

# Tree Based Models

Machine Learning | Enginyeria Informàtica

Santi Seguí | 2020-2021

# Tree-based Models

- Tree-based methods are suitable for regression and classification.
- These involve stratifying or segmenting the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods.

# Prons and Cons

- Tree-based methods are **simple** and **useful for interpretation**.
  - However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- **Bagging, random forests**, and **boosting** grows multiple trees which are then combined to yield a single consensus prediction.
  - Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

# Example - Decision tree for regression

Baseball salary data: how would you stratify it?

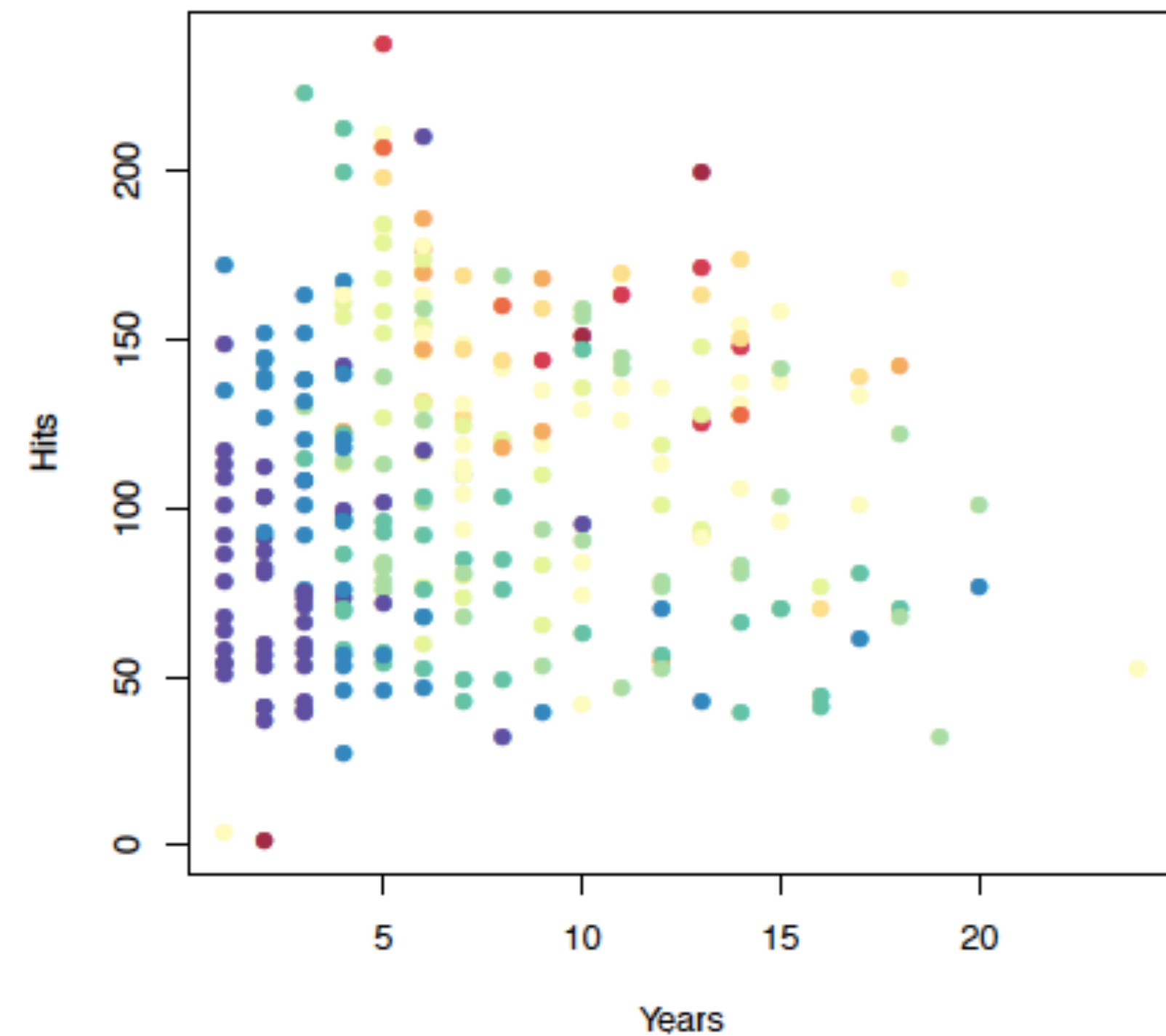


Figura: Salary is color-coded from low (blue, green) to high (yellow, red)

# Example - Decision tree for regression

Baseball salary data: how would you stratify it?

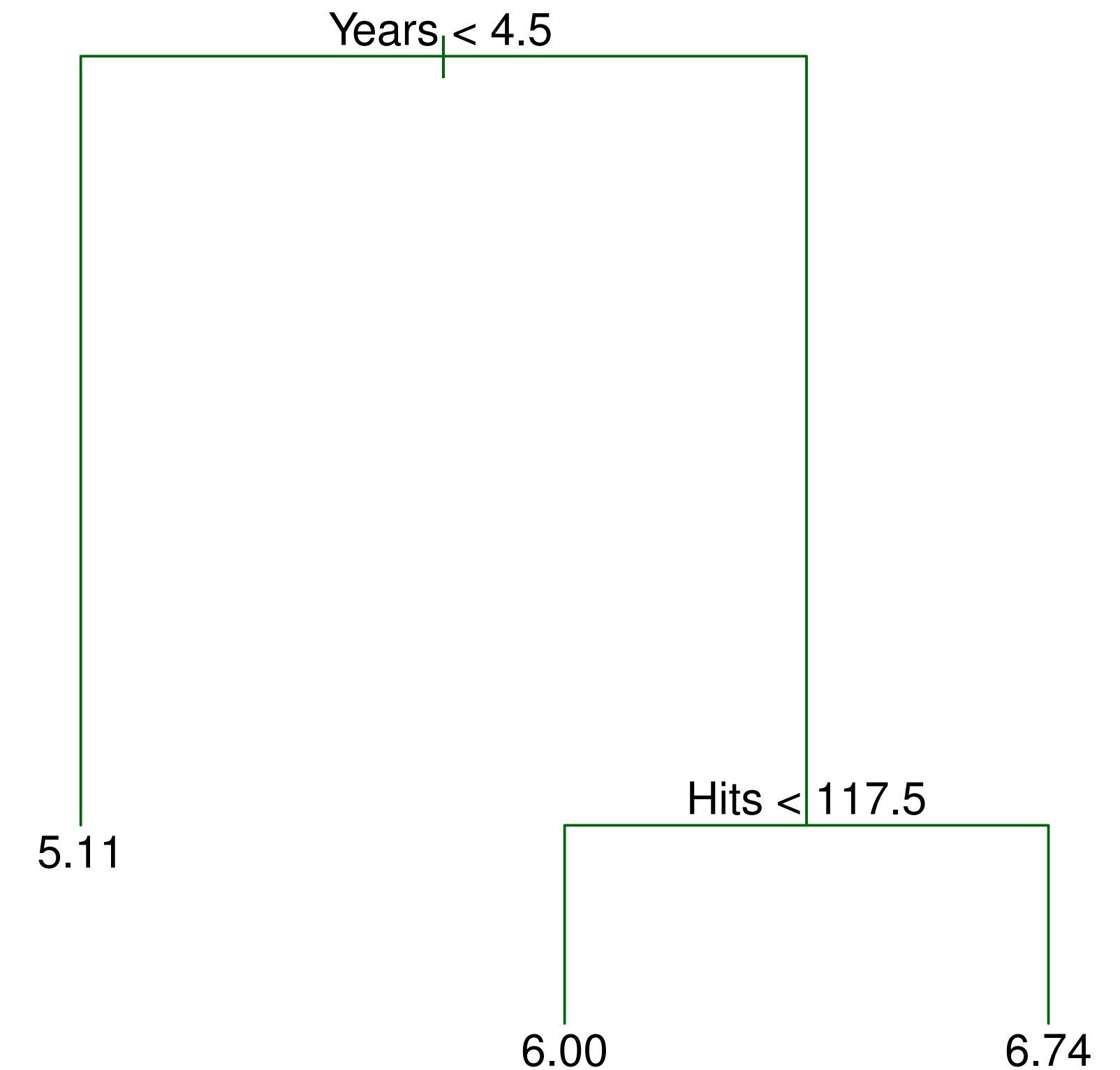
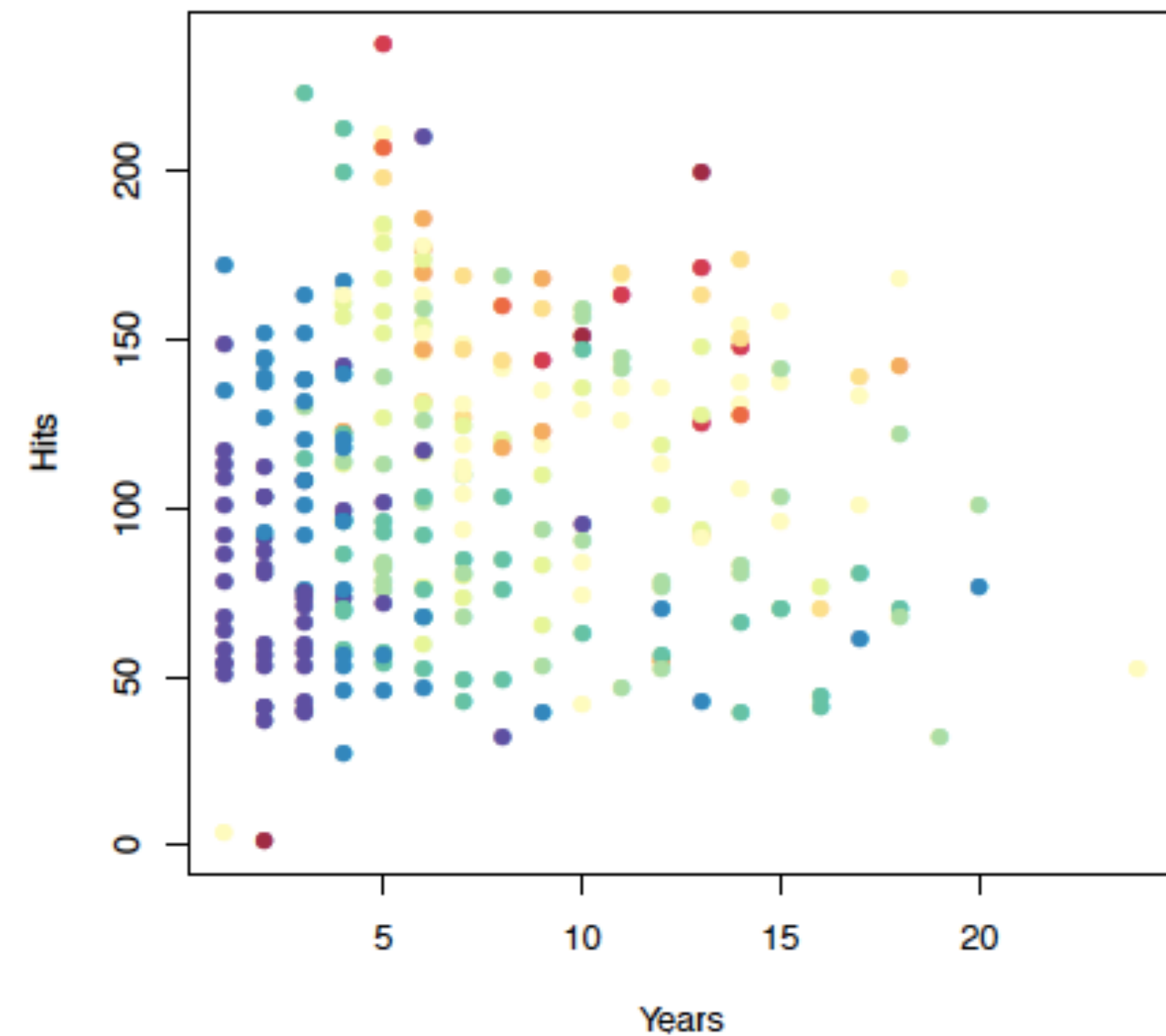


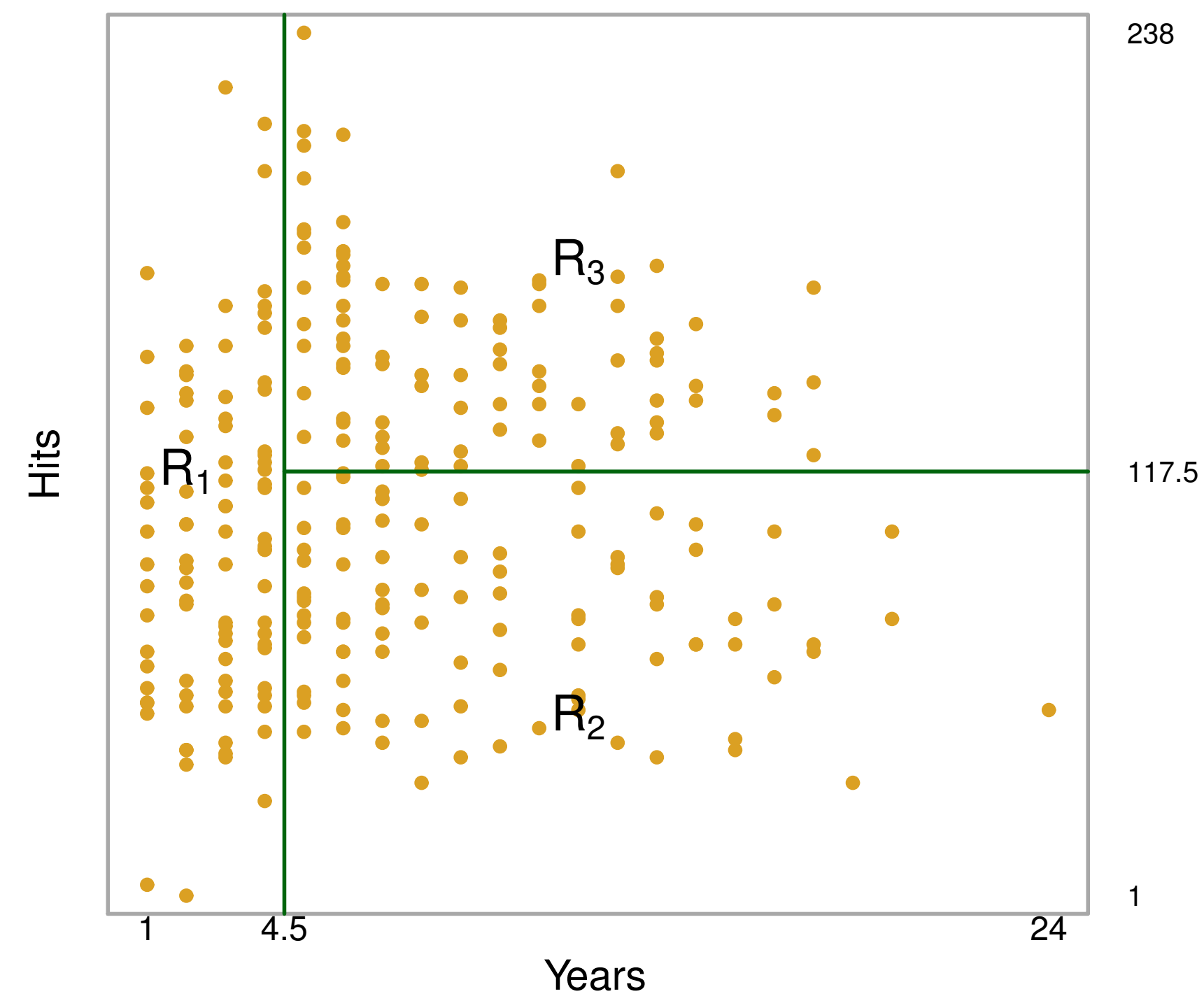
Figura: Salary is color-coded from low (blue, green) to high (yellow, red)

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





# 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 the 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

- We divide the predictor space - that is, the set of possible values for  $X_1, X_2, \dots, X_p$  - into  $J$  distinct and non-overlapping regions,  $R_1, R_2, \dots, R_J$ .
- 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  $R_j$ .
- 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  $R_1, \dots, R_J$  that minimize the RSS, given by

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

- where  $\hat{y}_{R_j}$  is the mean response for the training observations within  $j_{\text{th}}$  box.

