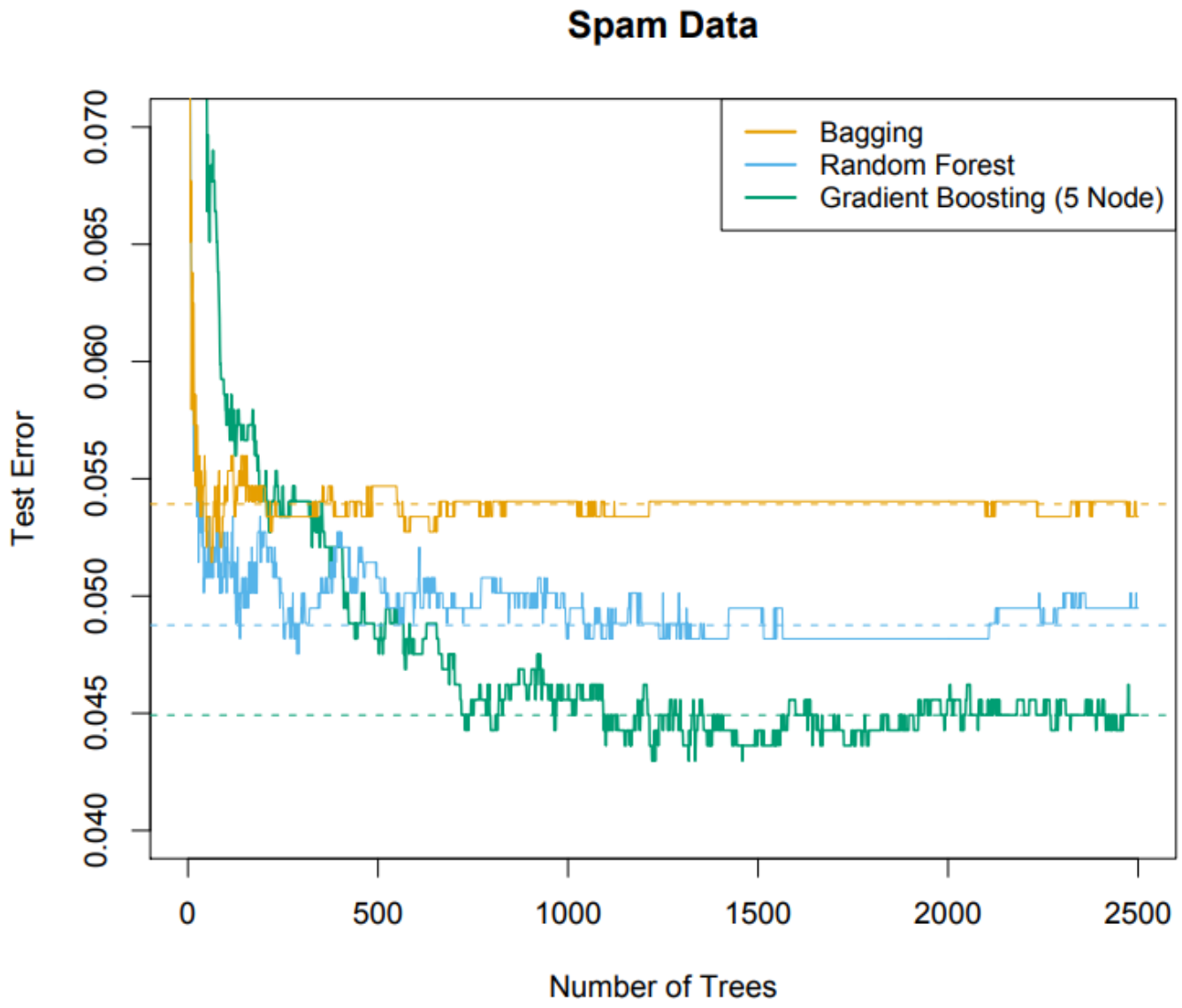
Tuning parameters for boosting

- The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
- The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
- The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often d = 1 works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Another regression example



