



UTN – FRVM

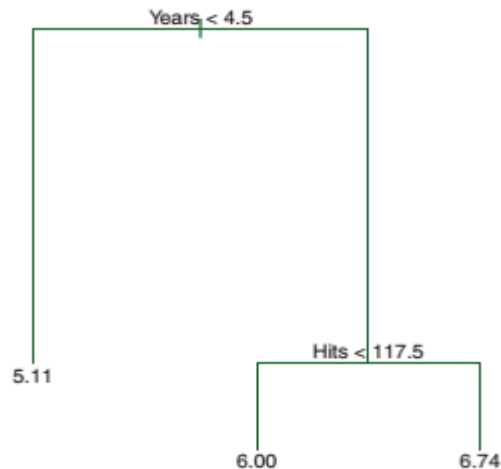
Agenda



- Decision Trees
 - Regression
 - Classification
- Bagging
- Random Forests

Decision Trees

- Decision trees can be applied to both regression and classification problems.



Equation: $\log(Y) = a + bX$ (From exponentiating both sides of the equation: $Y = e^a e^{bX}$)

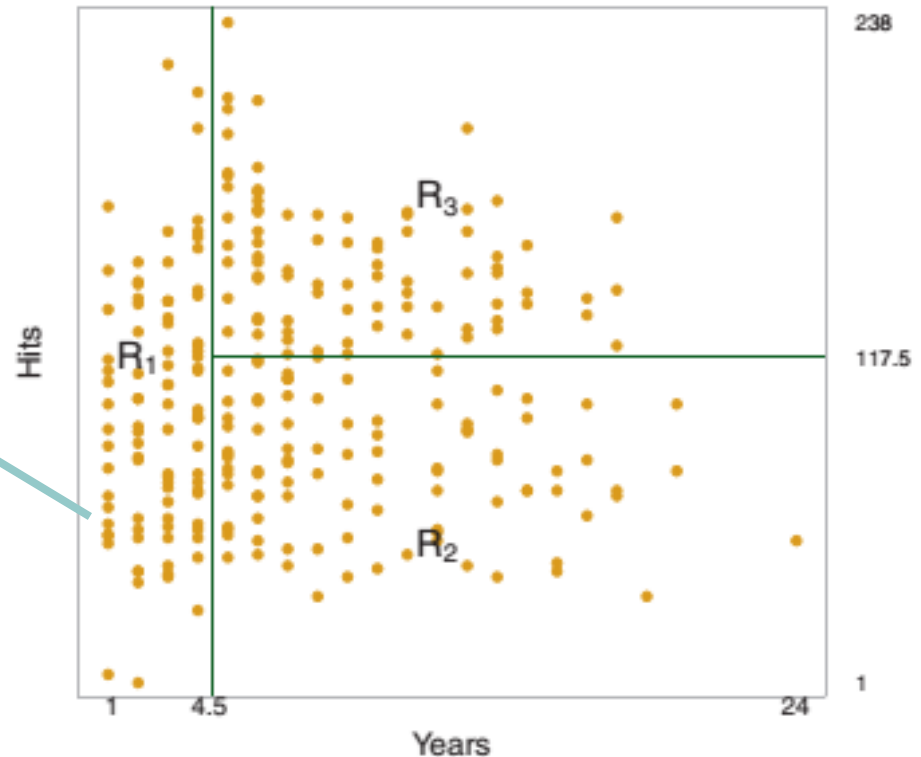
Meaning: A unit increase in X is associated with an average of $b\%$ increase in Y .

- 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 in major leagues and the number of hits that he made in the previous year.
- 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 $X_j \geq t_k$.
- 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.

Regression Trees

- We first remove observations that are missing **Salary** values, and log-transform **Salary** so that its distribution has more of a typical bell-shape. (Recall that Salary is measured in thousands of dollars.)

For such players, the mean log salary is 5.107, and so we make a prediction of $e^{5.107}$ thousands of dollars, i.e. \$165,174, for these players.



Regression Trees

- The tree stratifies or segments the players into three regions of predictor space: players who have played for four or fewer years, players who have played for five or more years and who made fewer than 118 hits last year, and players who have played for five or more years and who made at least 118 hits last year.
- These three regions can be written as $R1 = \{X / \text{Years} < 4.5\}$, $R2 = \{X / \text{Years} \geq 4.5, \text{Hits} < 117.5\}$, and $R3 = \{X / \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$.

$$\$1,000 \times e^{5.107} = \$165,174$$

$$\$1,000 \times e^{5.999} = \$402,834$$

$$\$1,000 \times e^{6.740} = \$845,346$$



Regression Trees

- 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.
- It has advantages over other types of regression models: it is easier to interpret, and has a nice graphical representation.