- 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.

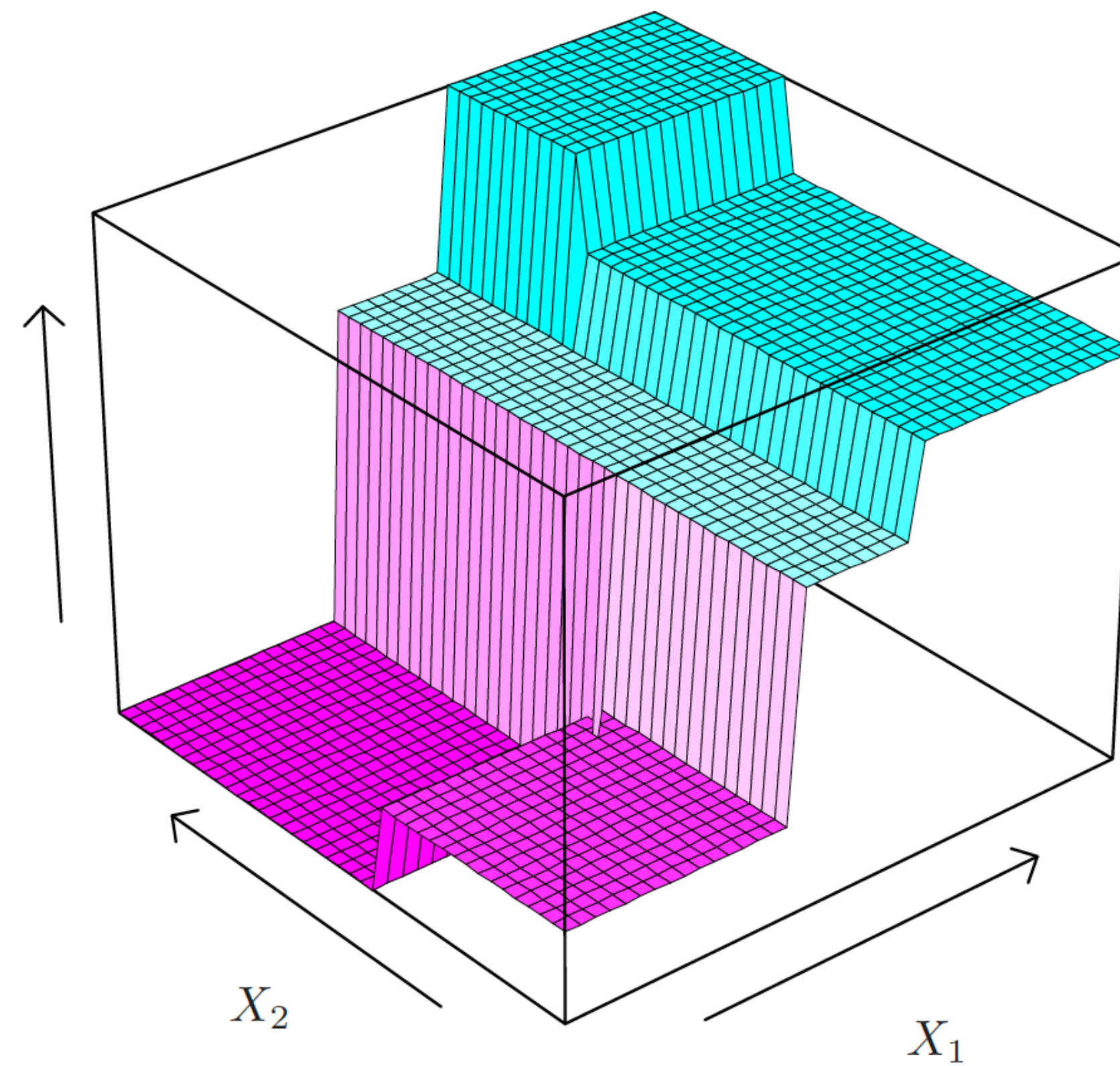
- We first select the predictor  $X_j$  and the cut point  $s$  such that splitting the predictor space into the regions  $\{X | X_j < s\}$  and  $\{X | X_j \geq s\}$  leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cut point in 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.

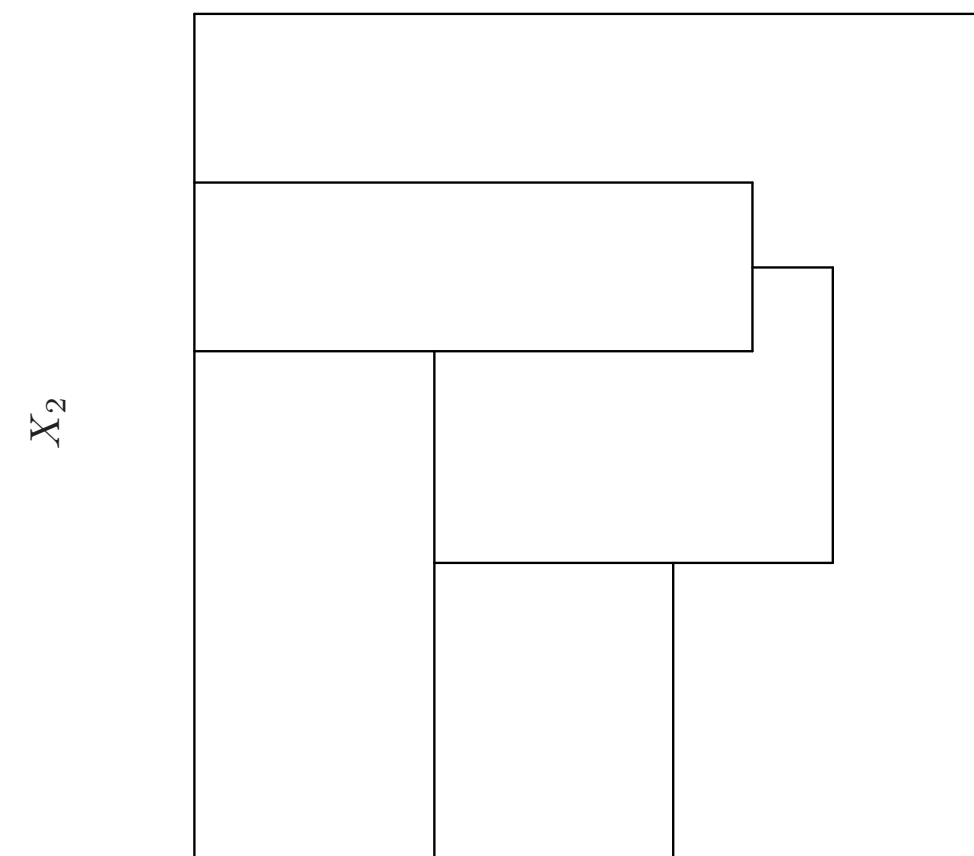
# Predictions

- 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

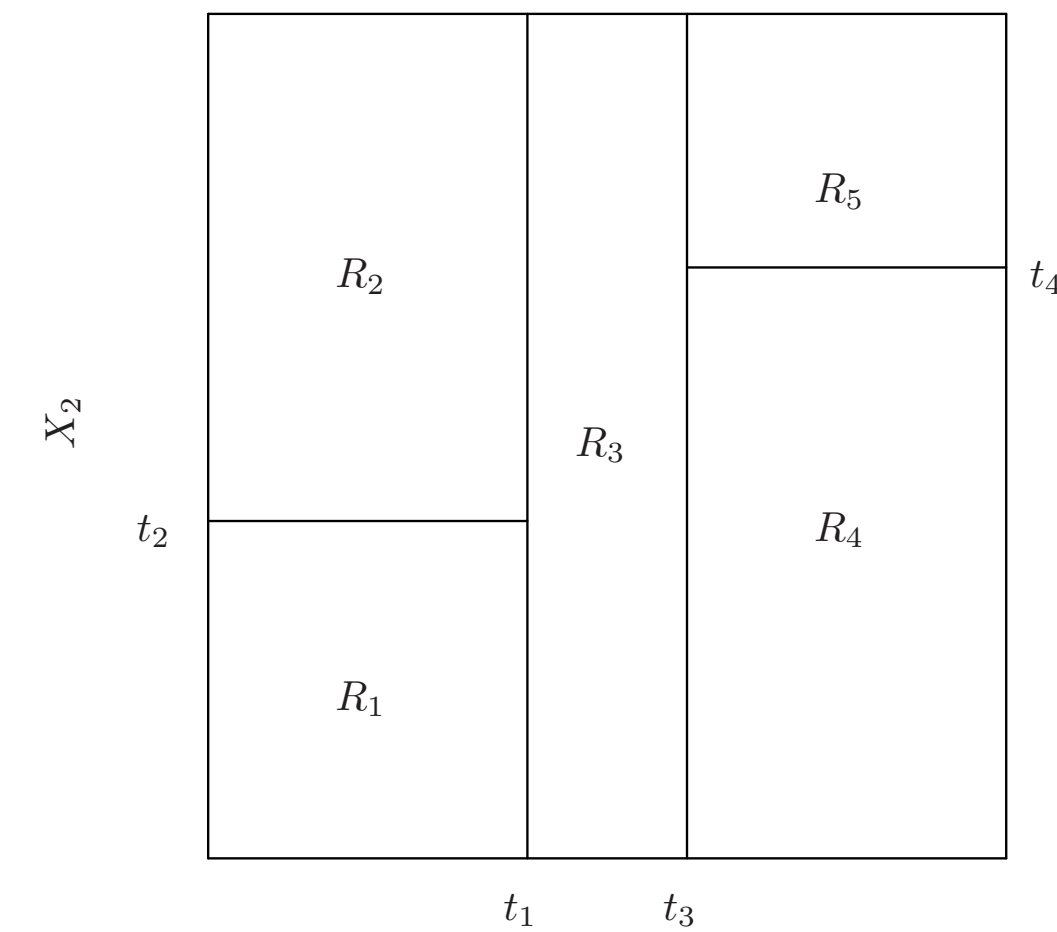


# Which will be the output?

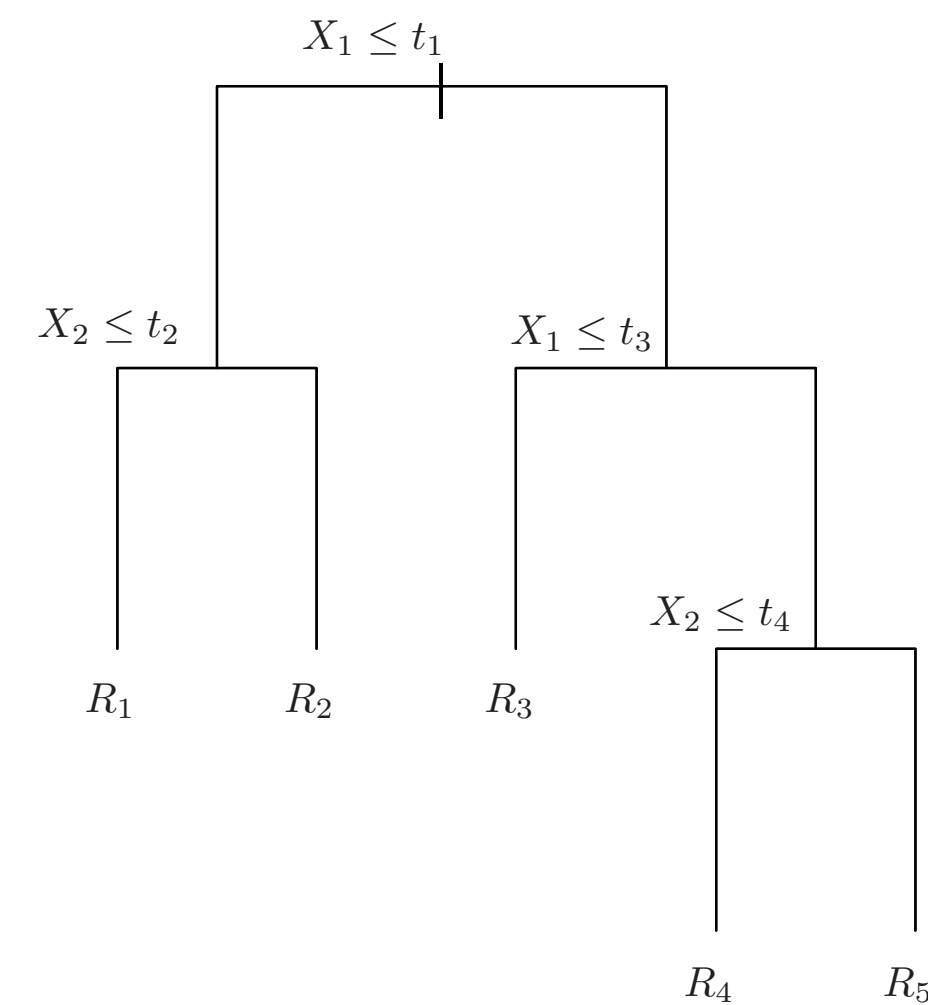




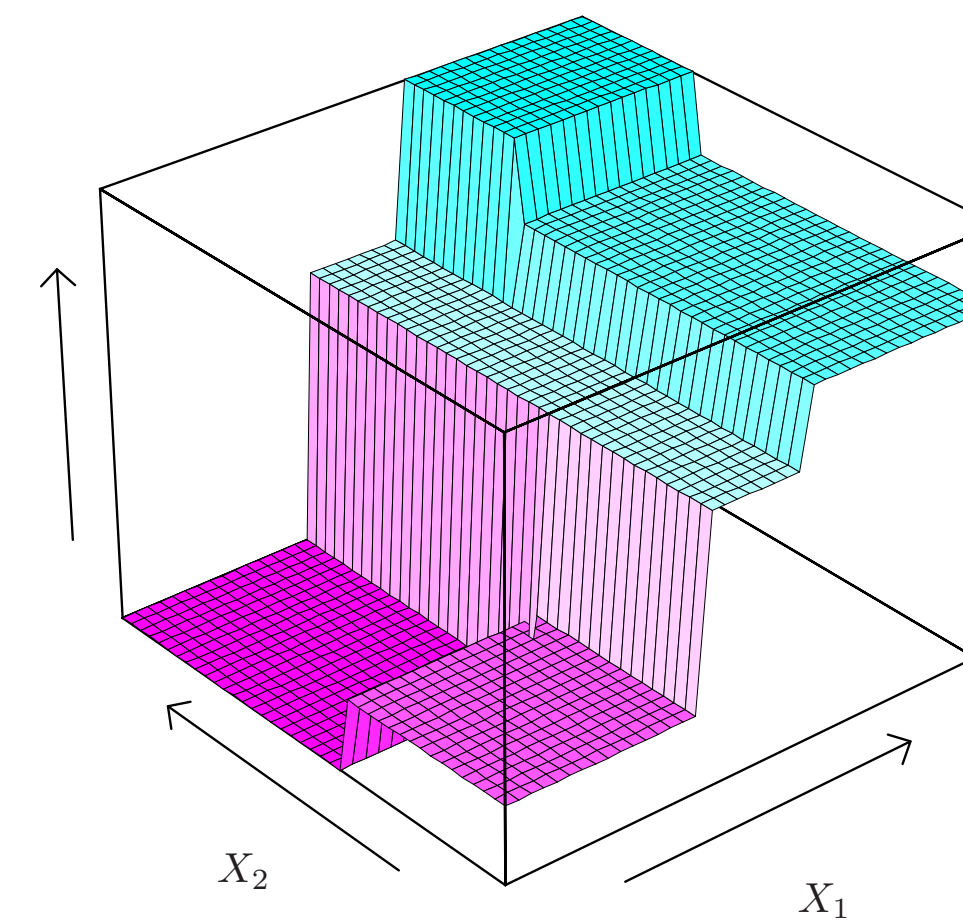
A partition of two-dimensional feature space **that could not result** from recursive binary splitting



The output of recursive binary splitting on a two-dimensional example



A tree corresponding to the partition in the top right panel.



