

# Advanced Computational Econometrics: Machine Learning

## Chapter 5: Tree-based and ensemble methods

Thierry Denœux

Spring 2022



# Tree-based methods

- Here we describe **tree-based** methods for regression and classification.
- These involve **recursively 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 **tree-based** methods.



# Tree-based methods

- 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.
- Hence we also discuss two methods for combining several trees:
  - Bagging and
  - Random forests.

These methods grow 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 of interpretability.



# Regression and classification Trees

- The tree-based approach can be applied to both regression and classification problems.
- We first consider **regression trees**, and then move on to **classification/decision trees**.



# Overview

## 1 Introductory example

## 2 Learning a regression tree

- Tree building process
- Cost-complexity pruning

## 3 Classification trees

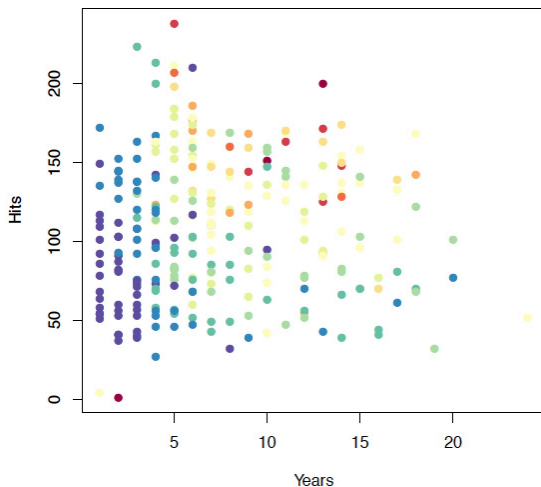
## 4 Combining trees

- Bootstrap
- Bagging
- Random Forests

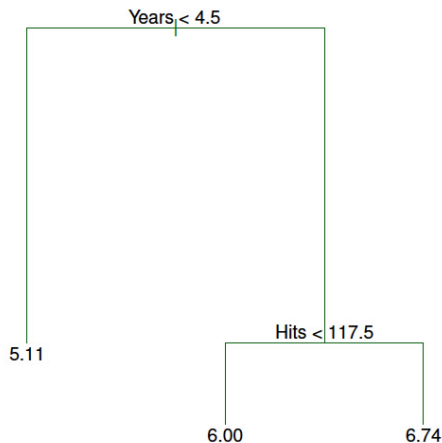


# Baseball salary data

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



## Regression tree for these data

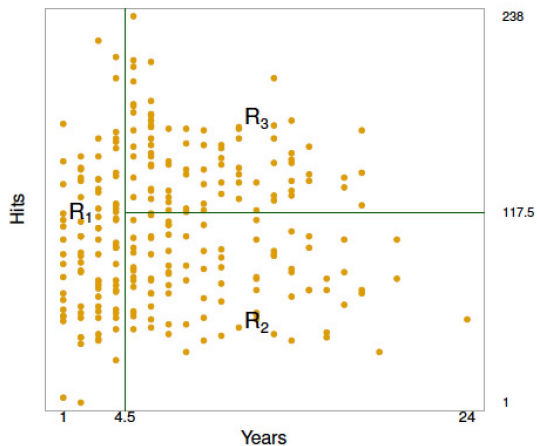


- At a given **internal node**, the label (of the form  $X_j < s$ ) indicates the left-hand branch emanating from the split, and the right-hand branch corresponds to  $X_j \geq s$ .
- The tree has two internal nodes and three **terminal nodes**, or **leaves**. The number in each leaf is the mean of the response for the observations that fall there.