Prediction via Stratification of the Feature Space

- ***Prediction via Stratification of the Feature Space***
- 1. We divide the predictor space—that is, the set of possible values for $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_p$ —into J distinct and non-overlapping regions, $\mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_J$.
- 2. For every observation that falls into the region \mathbf{R}_j , we make the same prediction, which is simply the mean of the response values for the training observations in \mathbf{R}_j .

Constructing Regions

- How do we construct the regions R_1, \dots, R_J ?
- 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, \dots, R_J that minimize the RSS, given by

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

Mean response for the training observations within the j th box.



Recursive binary splitting

- 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 ***recursive binary splitting*** approach is *top-down* because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.



Recursive binary splitting

- 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.
- In order to perform recursive binary splitting, we first select the predictor X and the cutpoint s such that splitting the predictor space into the regions $\{X/ X_j < s\}$ and $\{X/ X_j \geq s\}$ leads to the greatest possible reduction in RSS.

Recursive binary splitting

- That is, we consider all predictors X_1, \dots, X_p , and all possible values of the cutpoint s for each of the predictors, and then choose the predictor and cutpoint such that the resulting tree has the lowest RSS.
- In greater detail, for any j and s , we define the pair of half-planes

$$R_1(j, s) = \{X | X_j < s\} \quad \text{and} \quad R_2(j, s) = \{X | X_j \geq s\}$$

minimize

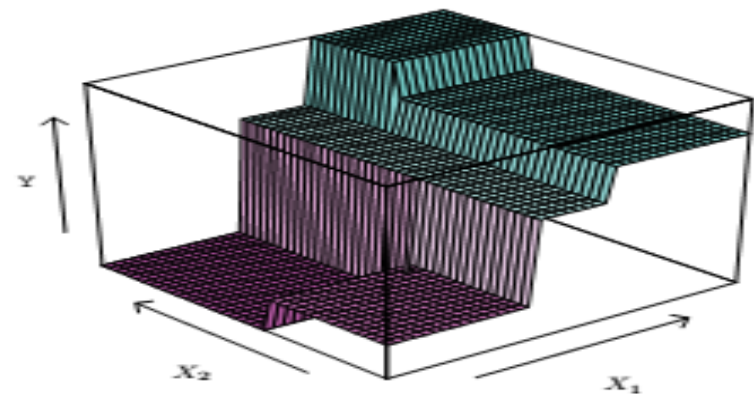
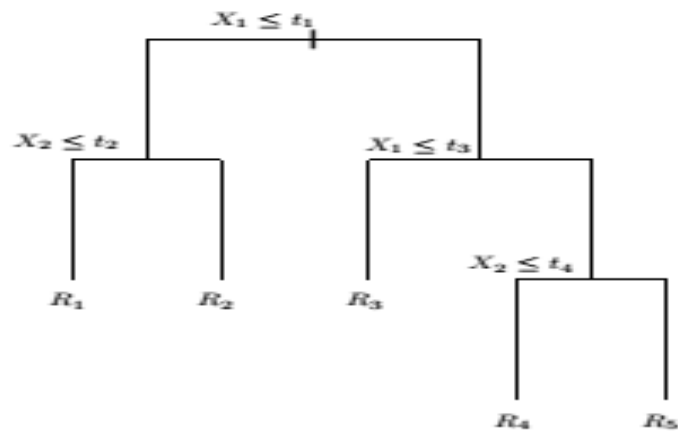
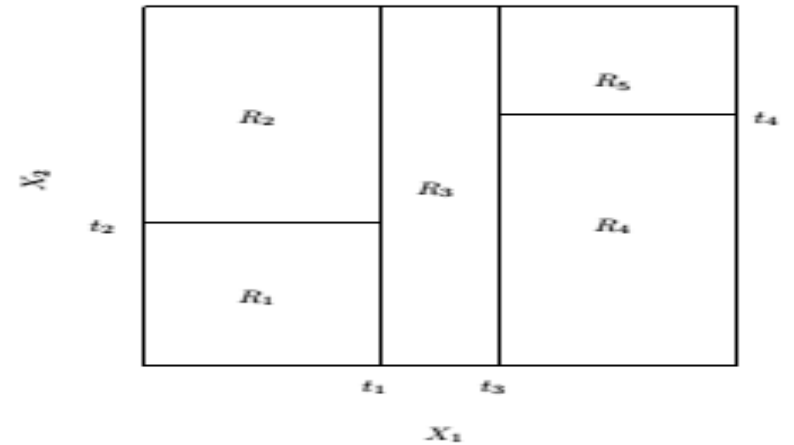
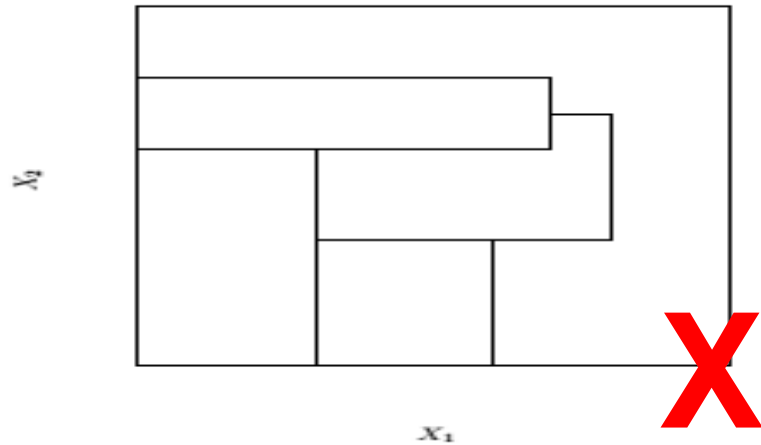
$$\sum_{i: x_i \in R_1(j, s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j, s)} (y_i - \hat{y}_{R_2})^2$$