A perspective plot of the prediction surface corresponding to that tree.



# Prunning the 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?**

# Pruning the 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  $R_1, \dots, R_J$ ) 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 the tree

- 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 non-negative tuning parameter  $\alpha$ . For each value of  $\alpha$  there corresponds a subtree  $T \in T_0$  such that

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

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

# Choosing the best subtree

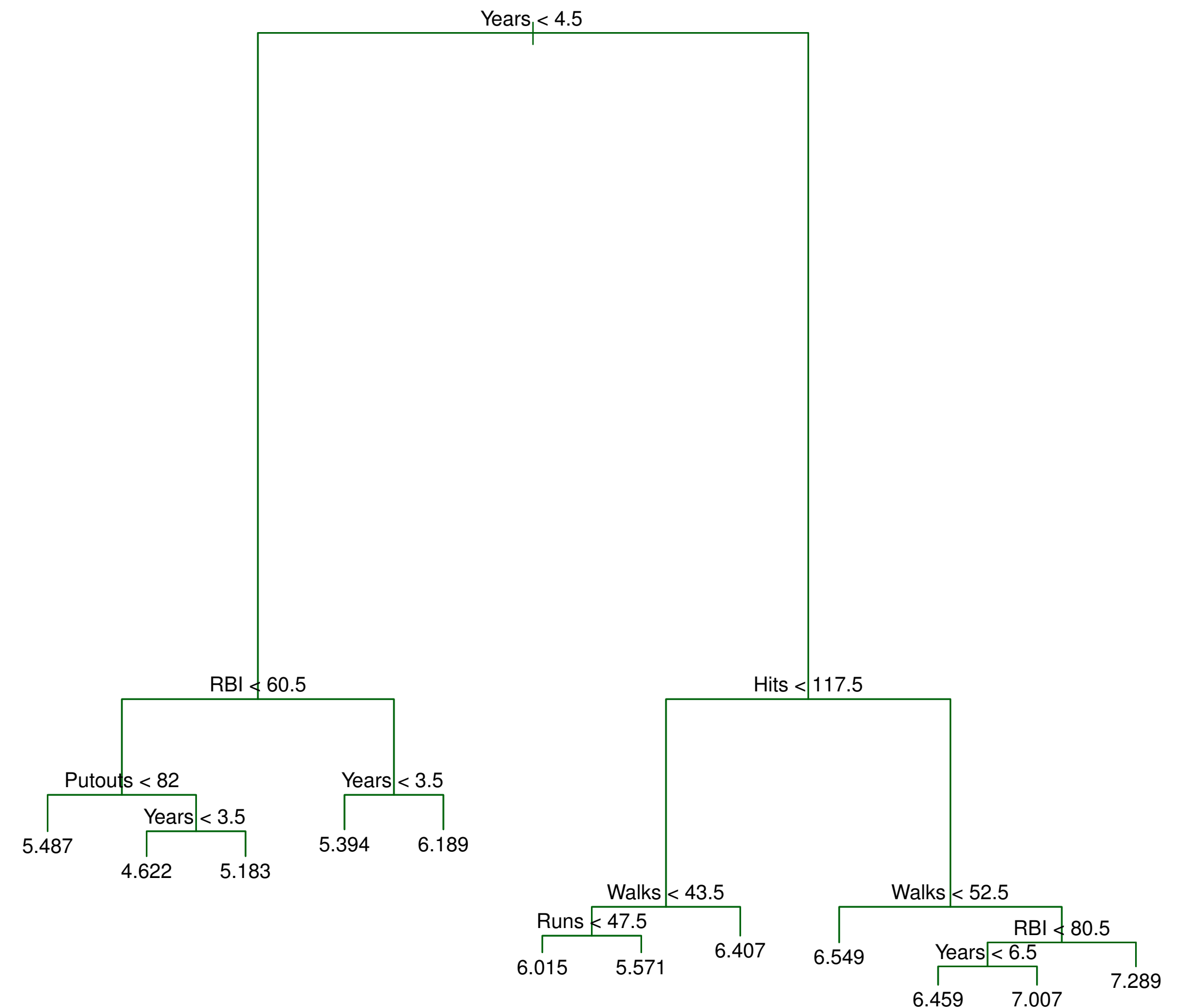
- The tuning parameter  $\alpha$  controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value  $\hat{\alpha}$  using **cross-validation**.
- We then return to the full data set and obtain the subtree corresponding to  $\hat{\alpha}$ .

# Tree algorithm

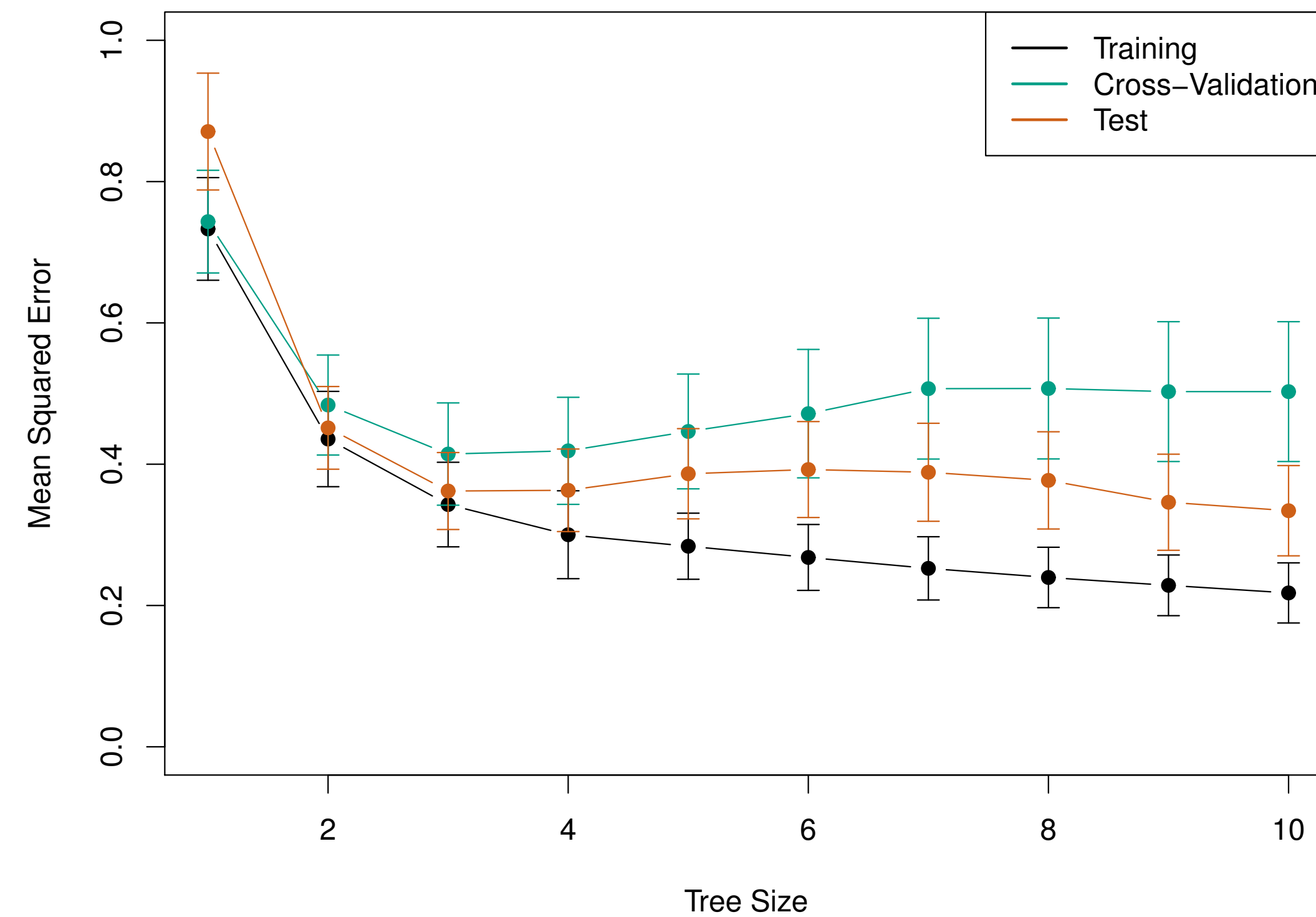
1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of  $\alpha$ .
3. Use K-fold cross-validation to choose  $\alpha$ . For each  $k = 1, \dots, K$ :
  1. Repeat Steps 1 and 2 on the  $\frac{K-1}{K}$  th fraction of the training data, excluding the  $k$  th fold.
  2. Evaluate the mean squared prediction error on the data in the left-out  $k$  th fold, as a function of  $\alpha$ .
    - Average the results, and pick  $\alpha$  to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value  $\alpha$ .

# 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  $\alpha$  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  $\alpha$ .



