✱ Regression Trees :

Dataset : Hitters dataset :

Contains salaries of baseball players given their years in the league and no. of hits.

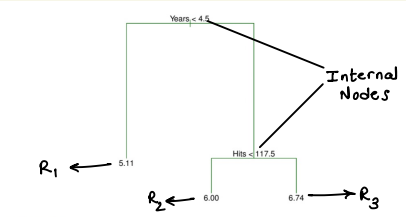


FIGURE 8.1. *For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$) indicates 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.*

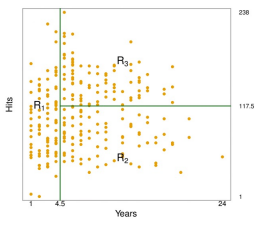


FIGURE 8.2. *The three-region partition for the Hitters data set from the regression tree illustrated in Figure 8.1.*

R_1 = Years < 4.5

R_2 = Years ≥ 4.5 , hits < 118

R_3 = Years ≥ 4.5 , hits ≥ 118

Regions R_1, R_2 and R_3 are known as 'terminal nodes' or 'leaves' of tree.

+ Decision trees are drawn upside down (obvious).
+ Split points are k/a internal nodes.

− Advantage of Regression Trees :
Easy to interpret and can be represented graphically.

Prediction via Stratification of the Feature Space

We now discuss the process of building a regression tree. Roughly speaking, there are two steps.

1. We divide the predictor space — that is, 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. 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.

✱✱ For instance, suppose that in Step 1 we obtain two regions, R_1 and R_2, and that the response mean of the training observations in the first region is 10, while the response mean of the training observations in the second region is 20. Then for a given observation $X = x$, if $x \in R_1$ we will predict a value of 10, and if $x \in R_2$ we will predict a value of 20.

For defining regions $R_1, R_2 \ldots R_J$, we find boxes $R_1, R_2, R_3 \ldots R_J$ that minimize RSS

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

* Recursive Binary Splitting :
- Greedy approach (bcz finds best branching at each stage)
- Initially, there is one one region i.e. complete set of predictors
- Successively split the predictor space.

- We first select the predictor X_j 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.
- 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.

- For any j, s, the pair of half planes can be defined as: $R_1(j,s) = \{X | X_j < s\}$ and $R_2(j,s) = \{X | X_j \geq s\}$

j and s are chosen such that:

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

- One $R_1, R_2 \ldots R_J$ have been defined, the response for test observation is predicted using mean of responses of every region.

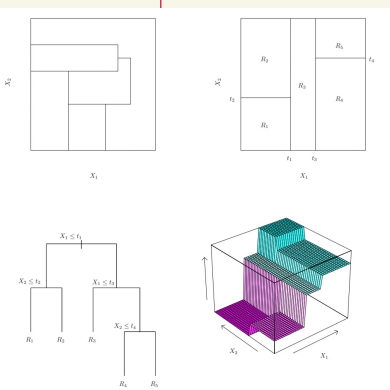


FIGURE 8.3. Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A 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.

Pruning a tree

- 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.

But this is a vague approach. It is possible that a 'not so important' predictor is used to split and later a more efficient split (a predictor which gives much less RSS) is found.

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

* Cost Complexity Pruning : (Weakest Link Pruning)

- 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

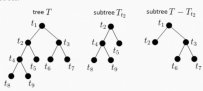
$$\left\{ \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \right\}$$

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 m^{th} terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

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

Pruning Subtrees

Subtrees:



Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.
   Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Choosing the best subtree

- 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 $\hat{\alpha}$.

+ $\alpha = 0 \Rightarrow T_0$ i.e. Complete dec. tree
+ as $\alpha \uparrow$, having tree with all terminal nodes is 'costly', this results in creation of subtrees.
+ as $\alpha \uparrow$, branches get pruned in a nested and predictable fashion.