Recursive binary splitting

- 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.
- Once the regions R_1, \dots, R_J have been created, 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.

Examples





Tree Pruning

- 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.
- This is because the resulting tree might be too complex. A smaller tree with fewer splits (that is, fewer regions R_1, \dots, R_J) might lead to lower variance and better interpretation at the cost of a little bias.



Tree Pruning

- A better strategy is to grow a very large tree T_0 , and then *prune* it back in order to obtain a *subtree*. How do we determine the best prune way to prune the tree?
- Intuitively, our goal is to select a subtree that leads to the lowest test error rate. Given a subtree, we can estimate its test error using cross-validation or the validation set approach.
- However, estimating the cross-validation error for every possible subtree would be too cumbersome, since there is an extremely large number of possible subtrees.
- Instead, we need a way to select a small set of subtrees for consideration.

Cost complexity pruning

- **Cost complexity pruning**—also known as *weakest link pruning*—gives us a way to do just this. 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 corresponds a subtree $T \subset T_0$ such that:

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

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data. When $\alpha = 0$, then the subtree T will simply equal T_0
- However, as α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity in equation will tend to be minimized for a smaller subtree.
- For more detail on CCP see http://mlwiki.org/index.php/Cost-Complexity_Pruning

Cost complexity pruning generates a series of trees $T_0 \dots T_m$ where T_0 is the initial tree and T_m is the root alone. At step i , the tree is created by removing a subtree from tree $i - 1$ and replacing it with a leaf node with value chosen as in the tree building algorithm. The subtree that is removed is chosen as follows:

1. Define the error rate of tree T over data set S as $\text{err}(T, S)$.
2. The subtree that minimizes $\frac{\text{err}(\text{prune}(T, t), S) - \text{err}(T, S)}{|\text{leaves}(T)| - |\text{leaves}(\text{prune}(T, t))|}$ is chosen for removal.