# Baseball example continued





# Classification Trees

# 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.

# 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  $m$ th region that are from  $k$ th class.

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

# Gini index and Deviance

- The **Gini index** is defined by

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

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

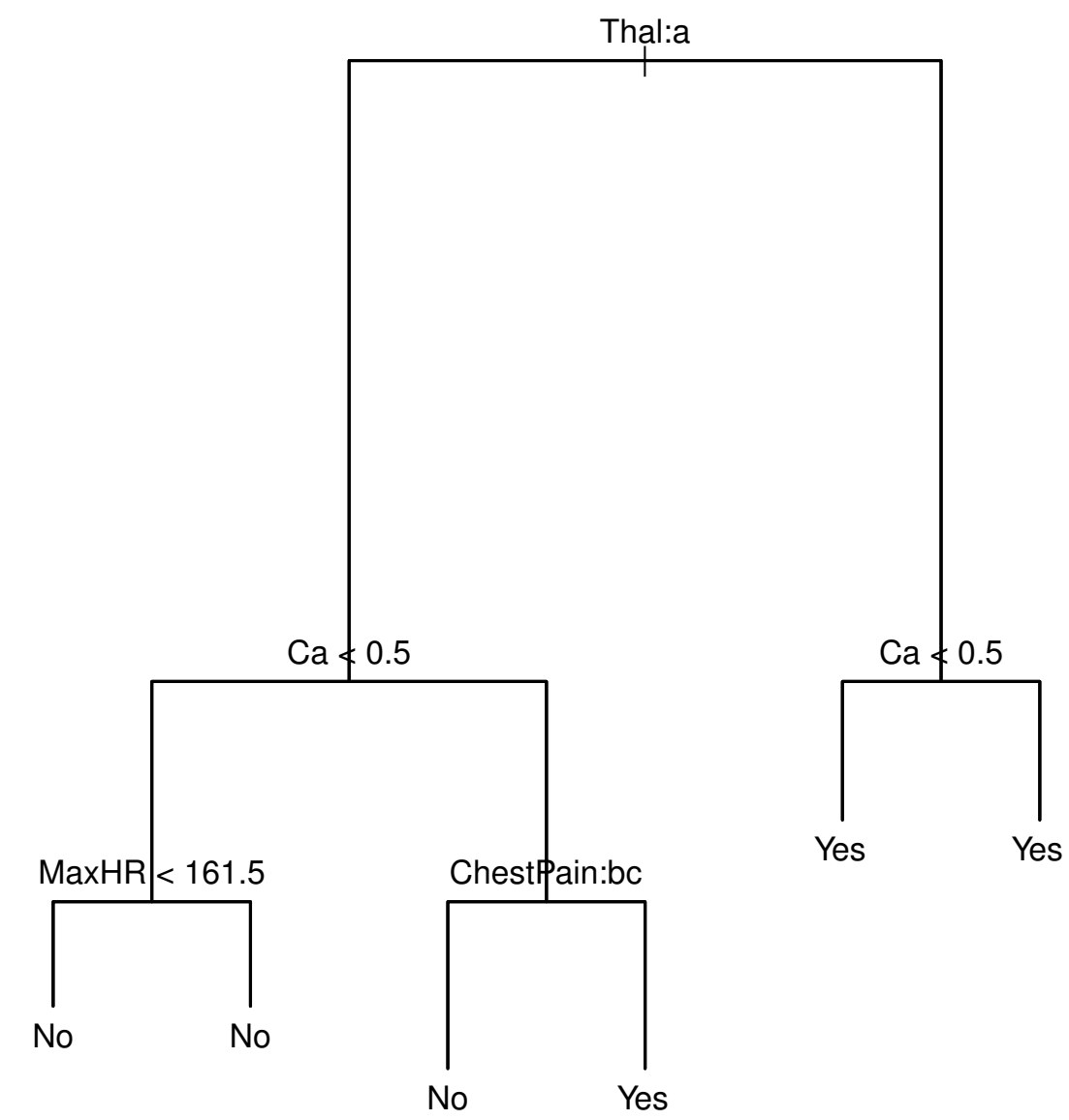
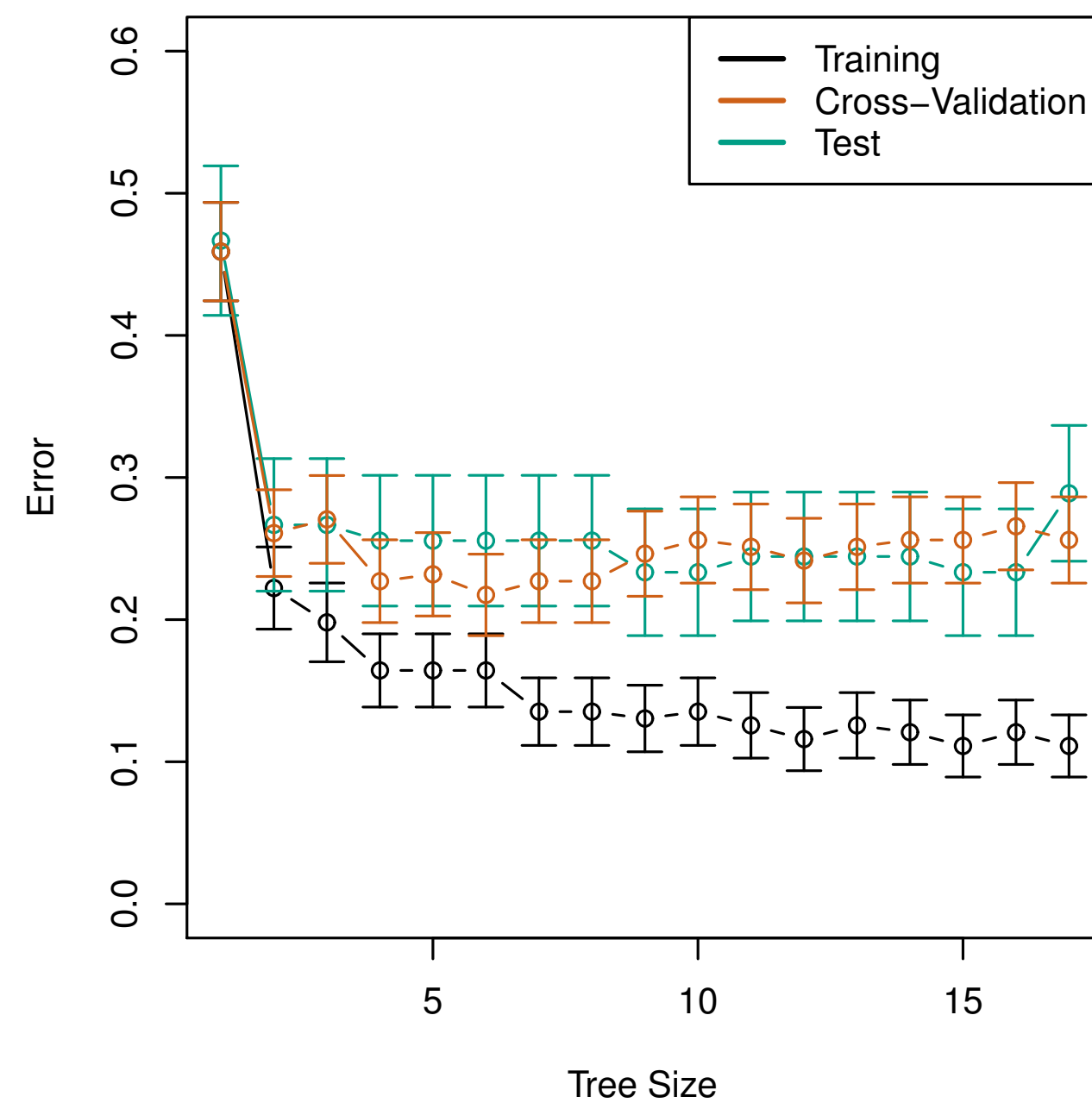
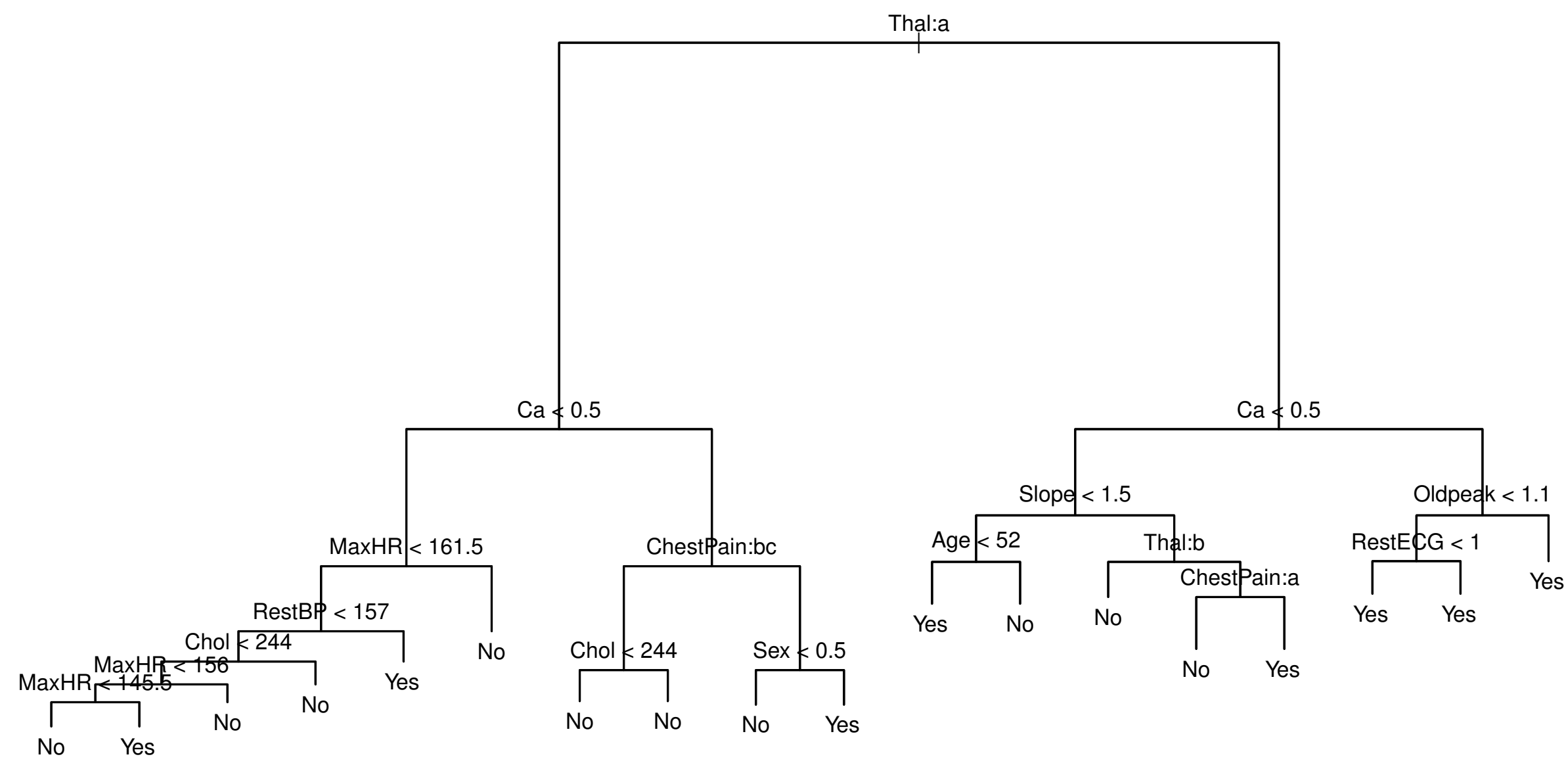
- For this reason the Gini index is referred to as a measure of **node purity** - a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is **cross-entropy**, given by

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

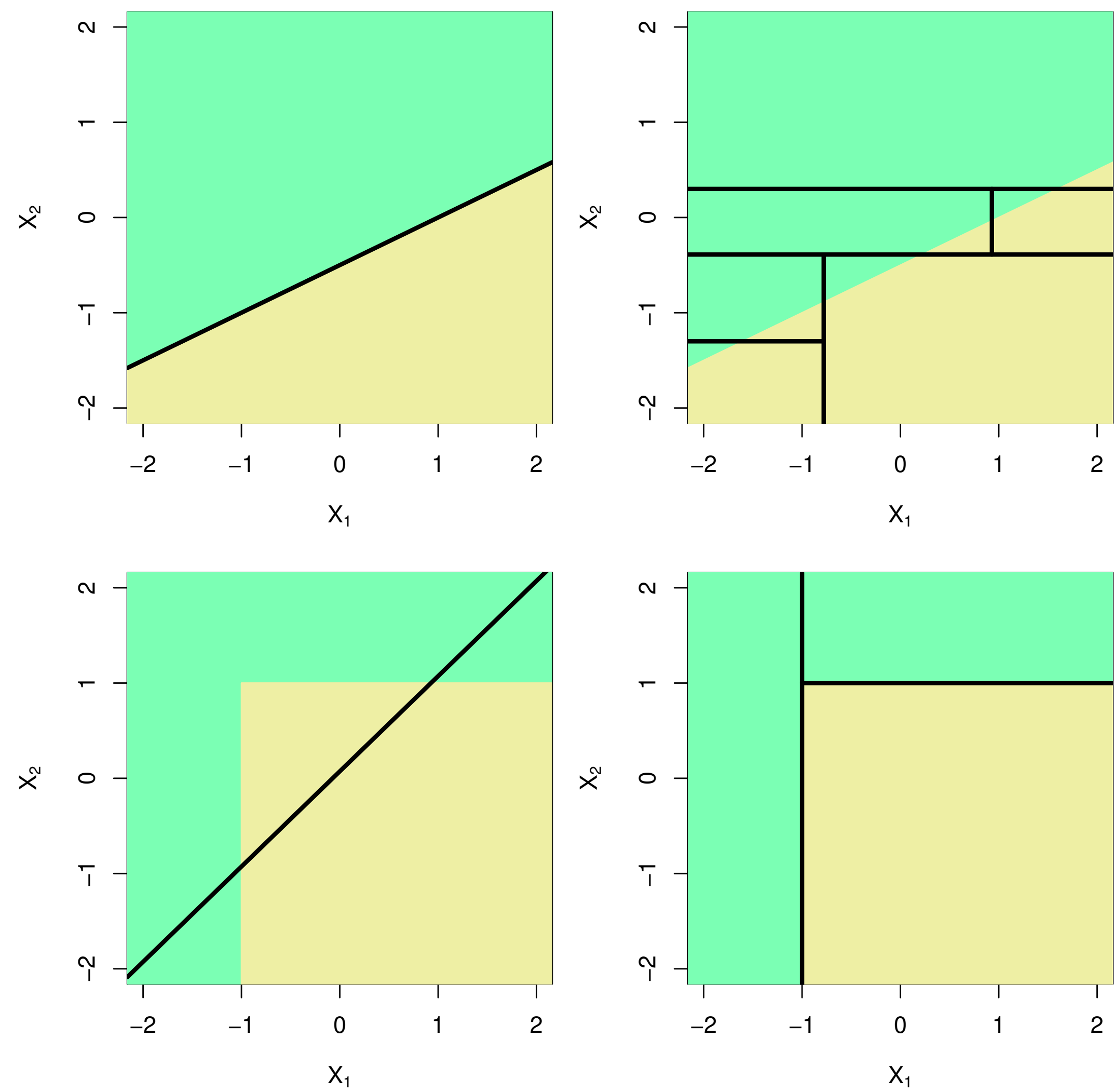
- It turns out that the Gini index and the cross-entropy are very similar numerically.

# 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.



# Trees vs Linear Models





# 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.

# How to improve Decision Trees?

*Any Idea?*

# Bagging

$$\underbrace{\mathbb{E}[(h_D(x) - y)^2]}_{\text{Error}} = \underbrace{\mathbb{E}[(h_D(x) - \bar{h}(x))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(x) - \bar{y}(x))^2]}_{\text{Bias}} + \underbrace{\mathbb{E}[(\bar{y}(x) - y(x))^2]}_{\text{Noise}}$$

$$\frac{1}{m} \sum_{i=1}^m x_i \rightarrow \bar{x} \text{ as } m \rightarrow \infty$$

$$\hat{h} = \frac{1}{m} \sum_{i=1}^m h_{D_i} \rightarrow \bar{h} \quad \text{as } m \rightarrow \infty$$

# Bagging

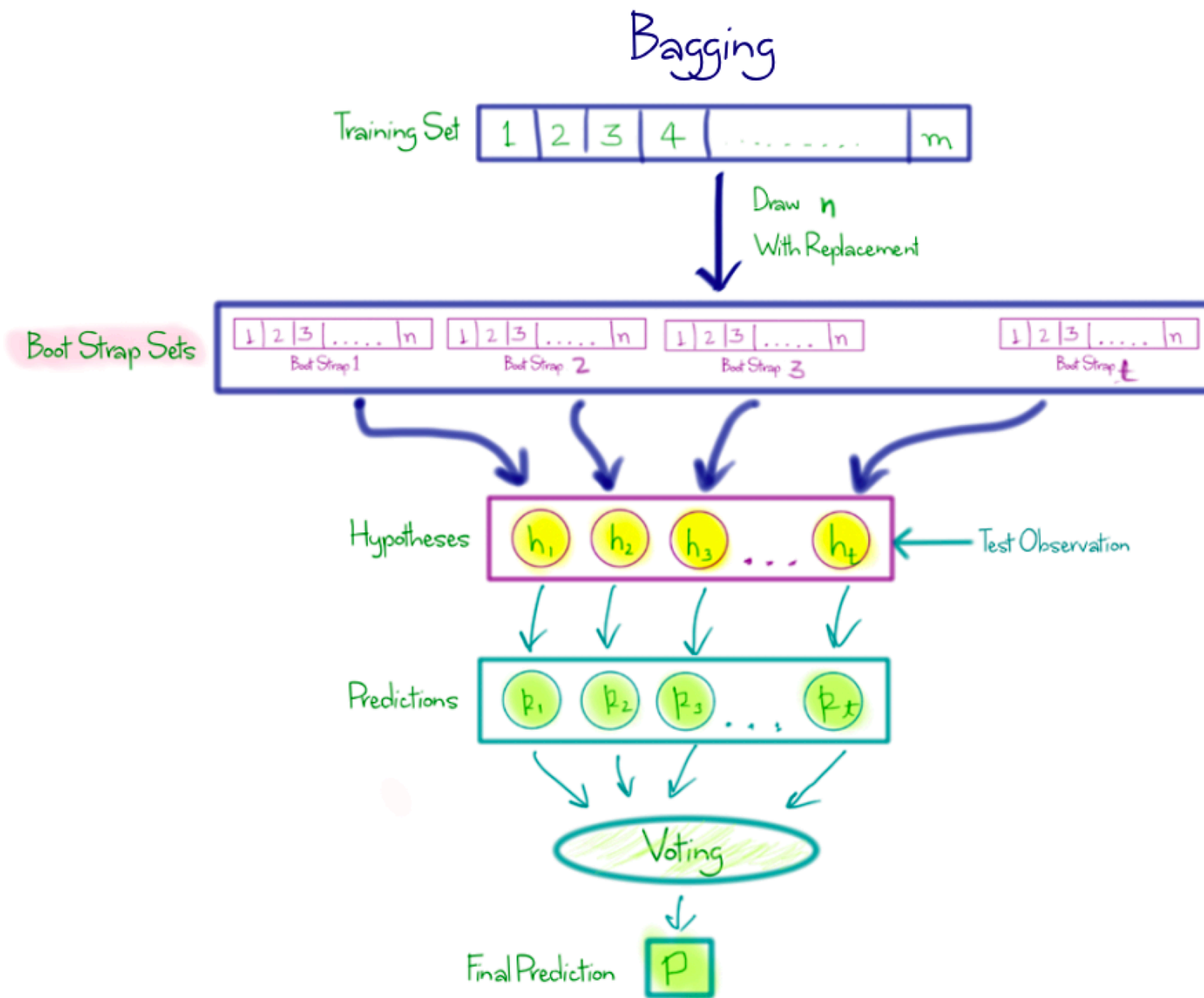
- **Bootstrap aggregation**, or **bagging**, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of  $n$  independent observations  $Z_1, \dots, Z_n$ , each with variance  $\sigma^2$ , the variance of the mean  $\bar{Z}$  of the observations is given by  $\sigma^2/n$ .
- 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.