Another classification example

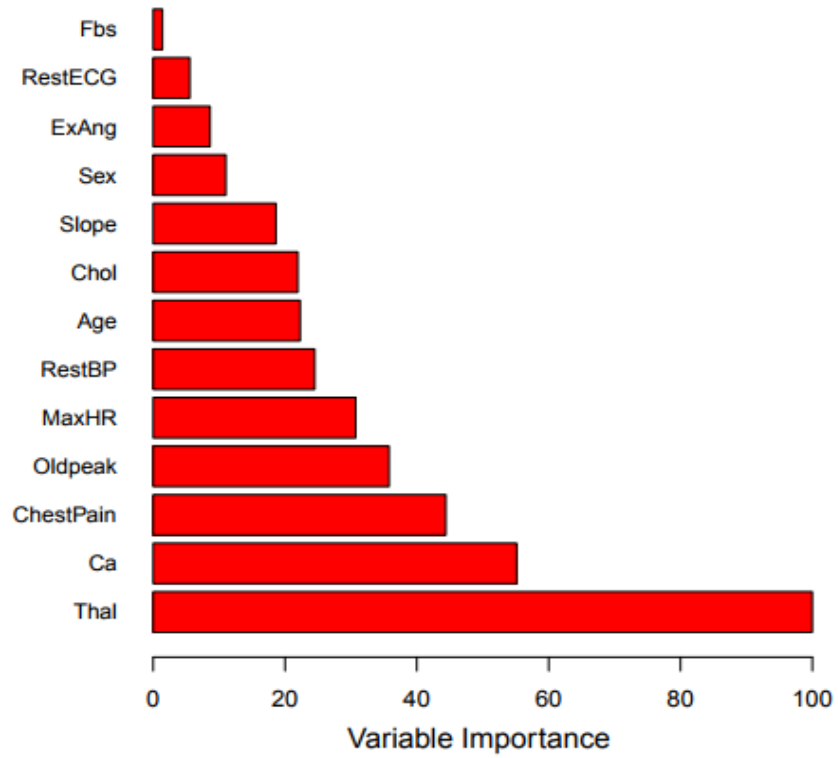


Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.

Variable importance plot

for the Heart data



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 and boosting 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.
- The latter two methods— random forests and boosting— are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Example from Ben Gorman's blog at Kaggle.com

- We'll start with a simple example. We want to predict a person's age based on whether they play video games, enjoy gardening, and their preference on wearing hats. Our objective is to minimize squared error. We have these nine training samples to build our model.

Data

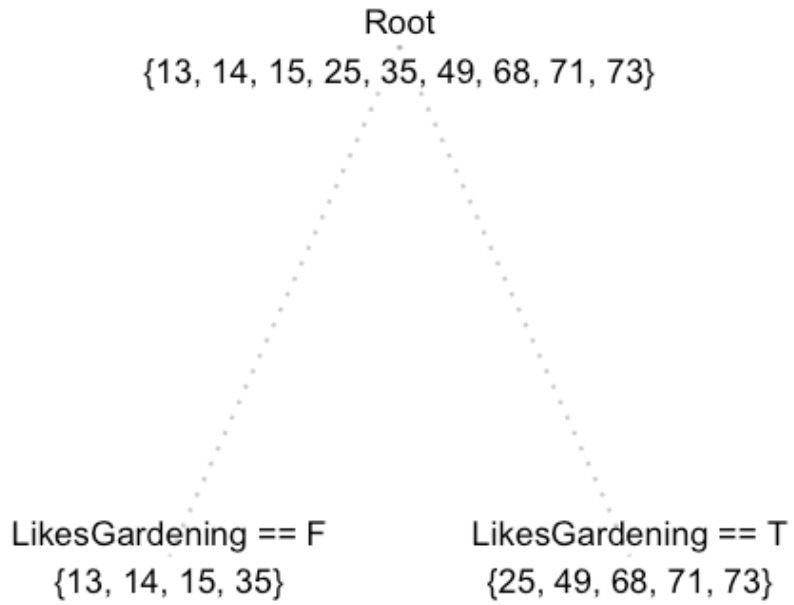
PersonID	Age	LikesGardening	PlaysVideoGames	LikesHats
1	13	FALSE	TRUE	TRUE
2	14	FALSE	TRUE	FALSE
3	15	FALSE	TRUE	FALSE
4	25	TRUE	TRUE	TRUE
5	35	FALSE	TRUE	TRUE
6	49	TRUE	FALSE	FALSE
7	68	TRUE	TRUE	TRUE
8	71	TRUE	FALSE	FALSE
9	73	TRUE	FALSE	TRUE