# Results

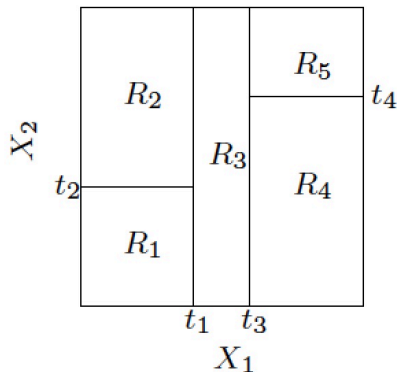
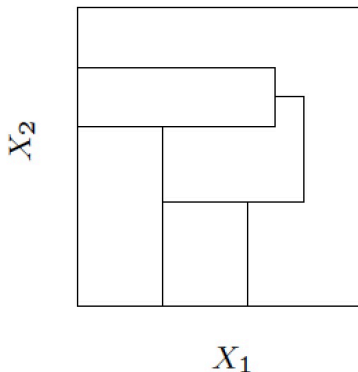
Overall, the tree stratifies or segments the players into three regions of predictor space:  $R_1 = \{X : \text{Years} < 4.5\}$ ,  $R_2 = \{X : \text{Years} \geq 4.5, \text{Hits} < 117.5\}$ , and  $R_3 = \{X : \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$ .





# 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.
- The prediction function is, thus, **stepwise constant**.



# Overview

- 1 Introductory example
- 2 Learning a regression tree
  - Tree building process
  - Cost-complexity pruning
- 3 Classification trees
- 4 Combining trees
  - Bootstrap
  - Bagging
  - Random Forests



# Overview

- 1 Introductory example
- 2 Learning a regression tree
  - Tree building process
  - Cost-complexity pruning
- 3 Classification trees
- 4 Combining trees
  - Bootstrap
  - Bagging
  - Random Forests



# Growing a regression tree

- We now turn to the question of how to **grow a regression tree**.
- Our training set consists of  $p$  predictors and a response, for each of  $n$  observations: that is,  $(x_i, y_i)$  for  $i = 1, 2, \dots, n$ , with  $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})$ .
- The algorithm needs to automatically decide on
  - 1 The splitting variables and split points
  - 2 What topology (shape) the tree should have.



## Predicted response in a given region

- Suppose first that we have a partition into  $M$  regions  $R_1, R_2, \dots, R_M$ , and we model the response as a constant  $c_m$  in each region:

$$f(x) = \sum_{m=1}^M c_m I(x \in R_m)$$

- If we adopt as our criterion minimization of the RSS error, we have

$$\text{RSS}(c_1, \dots, c_M) = \sum_{i=1}^n (y_i - f(x_i))^2 = \sum_{m=1}^M \sum_{\{i: x_i \in R_m\}} (y_i - c_m)^2$$

The LS estimate of  $c_m$  is, thus, the **average** of  $y_i$  in region  $R_m$ :

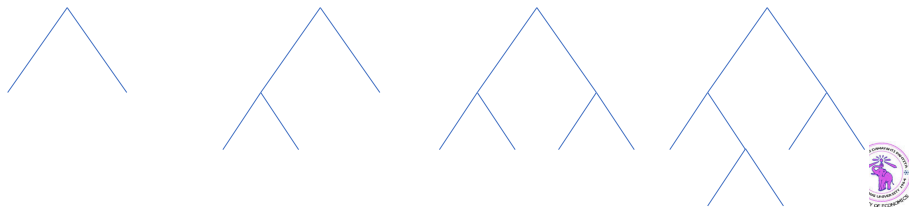
$$\hat{c}_m = \text{Ave} \{y_i : x_i \in R_m\}.$$



## Top-down, greedy approach

Now, finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence we proceed with a **top-down, greedy** approach:

- 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.



## Selecting a splitting variable and split point

- Starting with all of the data, consider a splitting variable  $X_j$  and split point  $s$ , and define the pair of half-spaces

$$R_1(j, s) = \{X : X_j \leq s\} \text{ and } R_2(j, s) = \{X : X_j > s\}$$

- Then we seek the splitting variable  $X_j$  and split point  $s$  that solve

$$\min_{j,s} \left[ \sum_{x_i \in R_1(j,s)} (y_i - \hat{c}_1(j,s))^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{c}_2(j,s))^2 \right]$$

with

$$\hat{c}_1(j, s) = \text{Ave}\{y_i : x_i \in R_1(j, s)\} \text{ and } \hat{c}_2(j, s) = \text{Ave}\{y_i : x_i \in R_2(j, s)\}$$

- For each splitting variable, the determination of the split point  $s$  can be done very quickly and hence by scanning through all of the predictors, determination of the best pair  $(j, s)$  is feasible.