# Bagging

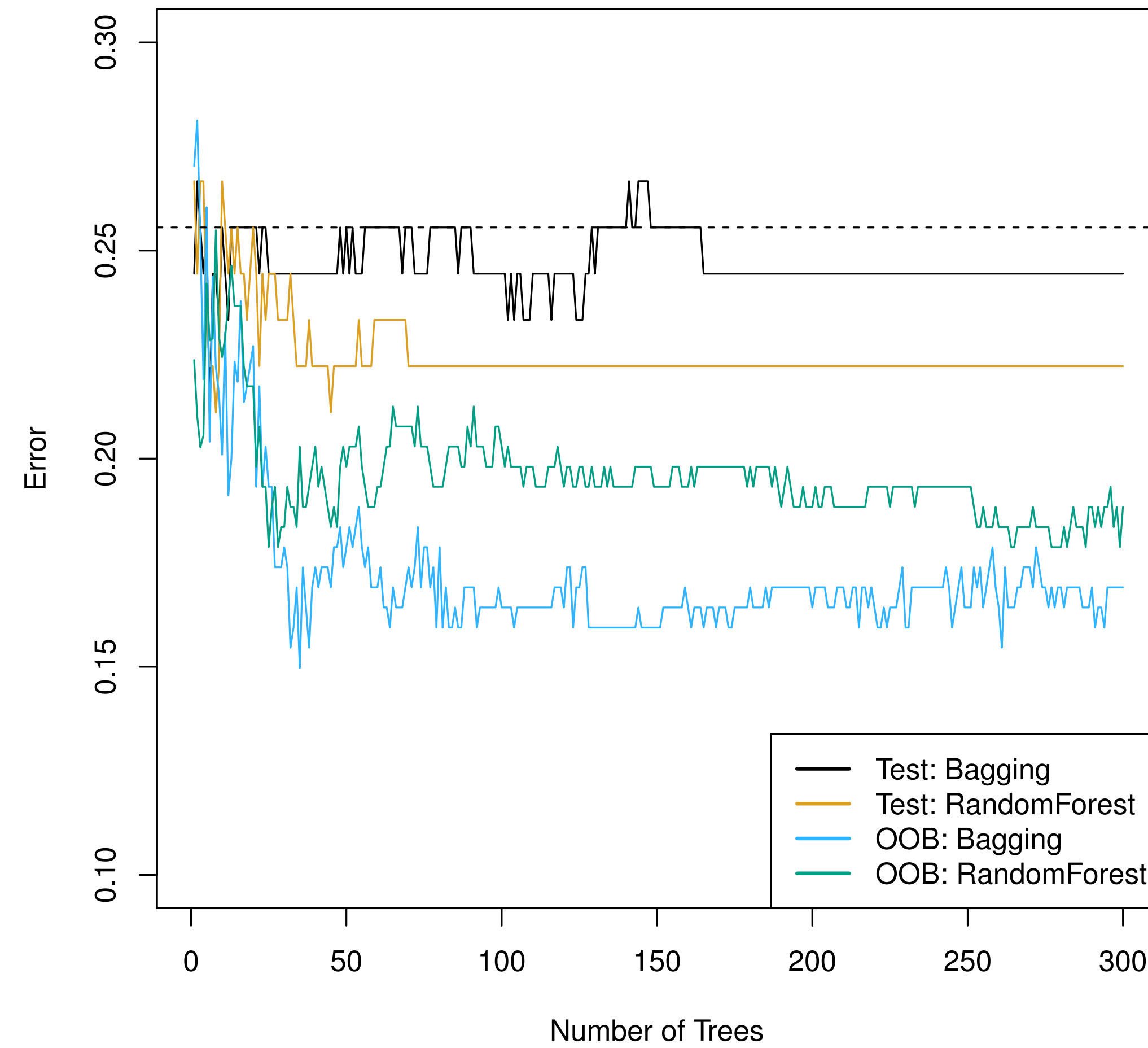
- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate  $B$  different **bootstrapped training data sets**. We then train our method on the  $b$  th bootstrapped training set in order to get  $\hat{f}^{*b}(x)$ , the prediction at a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

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

# Bagging



# Example





# 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  $m = p$ .
- 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

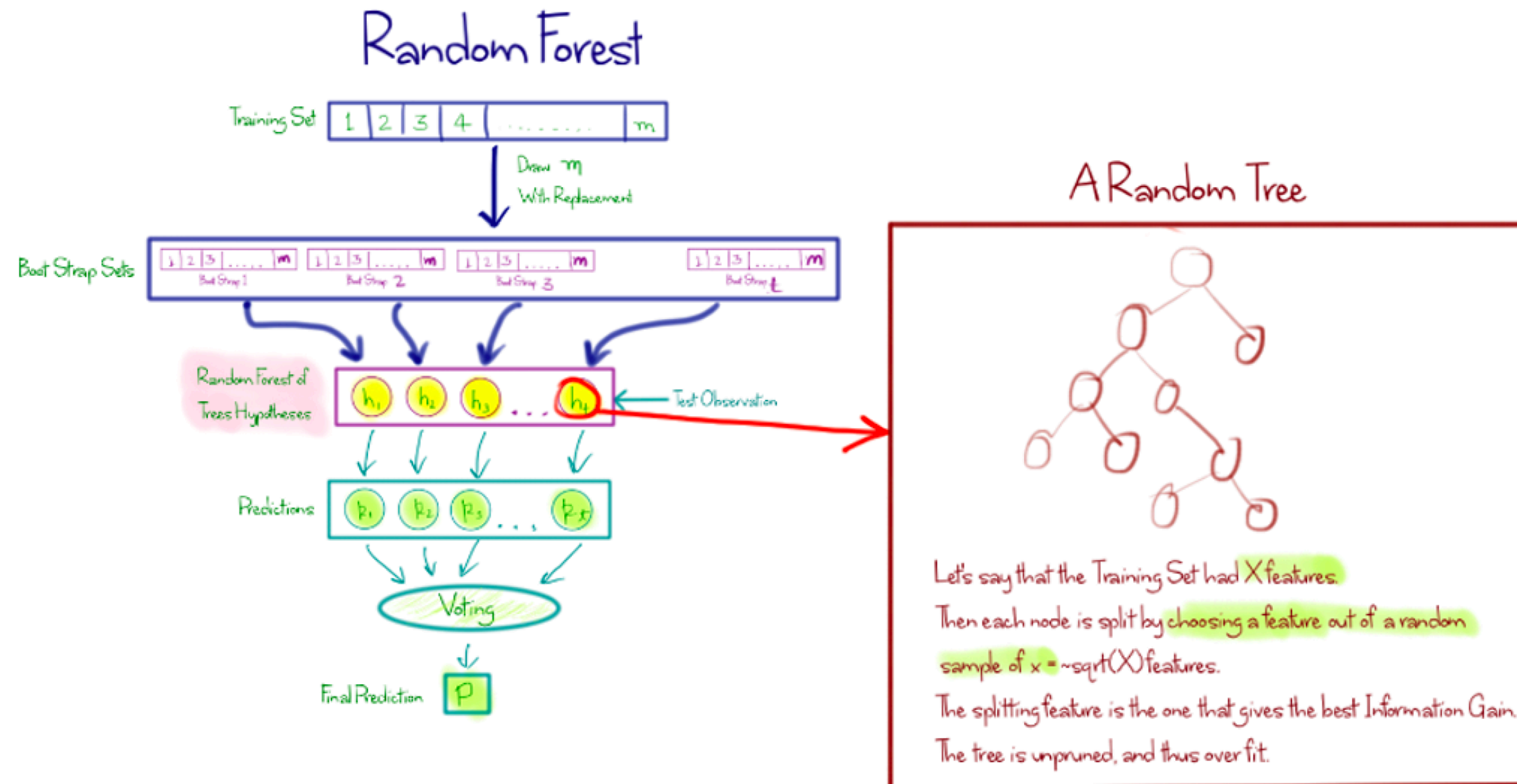
- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- 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  $i$ 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.

# Random Forest

# Random Forest

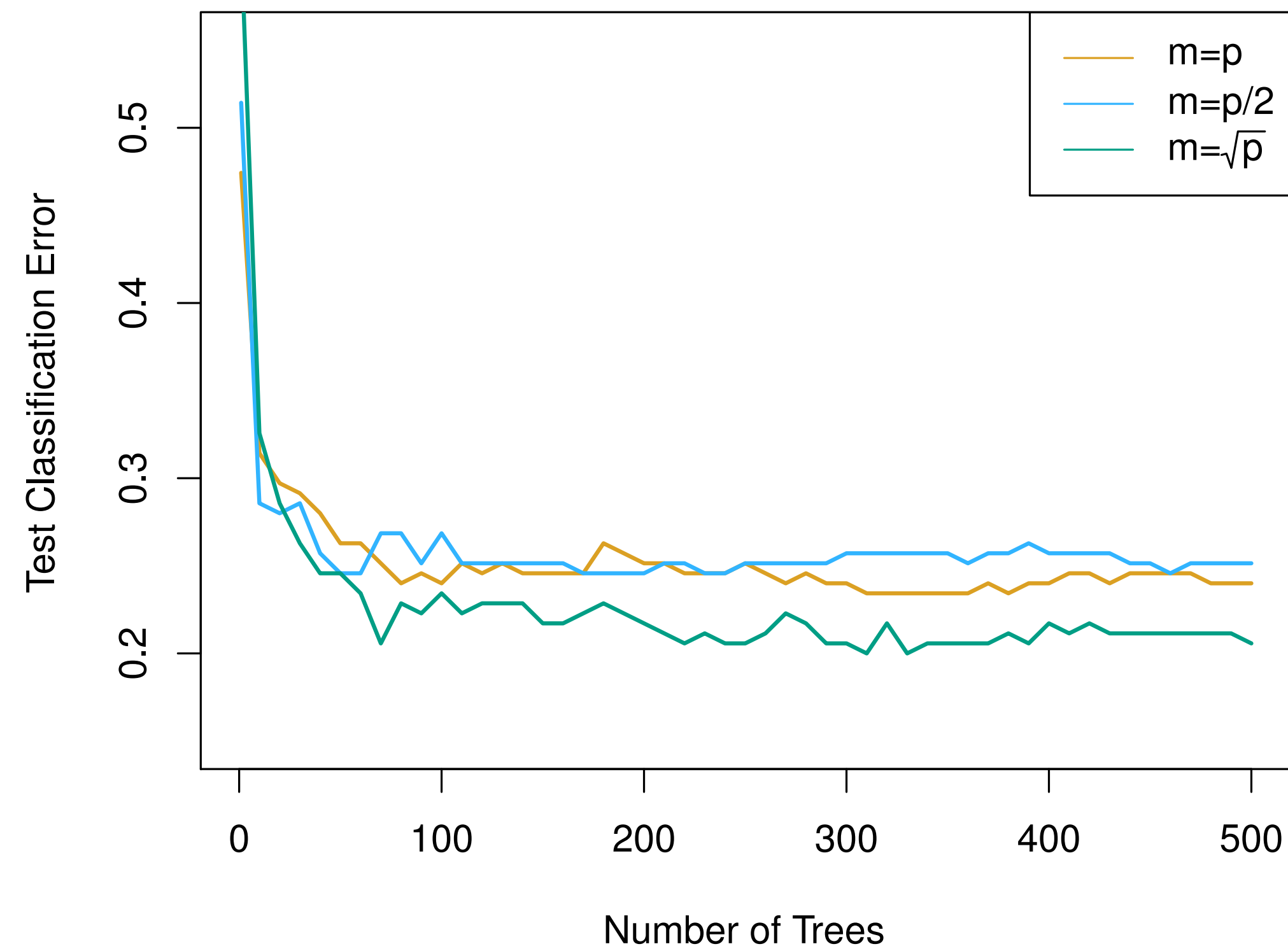
- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time 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.
- 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.

# Random Forest



# 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 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

# Boosting

- Boosting is suitable for regression or classification.
- 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 parameters

- The number of trees Boostings. Unlike bagging and random forests **boosting can overfit if Boosting is too large**, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
- The **shrinkage parameter**  $\lambda$ , 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  $\lambda$  can require using a very large value of B in order to achieve good performance.
- 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

# 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  $b = 1, 2, \dots, B$ , 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 shrunk version of the model tree:

$$\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}^b(x)$$

- 3 Update the residuals,

$$r_i = r_i - \lambda \hat{f}^b(x)$$

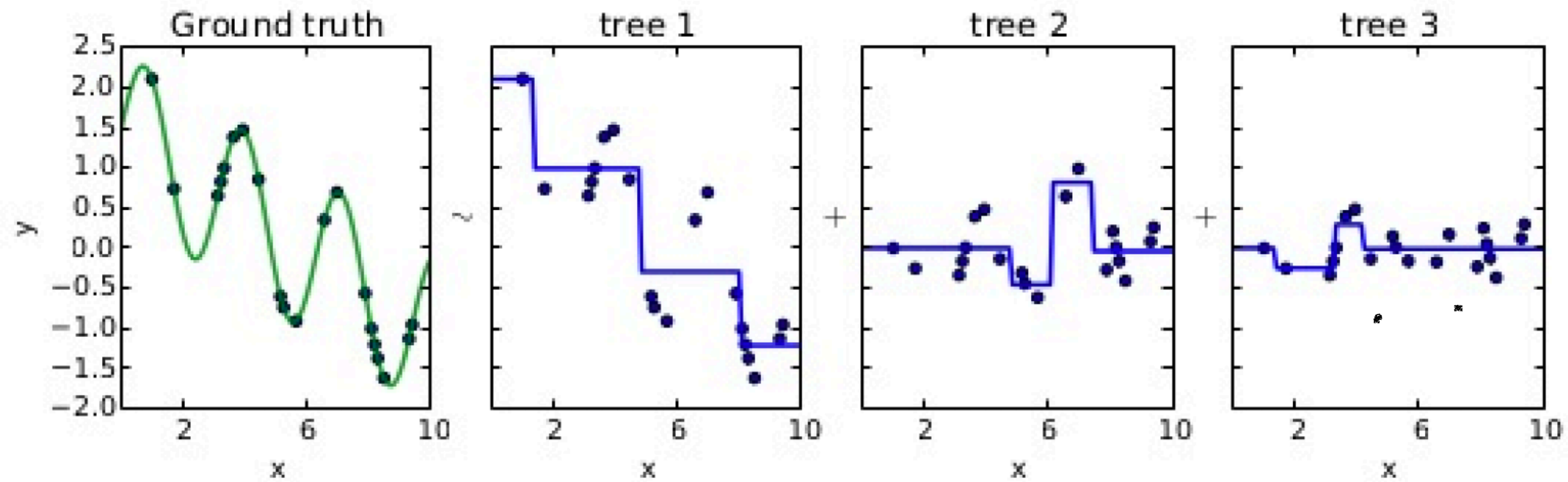
- 3 Output the boosted model,

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

# 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  $\lambda$  slows the process down even further, allowing more and different shaped trees to attack the residuals.