Intuitively, we might expect
– The people who like gardening are probably older
– The people who like video games are probably younger
– *LikesHats* is probably just random noise

Gradient Boosting Continued

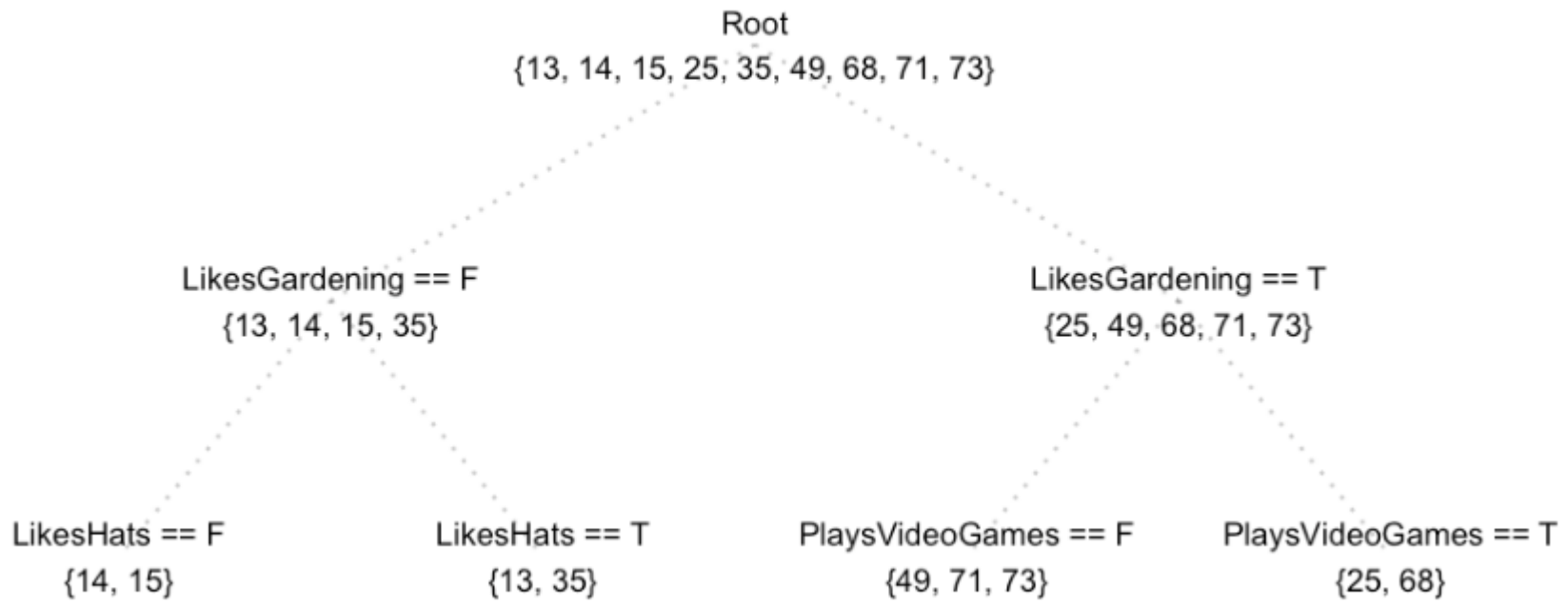
Feature	FALSE	TRUE
LikesGardening	{13, 14, 15, 35}	{25, 49, 68, 71, 73}
PlaysVideoGames	{49, 71, 73}	{13, 14, 15, 25, 35, 68}
LikesHats	{14, 15, 49, 71}	{13, 25, 35, 68, 73}

Now let's model the data with a regression tree. To start, we'll require that terminal nodes have at least three samples. With this in mind, the regression tree will make its first and last split on LikesGardening.



This is nice, but it's missing valuable information from the feature LikesVideoGames. Let's try letting terminal nodes have 2 samples.