* Applying above knowledge to Hitters data :

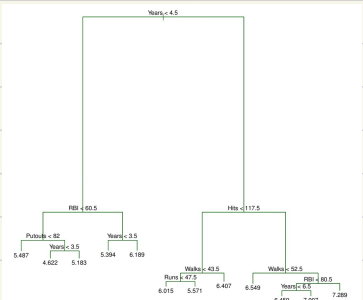
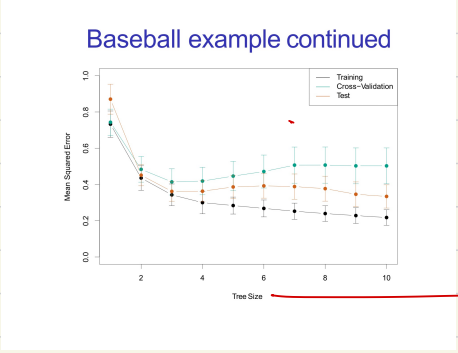


FIGURE 8.4. *Regression tree analysis for the* Hitters *data. The unpruned tree that results from top-down greedy splitting on the training data is shown.*



+ Training size = 132
+ Test size = 131
+ 6 fold CV used

Tree size = $|T|$
= No. of leaves after pruning for some α.

Classification Trees :

- For a classification tree, we predict (assign) the most occuring class to the test observation.
- We use recursive binary splitting to construct the tree.
- RSS cannot be used for classification as a splitting criterion.
- Hence, we use **classification error rate** :

  Fraction of training observations that do not belong to majority class

  eg. : 4 out of 10 training obs. belong to class 0 (i.e. 6 to class 1)
  So, CER = $4/10 = 0.4$

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

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

- So, 2 diff. measures can be used instead :
  a) Gini Index
  b) Entropy

- An alternative to the Gini index is *cross-entropy*, given by

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

- The tree algorithms that use cross entropy are ID3, C4.5, and C5.0
- It turns out that the Gini index and the cross-entropy are very similar numerically.

Justifica^n: $\hat{p}_{mk} \cdot \log \hat{p}_{mk}$ will be close to zero if $\hat{p}_{mk}$ is very small or close to 1

$\hat{p}_{mk}$ close to $0 \Rightarrow$ zero due to $\hat{p}_{mk}$
$\hat{p}_{mk}$ close to $1 \Rightarrow$ zero due to $\log(\hat{p}_{mk})$

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.

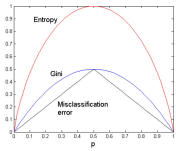
- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.
- The Tree Algorithm in this case is called CART (Classification and Regression Trees)

$\hat{p}_{mk}$ is the fraction of training observations belonging to k^{th} class

NOTE: Any of the 3 i.e.
a) Classifi. error rate
b) Gini Index
c) Entropy
can be used for pruning but classific^n error is preferred if accuracy is the goal.

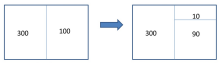
Comparison of Impurity Measures

- For a two-class problem



Weighted Impurity

- Most of the time, to be more fair, we add up the *weighted impurity* of the regions resulting from a split.
- We choose weights to be the fraction of data points in each region.

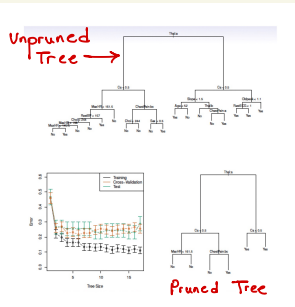


Example: heart data

- 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.

Example: heart data

- 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.



* Linear Model vs. Tree Model.

- LR: $f(x) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$

Tree Model: $f(x) = \sum_{m=1}^{M} c_m \cdot 1(x \in R_m)$

$R_1, R_2, \ldots R_M$ represent partition space.

- If rel^n bet^n each predictor and response is linear: Use LR
- If rel^n bet^n predictor and response is non-linear and complex: use trees.

Good practice:
① Make plots of predicts with response.
② Decide the approach to follow after analyzing the plot.

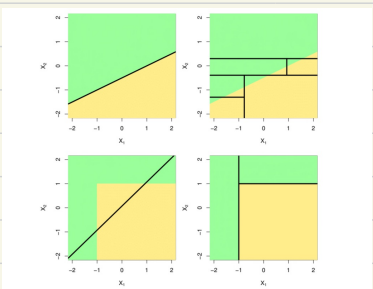


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: *Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).*

8.1.4 Advantages and Disadvantages of Trees

Decision trees for regression and classification have a number of advantages over the more classical approaches seen in Chapters 3 and 4:

- ▲ 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.
- ▼ 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.
- ▼ 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.