# Boosting





# Boosting for classification

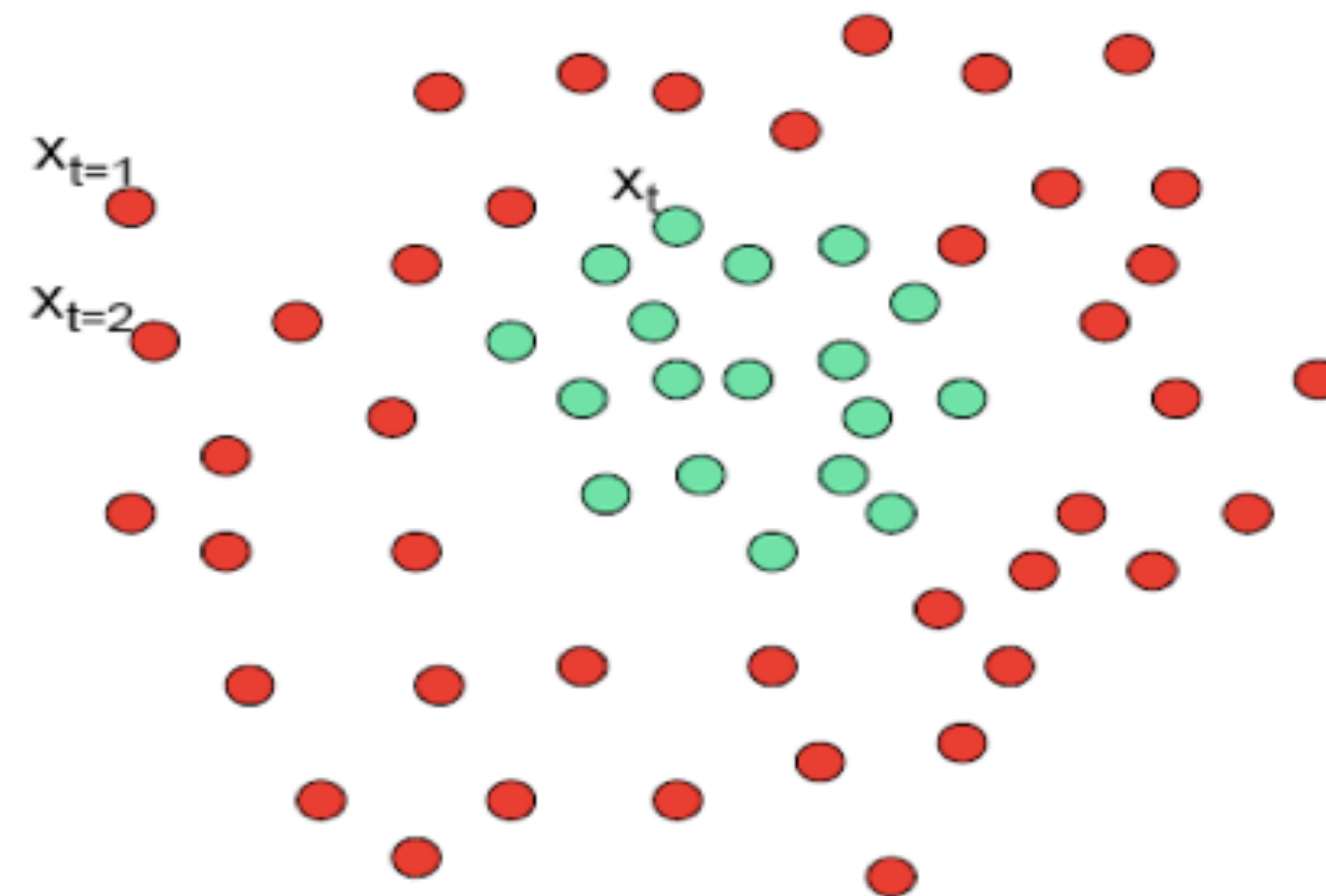
Boosting for classification is similar in spirit to boosting for regression. For binary classification:

- Training data:

$$(x_i, y_i), \dots, (x_m, y_m), \text{ where } x_i \in X, y_i \in Y = \{-1, 1\}$$

- $D_t(i)$ : the weight of  $x_i$  at round  $t$ .  $D_1(i) = 1/m$
- A learner  $L$  that finds a weak hypothesis  $h_t : y = f_w(x)$  given the training hypothesis  $D_t$ .
- The error of a weak hypothesis  $h_t$ :

$$\epsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$$

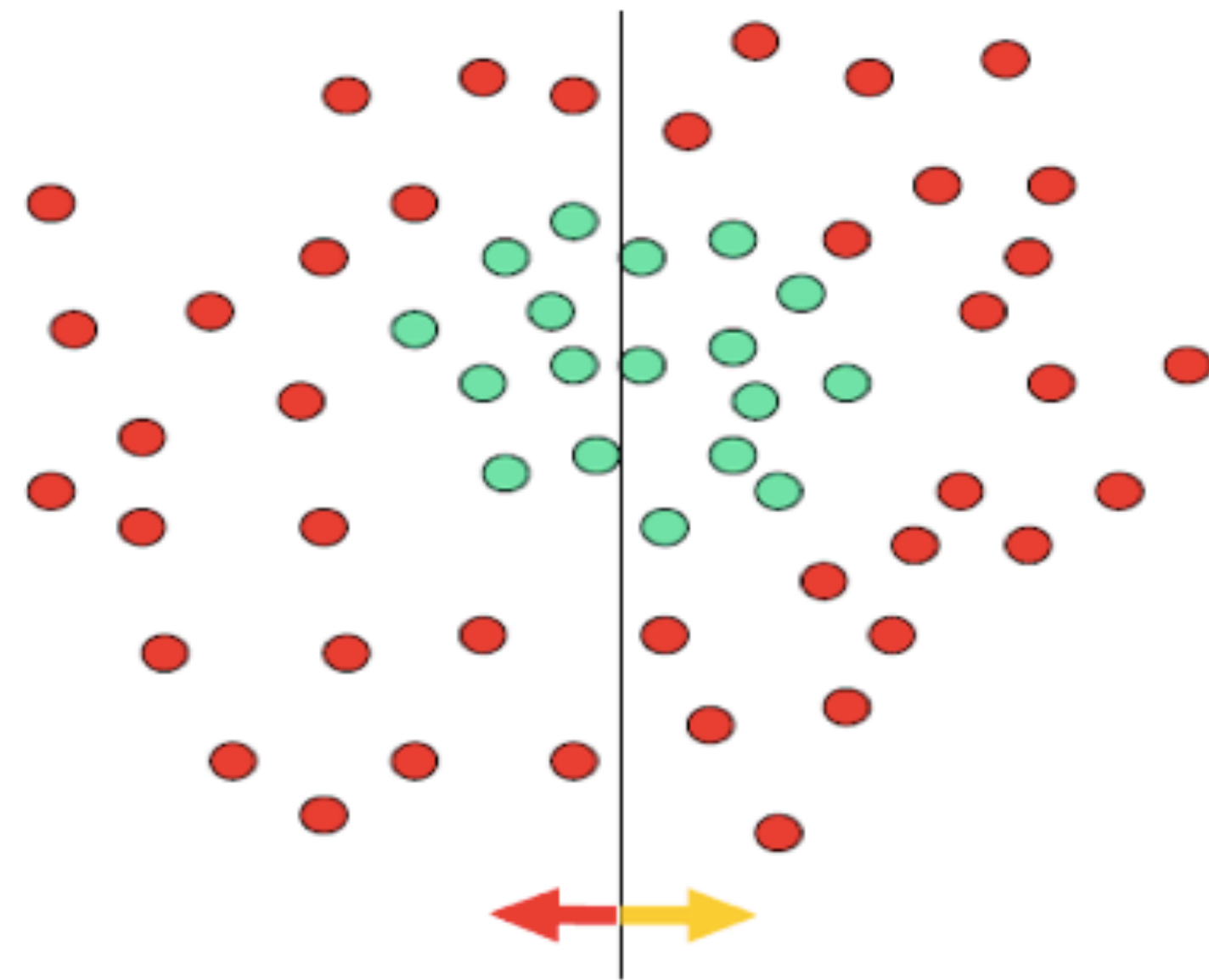


Each data point has  
a class label:

$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{green circle}) \end{cases}$$

and a weight:  
 $w_t = 1$





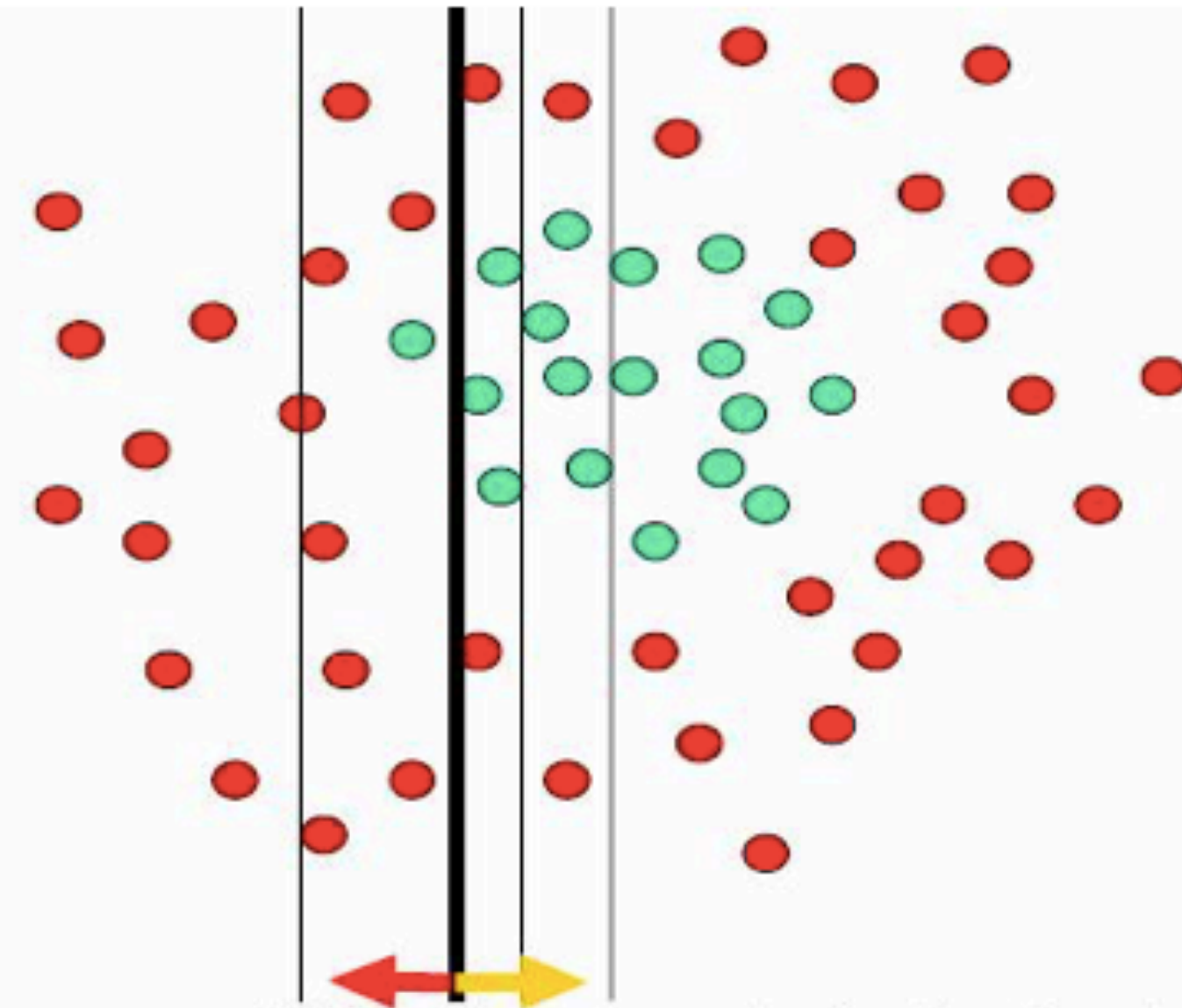
Each data point has  
a class label:

$$y_t = \begin{cases} +1 & (\text{red}) \\ -1 & (\text{green}) \end{cases}$$

and a weight:

$$w_t = 1$$

$h \Rightarrow p(\text{error}) = 0.5$  it is at chance



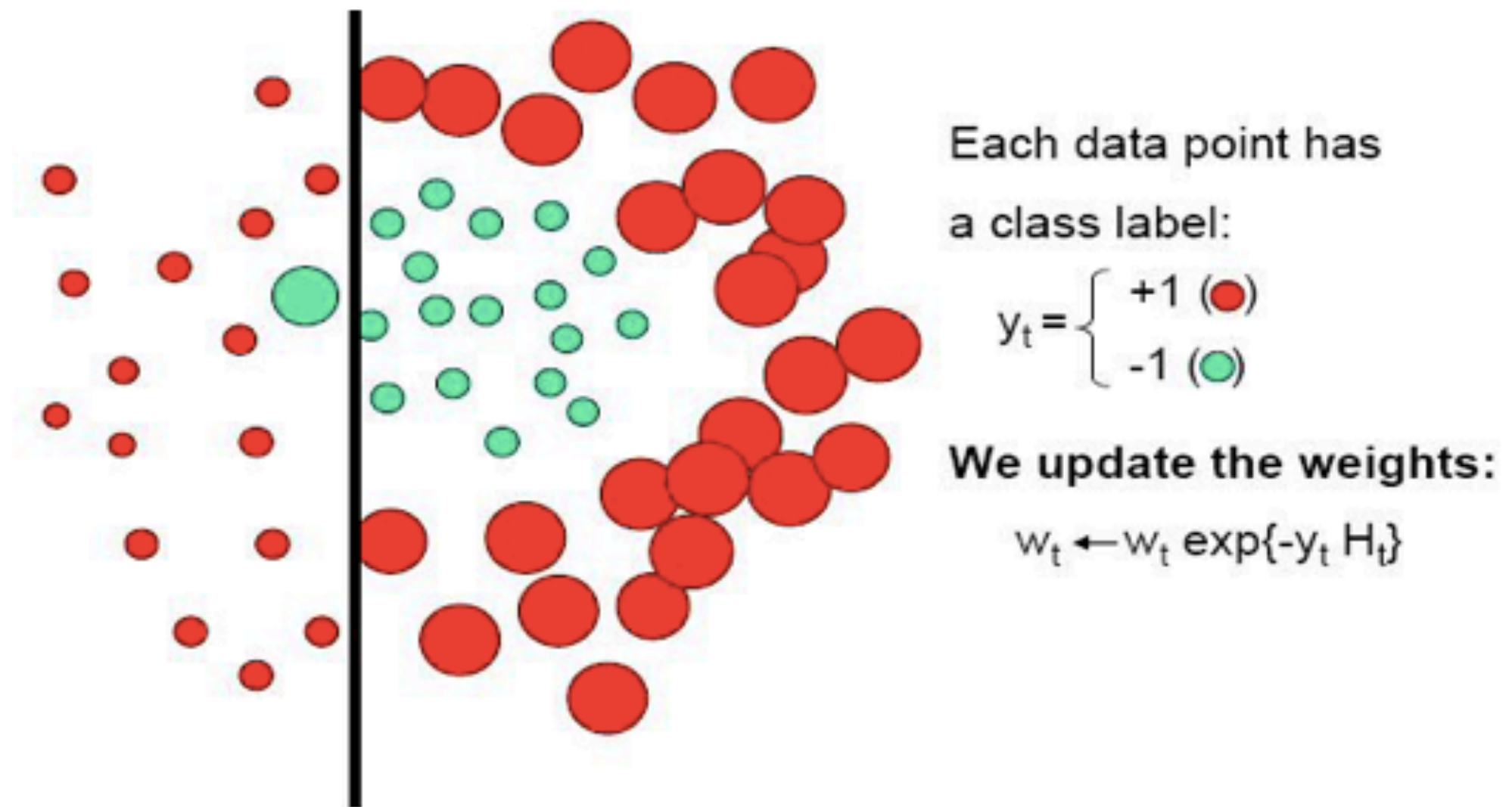
Each data point has  
a class label:

$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{green circle}) \end{cases}$$

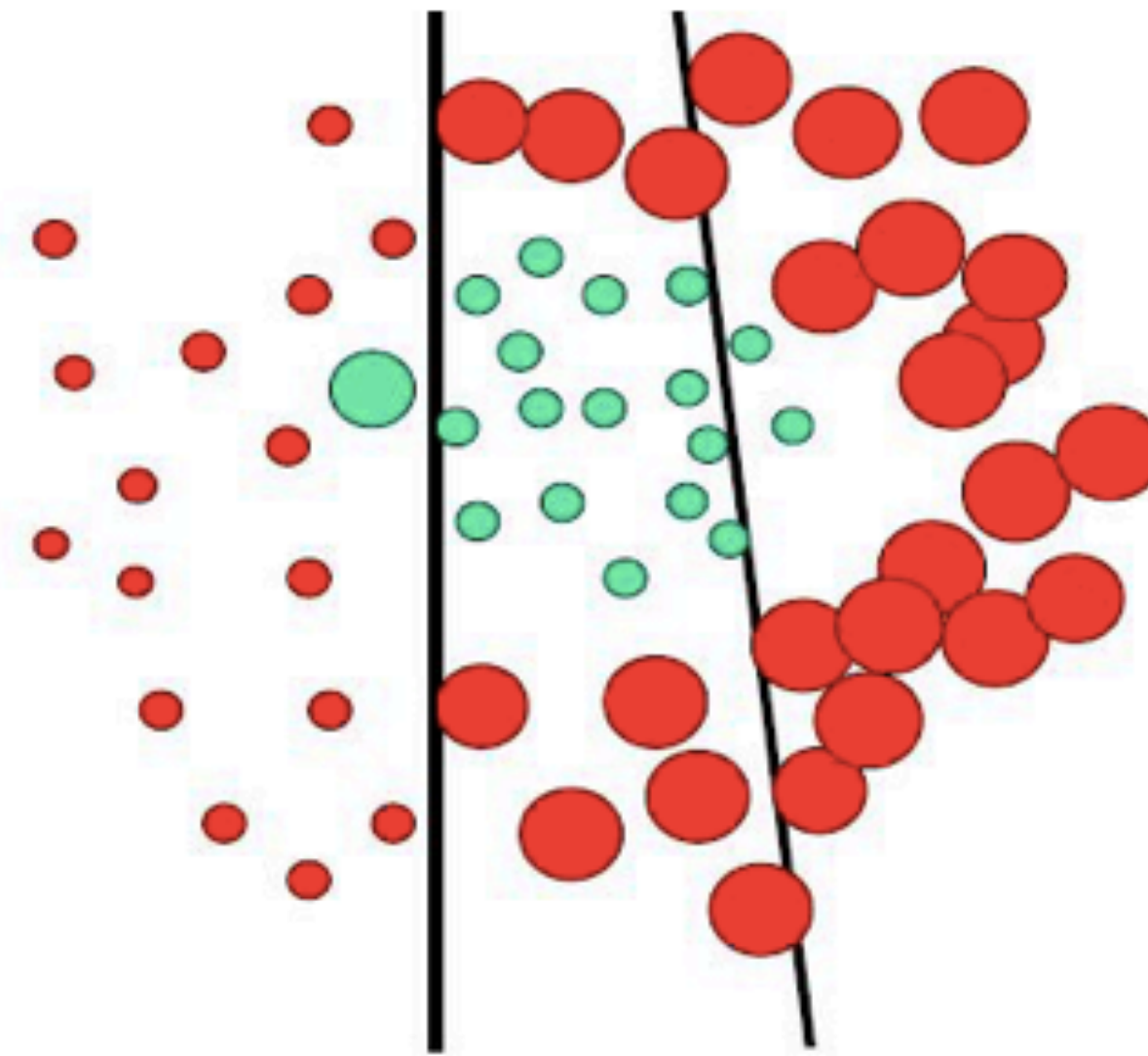
and a weight:  
 $w_t = 1$

This one seems to be the best

This is a '**weak classifier**': It performs slightly better than chance.



We set a new problem for which the previous weak classifier performs at chance again



Each data point has  
a class label:

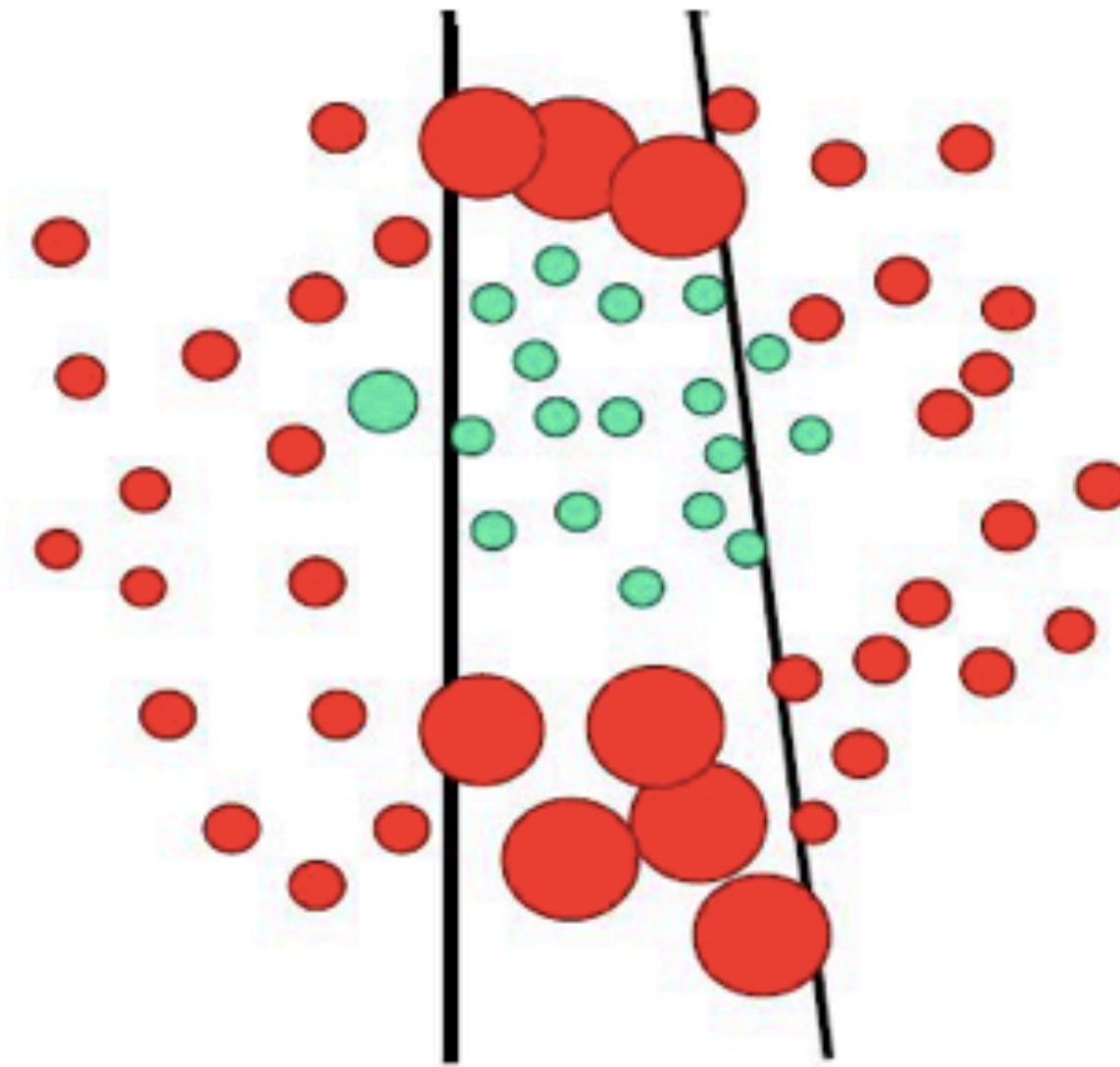
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{green circle}) \end{cases}$$

**We update the weights:**

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again





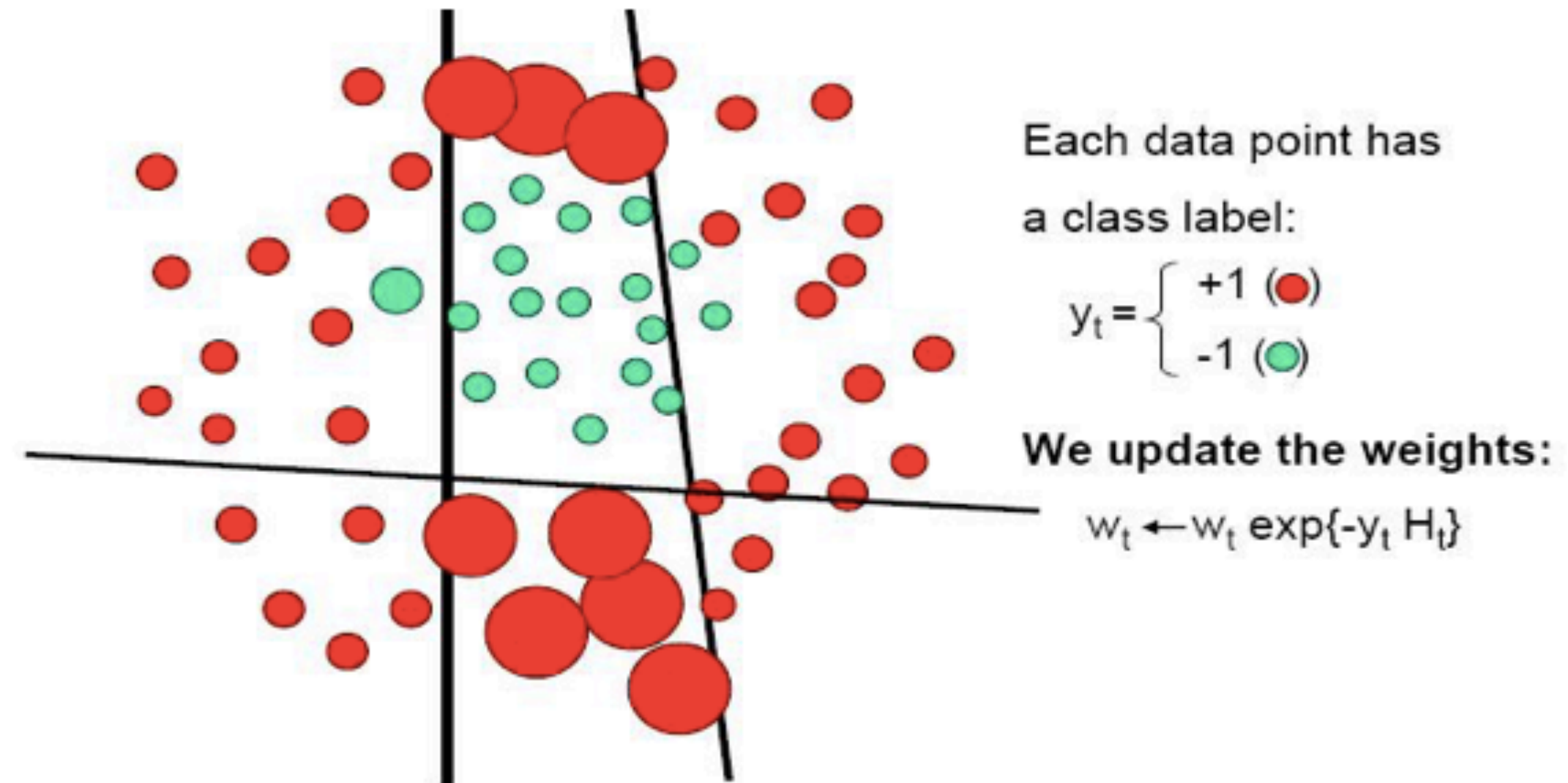
Each data point has  
a class label:

$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{green circle}) \end{cases}$$