Overfit Tree

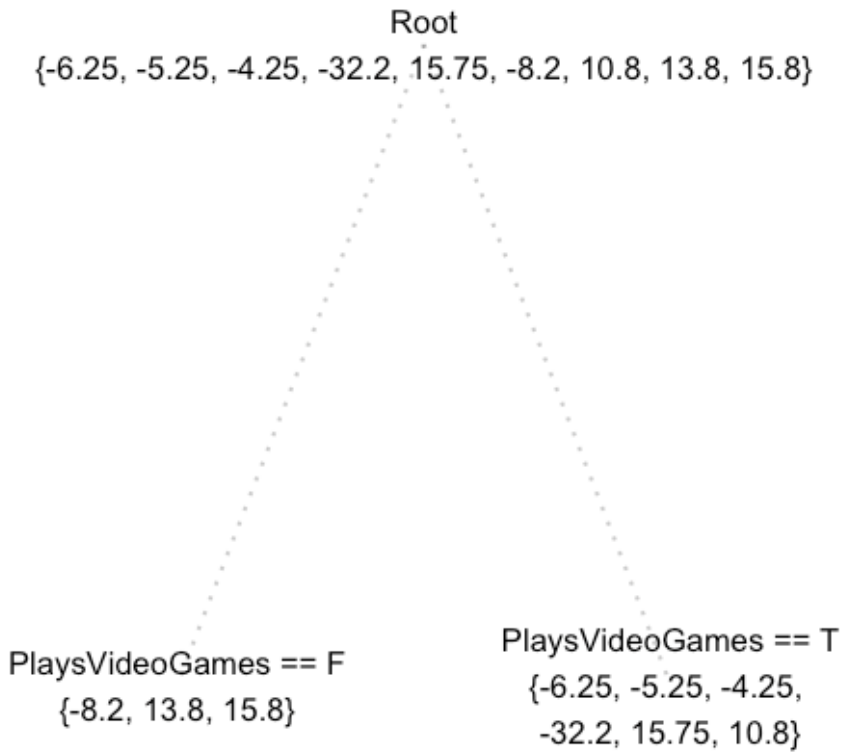


Here we pick up some information from *PlaysVideoGames* but we also pick up information from *LikesHats* – a good indication that we're overfitting and our tree is splitting random noise.

Here in lies the drawback to using a single decision/regression tree – **it fails to include predictive power from multiple, overlapping regions of the feature space**. Suppose we measure the training errors from our first tree.

Fitting a second regression tree to the residuals of the first tree.

PersonID	Age	Tree1 Prediction	Tree1 Residual
1	13	19.25	-6.25
2	14	19.25	-5.25
3	15	19.25	-4.25
4	25	57.2	-32.2
5	35	19.25	15.75
6	49	57.2	-8.2
7	68	57.2	10.8
8	71	57.2	13.8
9	73	57.2	15.8



Notice that this tree does **not** include *LikesHats* even though **our overfitted regression tree above did**. The reason is because this regression tree is able to consider LikesHats and PlaysVideoGames with respect to all the training samples, contrary to our overfit regression tree which only considered each feature inside a small region of the input space, thus allowing random noise to select *LikesHats* as a splitting feature.

Combining trees

- Now we can improve the predictions from our first tree by adding the "error-correcting" predictions from this tree.

PersonID	Age	Tree1 Prediction	Tree1 Residual	Tree2 Prediction	Combined Prediction	Final Residual
1	13	19.25	-6.25	-3.567	15.68	2.683
2	14	19.25	-5.25	-3.567	15.68	1.683
3	15	19.25	-4.25	-3.567	15.68	0.6833
4	25	57.2	-32.2	-3.567	53.63	28.63
5	35	19.25	15.75	-3.567	15.68	-19.32
6	49	57.2	-8.2	7.133	64.33	15.33
7	68	57.2	10.8	-3.567	53.63	-14.37
8	71	57.2	13.8	7.133	64.33	-6.667
9	73	57.2	15.8	7.133	64.33	-8.667

Tree1 SSE	Combined SSE
1994	1765

Gradient Boosting – Draft 1

- Inspired by the idea above, we create our first (naive) formalization of gradient boosting. In pseudocode