However, by aggregating many decision trees, using methods like *bagging, random forests*, and *boosting*, the predictive performance of trees can be substantially improved. We introduce these concepts in the next section.

* **Ensemble Methods:**
  - Approach that combines many 'simple building blocks to form a strong (efficient) model.
  - Each simple block is k/a weak learner as it leads to mediocre predictions.

NOTE:
+ Bagging, Boosting and Random Forest and Bayesian Additive Regression Trees are ensemble techniques which use Regression and Classification trees as "building blocks".

* **Bagging** : (Bootstrap Aggregation)

- Procedure to reduce the variance of a method.
* - Averaging a set of observations reduces variance.
- So, in order to reduce variance, what we can do is take many training sets from the population.
  - Build a model for each training set.
  - Average the predictions of each model.
  - Mathematically, calculate $\hat{f}^1(x), \hat{f}^2(x), \hat{f}^3(x) \ldots \hat{f}^B(x)$ for B training sets and average them to get single low-variance model.

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

  - Practically, taking many training sets is not possible so we can bootstrap samples from single training set.

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

* Bagging with Regression Trees :

- Construct B reg. trees using B bootstraps of training data.
- Average resulting predictions.

* Bagging with classification tree :

- Similar to regression approach.
- Assign class which is returned by maximum bagging trees.
- Large value of B does not lead to overfitting.
- B = 100 is sufficient to achieve good performance.

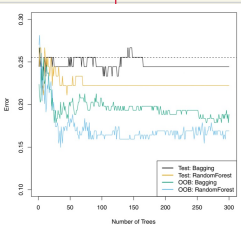


FIGURE 8.8. *Bagging and random forest results for the* Heart *data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is — by chance — considerably lower.*

* Inference :

1) Test error becomes constant i.e. the model reaches maximum accuracy after a certain value of B
   - B ≈ 150 for Bagging
   - B ≈ 70 for Random Forest
2) OOB classification error < test classification error for Bagging ⇒ **OOB is more effective measure**

* Out of Bag Error Estimation (OOB)

Out-of-Bag Error Estimation

- 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 called the *out-of-bag* (OOB) observations.

Out-of-Bag Error Estimation

- We can predict the response for the i^{th} observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the i^{th} observation, which we average.
- This estimate is essentially the LOO cross-validation error for bagging, if B is large.

- This way we can get n OOB predictions for n observations.

- OOB MSE (reg. tree) and classification error (class. tree) can be calculated.

Bagging improves accuracy at the cost of interpretability

Imp:- Decision trees are easy to construct using available data.
BUT
Cannot classify 'unseen' data effectively.

* **Random Forests :**

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.
- But when building these 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.

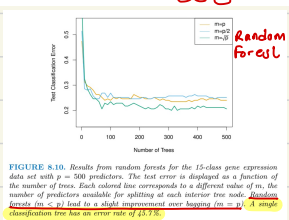
Usually $m = \sqrt{p}$

eg.: Choose 4 predictors if $p = 13$

Random Forests: An Alternative Version

- Alternatively, one can build *B* decision trees, each of which on a random subset of *m* features.
- Instead of a bootstrap sample from the training set, one can even use a subsample with replacement from the training set, which is equivalent to a bootstrap sample with a size different from the training set.

* if $m = p \Rightarrow$ Bagging



Random forest

FIGURE 8.10. *Results from random forests for the 15-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%.*

Use this OOB dataset as test data and check if it is correctly classified.

OOB error:
- Check for misclassification of OOB observation.
- There will be $B/3$ trees which do not contain this OOB observation.
- So, we have $B/3$ values of classification for this observation
- The $\frac{\text{(no. of misclassifications)}}{B/3}$

This is OOB error.

* **Algorithm to generate a random forest :**

Step 1 : Bootstrap data (Randomly select data) {can be selected again}

Step 2 : Create a decision tree using bootstrapped dataset considering a subset of predictors ($m < p$) (Usually, $m = \sqrt{p}$) at each step.

Step 3 : Continue steps 1 and 2 to get multiple decision trees.

Testing :
- To predict the decision (class) of a test observation, pass it to each tree and store the result.
- Assign the class with max. votes.