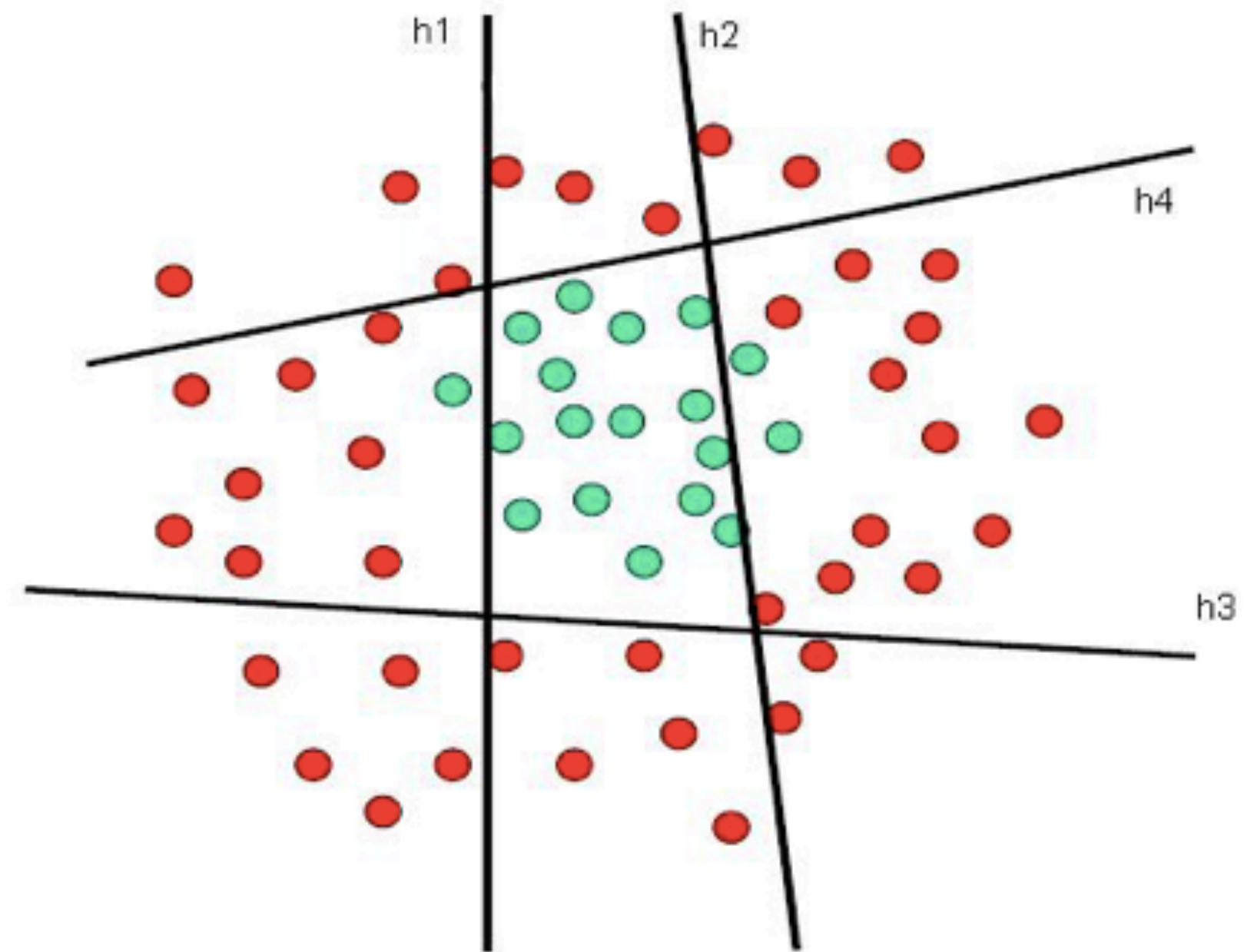
**We update the weights:**

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

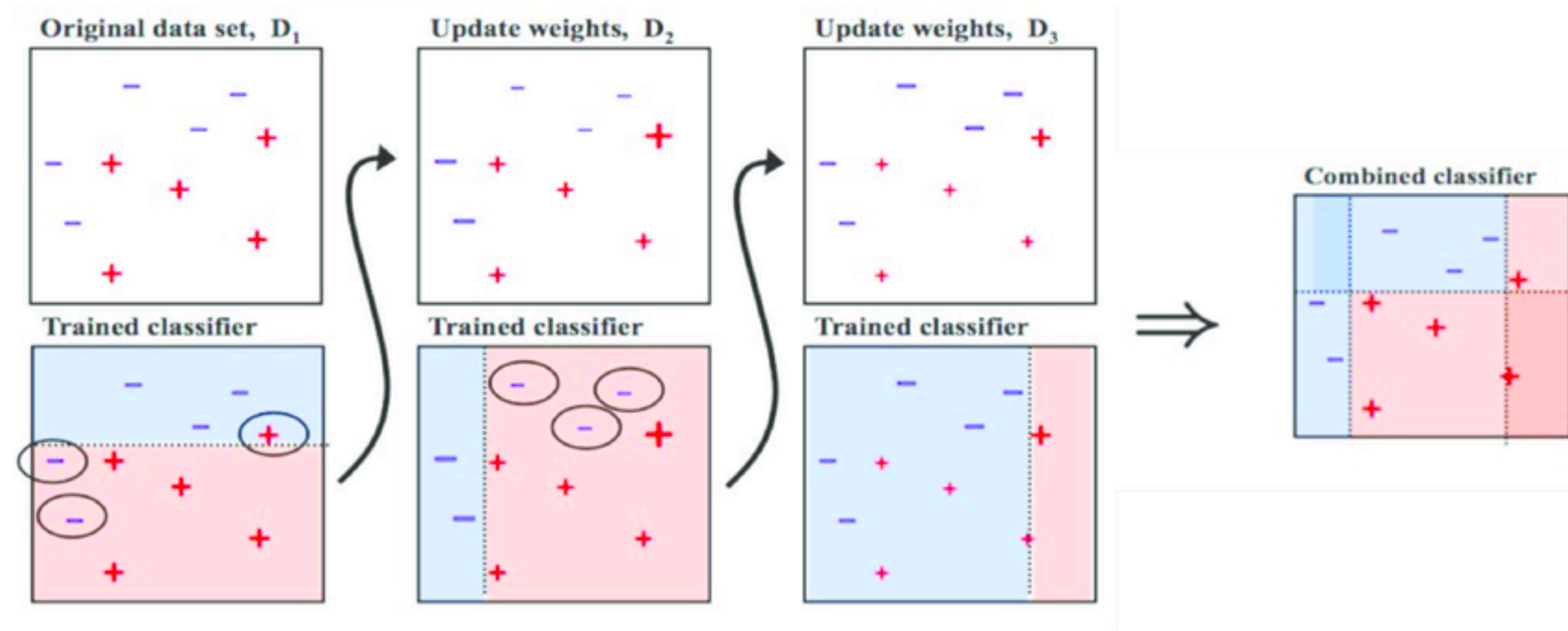


We set a new problem for which the previous weak classifier performs at chance again



The strong (non- linear) classifier is built as the combination of all the weak (linear) classifiers.

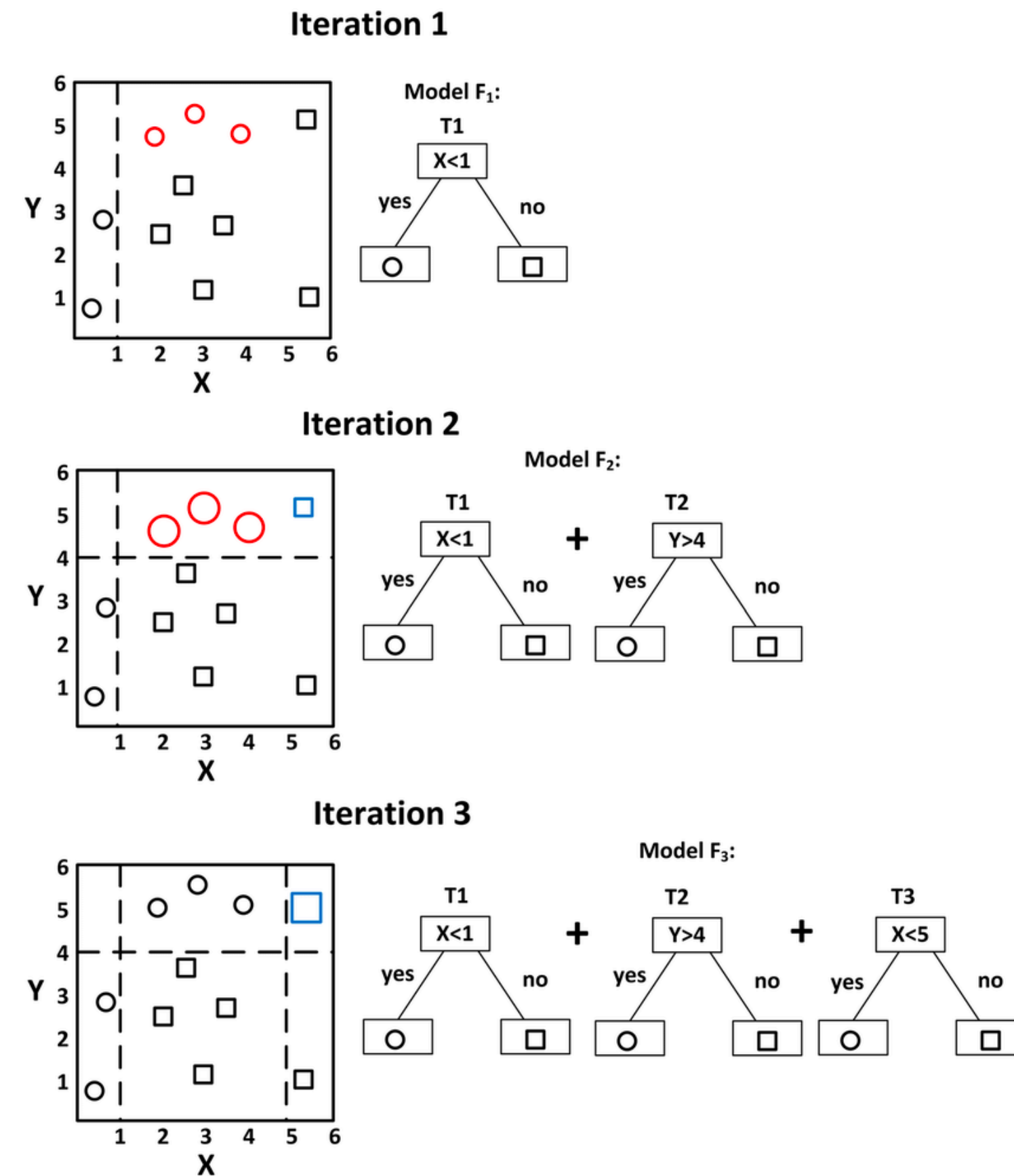
# Boosting



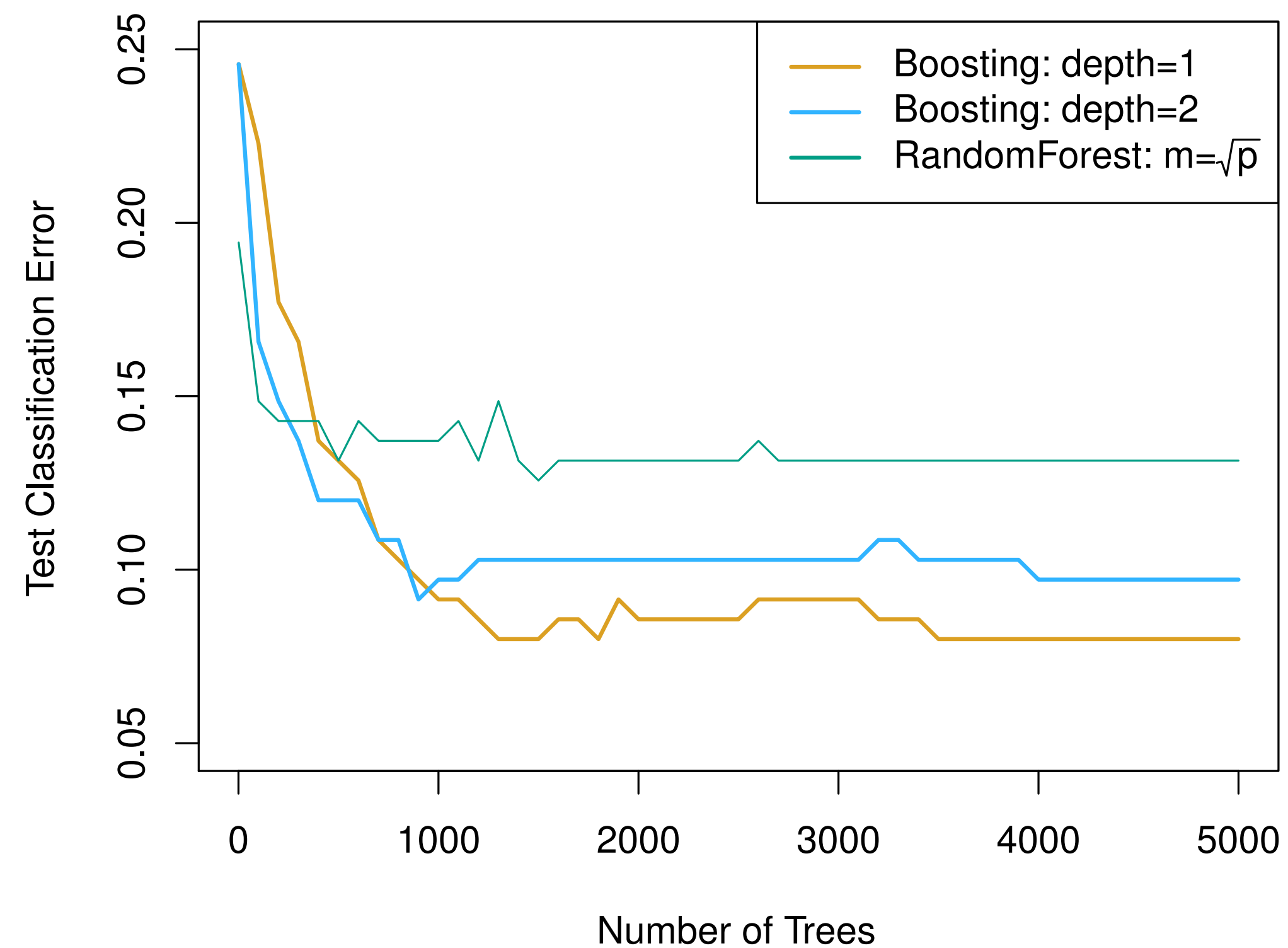
AdaBoost works on improving the areas where the base learner fails.



# Gradient Boosting

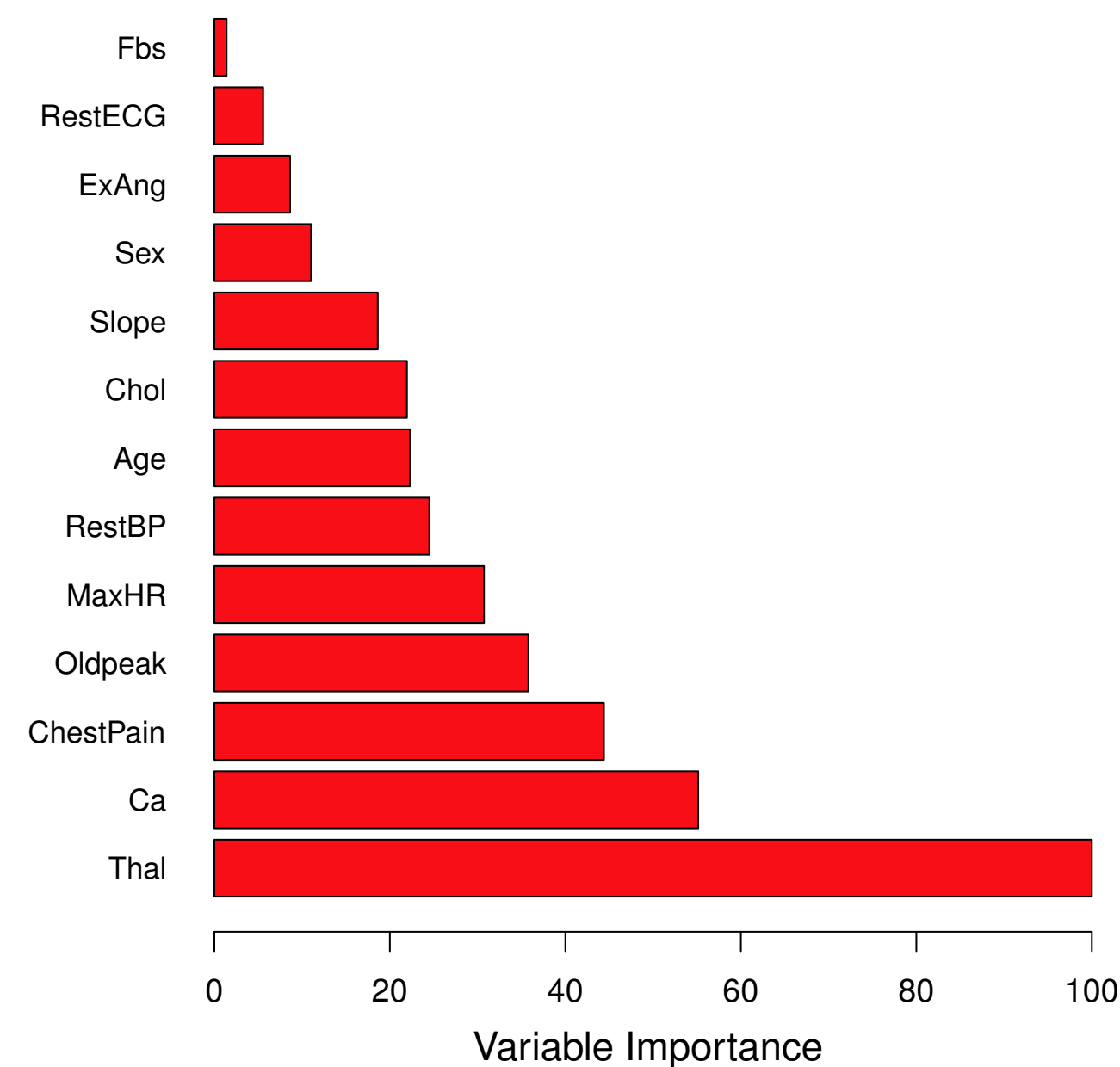


# Example: gene expression



# 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.



# 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.
- **Random forests** and **boosting** are among the **state-of-the-art methods for supervised learning**. However their results can be difficult to interpret.