  1. Fit a model to the data, $F_1(x) = y$
  2. Fit a model to the residuals, $h_1(x) = y - F_1(x)$
  3. Create a new model, $F_2(x) = F_1(x) + h_1(x)$

- It's not hard to see how we can generalize this idea by inserting more models that correct the errors of the previous model. Specifically,

  $$F(x) = F_1(x) \mapsto F_2(x) = F_1(x) + h_1(x) \ldots \mapsto F_M(x) = F_{M-1}(x) + h_{M-1}(x)$$

  where $F_1(x)$ is an initial model fit to y

  Since we initialize the procedure by fitting $F_1(x)$, our task at each step is to find $h_m(x) = y - F_m(x)$.

- Nothing in our definition requires it to be a tree-based model. This is one of the broader concepts and advantages to gradient boosting. It's really just a framework for iteratively improving any weak learner. In practice however, h_m is almost always a tree based learner, so for now it's fine to interpret h_m as a regression tree like the one in our example.

Gradient Boosting – Draft 2

- Now we'll tweak our model to conform to most gradient boosting implementations – we'll initialize the model with a single prediction value. Since our task (for now) is to minimize squared error, we'll initialize F with the mean of the training target values.

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma) = \arg\min_{\gamma} \sum_{i=1}^{n} (\gamma - y_i)^2 = \frac{1}{n} \sum_{i=1}^{n} y_i.$$

- Then we can define each subsequent F_m recursively, just like before

$$F_{m+1}(x) = F_m(x) + h_m(x) = y, \text{ for } m \geq 0$$

- Where h_m comes from a class of base learners (e.g. regression trees).

Gradient Boosting – Draft 3

- Up until now we've been building a model that minimizes squared error, but what if we wanted to minimize absolute error?
  We *could* alter our base model (regression tree) to minimize absolute error, but this has a couple drawbacks.
  1. Depending on the size of the data this could be very computationally expensive. (Each considered split would need to search for a median.)
  2. It ruins our "plug-in" system. We'd only be able to plug in weak learns that support the objective function(s) we want to use.
- Instead we're going to do something much niftier. Recall our example problem. To determine F_0, we start by choosing a minimizer for absolute error. This'll be median(y)= 35. Now we can measure the residuals, $y - F_0$.

Gradient Boosting – Draft 3

PersonID	Age	F0	Residual0
1	13	35	-22
2	14	35	-21
3	15	35	-20
4	25	35	-10
5	35	35	0
6	49	35	14
7	68	35	33
8	71	35	36
9	73	35	38

Consider the first and fourth training samples. They have F_0residuals of -22 and -10 respectively. Now suppose we're able to make each prediction 1 unit closer to its target. Respective squared error reductions would be 43 and 19, while respective absolute error reductions would be 1 and 1. So a regression tree, which by default minimizes squared error, will focus heavily on reducing the residual of the first training sample. But if we want to minimize absolute error, moving each prediction one unit closer to the target produces an equal reduction in the cost function. With this in mind, suppose that instead of training h_0 on the residuals of F_0, we instead train h_0 on the gradient of the loss function, $L\big(y, F_0(x)\big)$ with respect to the prediction values produced by $F_0(x)$.