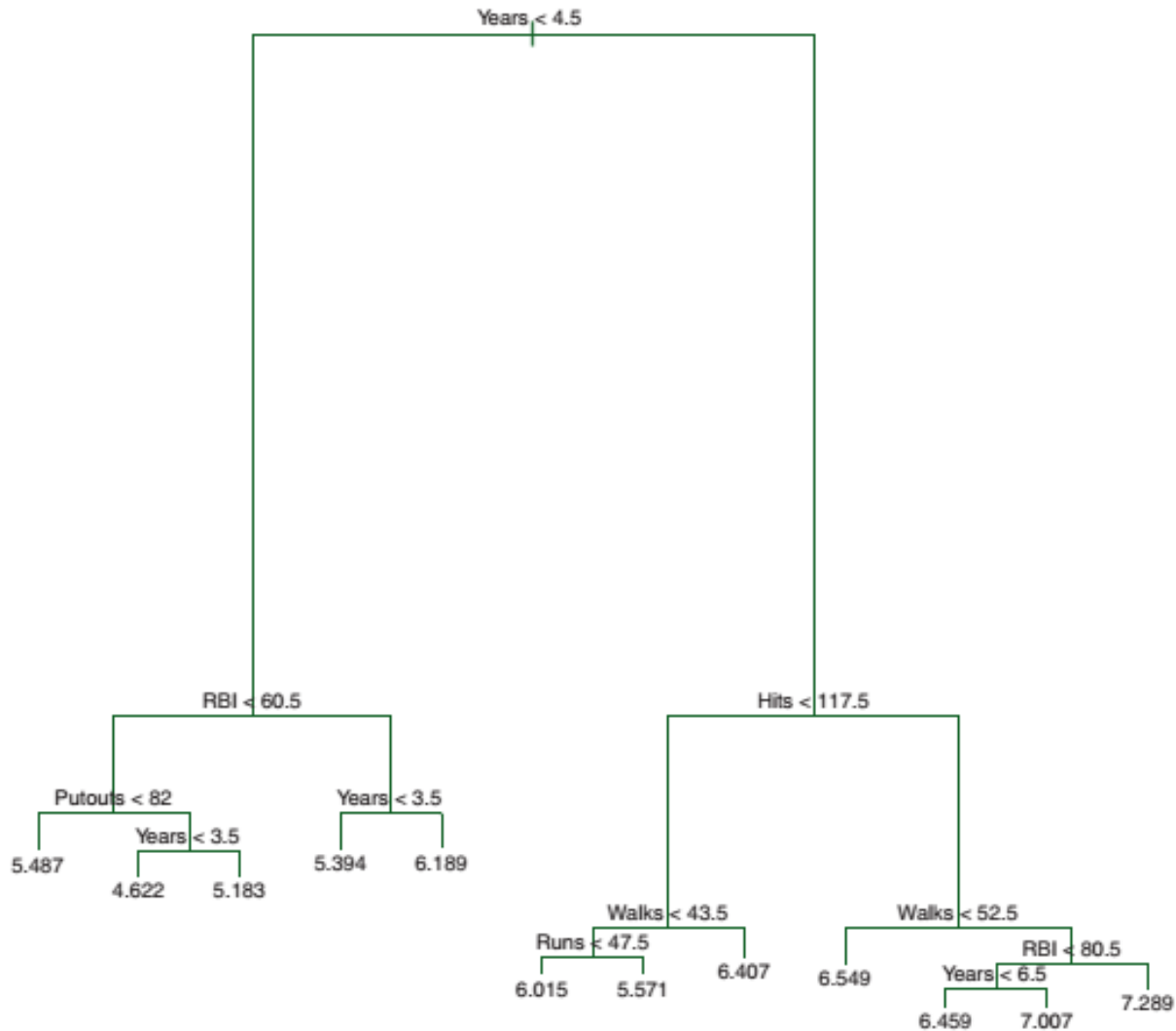
Pruning

Algorithm *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 $k = 1, \dots, K$:
 - (a) Repeat Steps 1 and 2 on all but the k th fold of the training data.
 - (b) Evaluate the mean squared prediction error on the data in the left-out k th 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 .
-

Unpruned Hitters Tree






Classification Trees

- A *classification tree* is very similar to a regression tree, except that it is classification 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.
- In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the *class proportions* among the training observations that fall into that region.
- The **classification error** rate is simply the fraction of the training observations in that region that do not belong to the most common class

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

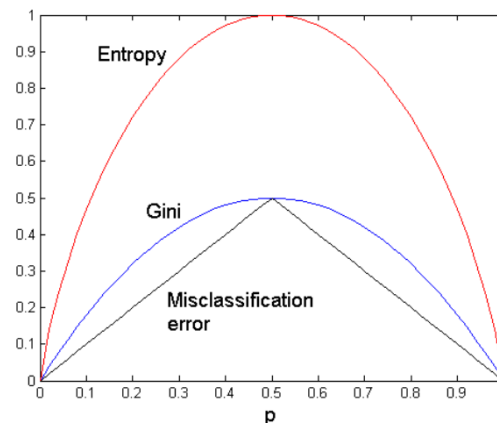


proportion of training observations in the m_{th} region that are from the k_{th} class