Bagging

We can check the accuracy of the Random Forest using OOB error.
- Typically $1/3^{rd}$ of the original data is not a part of the bootstrapped dataset.

For missing data: ① Categorical → Mode ② Quantitative → Median } Preferred practice

Proximity Matrix for handling missing data :

- Create an (n x n) matrix.
- Suppose n = 4
- For samples which end up at same leaf node, insert 1 at those positions in the proximity matrix.

	1	2	3	4
1				
2				
3				1
4			1	

eg: 3 and 4 end at same leaf node.
Insert 1 at (3,4) and (4,3)

- Do, this for all the trees. Add 1 if there is already an element at that position.

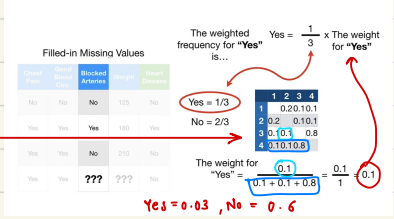
	1	2	3	4
1				
2			1	1
3		1		2
4		1	2	

eg: For second tree, obs. 2,3,4 end up at same leaf node. Fill (2,3) (3,2) (3,4) (4,3) (2,4) (4,2) in the proximity matrix.

- After all iterations, divide each proximity value by no. of trees.
- We use weighted frequency for classification.

weighted freq. = Freq. x weight of class

weight of class = Proximity value of that class / Total proximity value of that obs.



Boosting :

- Trees are grown in a sequential way.
- Each tree is grown using information from previous tree.
- Boosting does not involve bootstrap sampling. (Each tree is fit on a modified version of original dataset)

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:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \quad (8.10)$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \quad (8.11)$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \quad (8.12)$$

There are many different boosting algorithms for classification:

AdaBoost, LPBoost, BrownBoost, LogitBoost, Gradient Boosting, etc.

Example of Boosting Algorithm (AdaBoost):

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

- Boosting has 3 tuning parameters :

1. 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.
2. 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.
3. The number d of splits 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. In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable. 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.

stump

interaction depth

Boosting is remarkably resistant to overfitting, and it is fast and simple.

In fact, it can often continue to improve even when the training error has gone to zero.

It improves the performance of many kinds of machine learning algorithms.

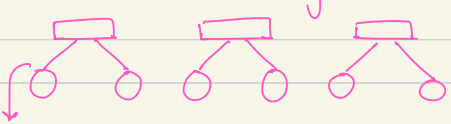
Boosting does not work when:
Not enough data, base learner is too weak or too strong, and/or susceptible to noisy data.

Boosting Intuition

- We adaptively weigh each data case.
- Data cases which are wrongly classified get high weight (the algorithm will focus on them)
- Each boosting round learns a new (simple) classifier on the weighed dataset.
- These classifiers are weighed to combine them into a single powerful classifier.
- Classifiers that that obtain low training error rate have high weight.
- We stop by using monitoring a hold out set (cross-validation).

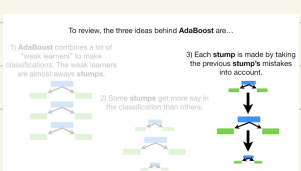
ADABOOST

- Adaboost trees are just a node with 2 leaves.



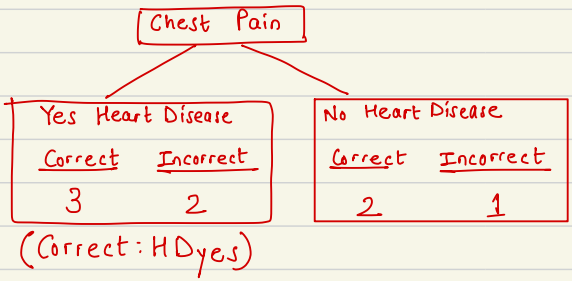
These are known as 'stumps'.

- Stumps are weak learners.
- In random forest, each tree created using bootstrap data has equal say in decision making.
- In Adaboost, diff. trees have diff. say in decision making.
- In Random Forest, sequence of tree generation does not matter.
- In Adaboost sequence matters; error in making 2nd stump is based on 1st stump and so on.



Creating a forest of stumps:

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample weights
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8



(Correct: HDyes)