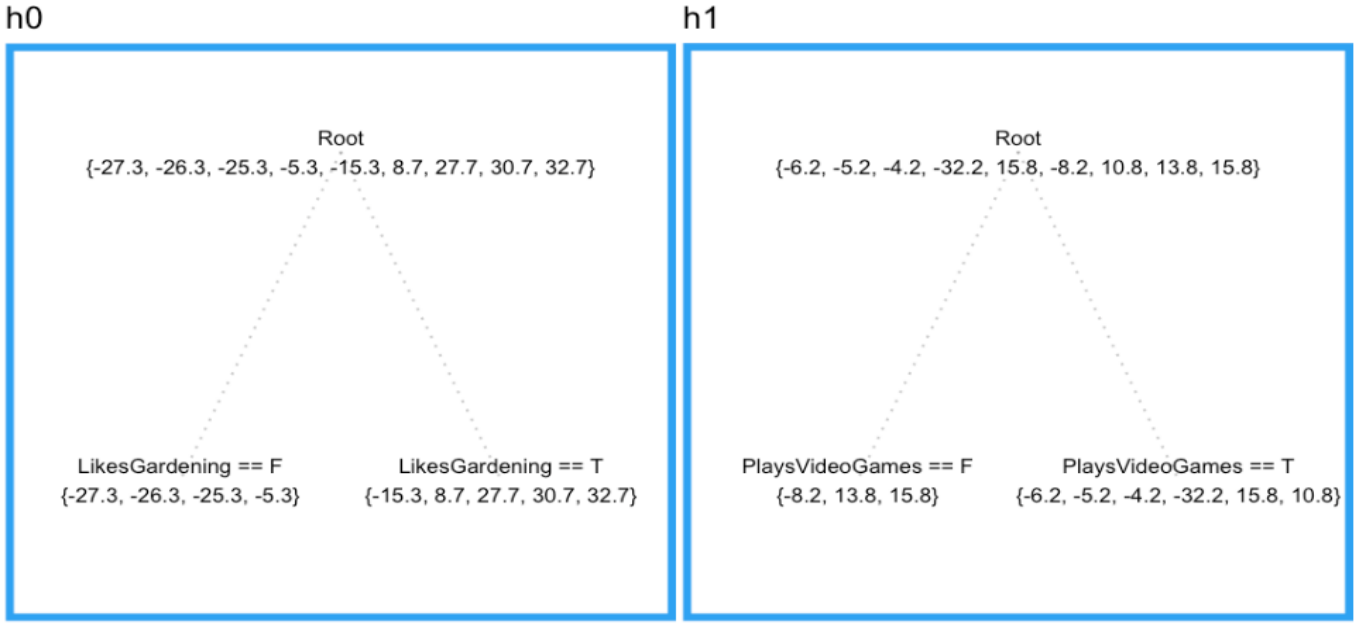
Gradient Boosting – Draft 3 cont.

- Essentially, we'll train h_0 on the cost reduction for each sample if the predicted value were to become one unit closer to the observed value. In the case of absolute error, h_m will simply consider the sign of every F_m residual (as apposed to squared error which would consider the magnitude of every residual). After samples in h_m, are grouped into leaves, an average gradient can be calculated and then scaled by some factor, γ so that $F_m + \gamma h_m$ minimizes the loss function for the samples in each leaf. (Note that in practice, a different factor is chosen for each leaf.)
- In case you want to check your understanding so far, our current gradient boosting applied to our sample problem for both squared error and absolute error objectives yields the following results.

Squared Error

Squared Error

Age	F0	PseudoResidual0	h0	gamma0	F1	PseudoResidual1	h1	gamma1	F2
13	40.33	-27.33	-21.08	1	19.25	-6.25	-3.567	1	15.68
14	40.33	-26.33	-21.08	1	19.25	-5.25	-3.567	1	15.68
15	40.33	-25.33	-21.08	1	19.25	-4.25	-3.567	1	15.68
25	40.33	-15.33	16.87	1	57.2	-32.2	-3.567	1	53.63
35	40.33	-5.333	-21.08	1	19.25	15.75	-3.567	1	15.68
49	40.33	8.667	16.87	1	57.2	-8.2	7.133	1	64.33
68	40.33	27.67	16.87	1	57.2	10.8	-3.567	1	53.63
71	40.33	30.67	16.87	1	57.2	13.8	7.133	1	64.33
73	40.33	32.67	16.87	1	57.2	15.8	7.133	1	64.33