Gini Index and Cross-Entropy

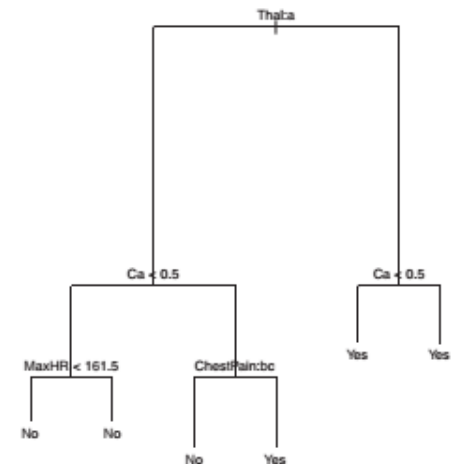
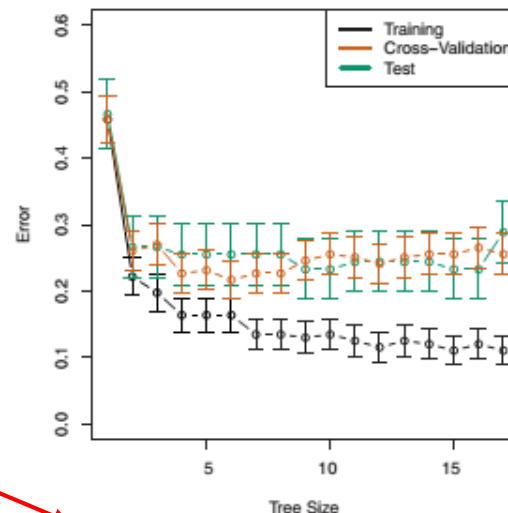
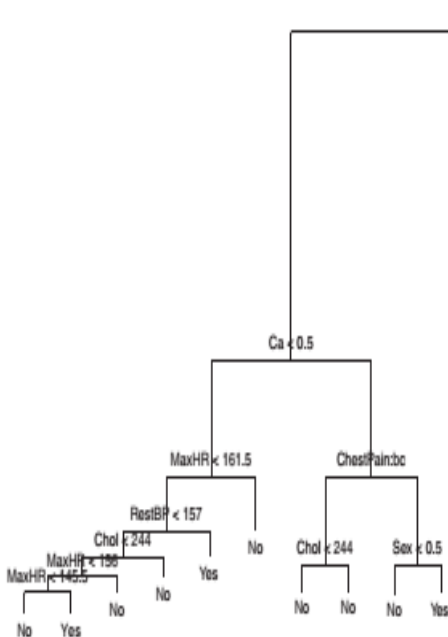
- In practice two other measures are preferable.
- Gini Index: $G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk})$ (a measure of total variance across the K classes)
- The Gini index takes on a small value if all of the p_{mk} 's are close to zero or one.
- For this reason the Gini index is referred to 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 $D = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$.

For a 2-class problem:



Heart Dataset

- 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.



7/11 = yes 9/9 = yes



Advantages and Disadvantages of Trees

Advantages:

- ▲ 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.
- ▲ 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.

Disadvantages

- ▼ Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.
- ▼ Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.

Possible solutions???



Bagging

- ***Bootstrap aggregation***, or *bagging*, is a general-purpose procedure for reducing the variance of a statistical learning method;
- It is particularly useful and frequently used in the context of decision trees.
- ***averaging a set of observations reduces variance.***
- Hence a natural way to reduce the variance and hence increase the prediction accuracy of a statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions



Bagging

$\hat{f}^1(x), \hat{f}^2(x), \dots, \hat{f}^B(x)$ using B separate training sets $\longrightarrow \hat{f}_{\text{avg}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$

- Of course, this is not practical because we generally do not have access to multiple training sets.
- Instead, 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 b th bootstrapped training set in order to get

$$\hat{f}^{*b}(x)$$

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

- To apply bagging to regression trees, we simply construct B regression trees using B bootstrapped training sets, and average the resulting predictions. These trees are grown deep, and are not pruned. Hence each individual tree has high variance, but low bias.
- Averaging these B trees reduces the variance. Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure.



Random Forests

- *Random forests* provide an improvement over bagged trees by way of a random small tweak that **decorrelates** the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, *a random sample 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 sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$



Random Forests

- In other words, in building a random forest, at each split in the tree, the algorithm is ***not even allowed to consider*** a majority of the available predictors.
- Suppose that there is **one very strong predictor** in the data set, along with a number of other moderately strong predictors.
- Then in the collection of bagged trees most or all of the trees will use this strong predictor in the top split.
- Consequently, all of the bagged trees will look quite similar to each other.
- Hence the predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.



Random Forests

- Random forests overcome this problem by forcing each split to consider only a subset of the predictors.
- We can think of this process as ***decorrelating*** the trees, thereby making the average of the resulting trees less variable and hence more reliable.
- The main difference between bagging and random forests is the choice of predictor subset size ***m***.