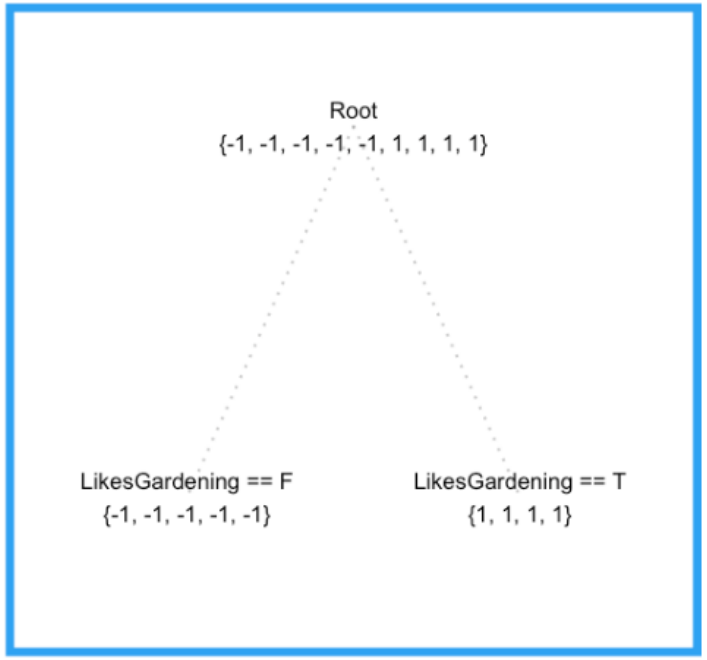


Absolute Error

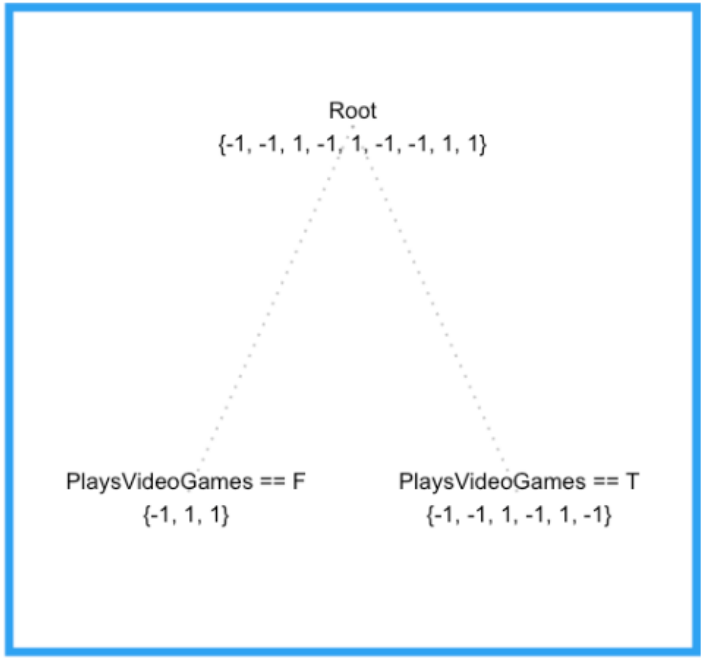
Absolute Error

Age	F0	PseudoResidual0	h0	gamma0	F1	PseudoResidual1	h1	gamma1	F2
13	35	-1	-1	20.5	14.5	-1	-0.3333	0.75	14.25
14	35	-1	-1	20.5	14.5	-1	-0.3333	0.75	14.25
15	35	-1	-1	20.5	14.5	1	-0.3333	0.75	14.25
25	35	-1	0.6	55	68	-1	-0.3333	0.75	67.75
35	35	-1	-1	20.5	14.5	1	-0.3333	0.75	14.25
49	35	1	0.6	55	68	-1	0.3333	9	71
68	35	1	0.6	55	68	-1	-0.3333	0.75	67.75
71	35	1	0.6	55	68	1	0.3333	9	71
73	35	1	0.6	55	68	1	0.3333	9	71

h0



h1



Gradient Boosting – Draft 4

- Here we introduce something called shrinkage. The concept is fairly simple. For each gradient step, the step magnitude is multiplied by a factor between 0 and 1 called a learning rate. In other words, each gradient step is *shrunken* by some factor. The current Wikipedia excerpt on shrinkage doesn't mention why shrinkage is effective – it just says that shrinkage appears to be empirically effective. My personal take is that it causes sample-predictions to *slowly* converge toward observed values. As this slow convergence occurs, samples that get closer to their target end up being grouped together into larger and larger leaves (due to fixed tree size parameters), resulting in a natural regularization effect.
- For more info see Ben Gorman's blog at Kaggle.com