# Recursive building process

- Having found the best split, we partition the data into the two resulting regions and **repeat the splitting process** on each of the two regions.
- Then this process is repeated on all of the resulting regions.
- How large should we grow the tree? Clearly a very large tree has a large variance and might overfit the data, while a small tree has a large bias and might not capture the important structure. (More on this later).





# Regression trees in R

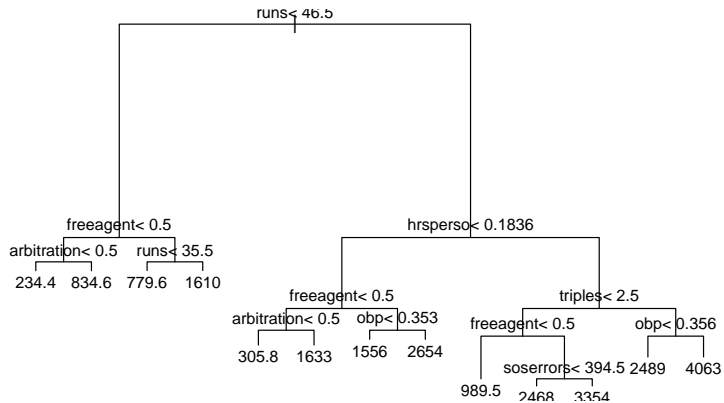
```
library(rpart)
baseball <- read.table("baseball.dat",header=TRUE)
n<-nrow(baseball)

train = sample(n, 2*n/3)
fit<-rpart(salary~.,data=baseball,subset=train,method="anova")

plot(fit)
text(fit,pretty=0,cex=0.8)
```



# Regression trees in R



# Overview

- 1 Introductory example
- 2 Learning a regression tree
  - Tree building process
  - Cost-complexity pruning
- 3 Classification trees
- 4 Combining trees
  - Bootstrap
  - Bagging
  - Random Forests



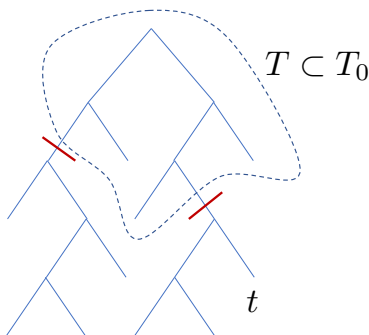
# Tuning the model's complexity

- **Tree size** is a tuning parameter governing the model's complexity, and the optimal tree size should be adaptively chosen from the data.
- One approach would be to split tree nodes only if the decrease in sum-of-squares due to the split exceeds some threshold. This strategy is too short-sighted, however, since a seemingly worthless split might lead to a very good split below it.
- The preferred strategy is to grow a large tree  $T_0$ , stopping the splitting process only when some minimum node size (say 5) is reached. Then this large tree is pruned using **cost-complexity pruning**.



# Notations

- We define a **subtree**  $T \subset T_0$  to be any tree that can be obtained by pruning  $T_0$ , that is, collapsing any number of its internal nodes.
- We index terminal nodes by  $t$ , with node  $t$  representing region  $R_t$ .



# Cost-complexity criterion

- Let  $\tilde{T}$  denote the set of terminal nodes in  $T$ . Let

$$n_t = \#\{x_i \in R_t\}, \quad \hat{c}_t = \frac{1}{n_t} \sum_{x_i \in R_t} y_i,$$

$$\text{RSS}(T) = \sum_{t \in \tilde{T}} \sum_{x_i \in R_t} (y_i - \hat{c}_t)^2$$

- We define the **cost-complexity criterion** as

$$C_\lambda(T) = \text{RSS}(T) + \lambda |\tilde{T}|$$



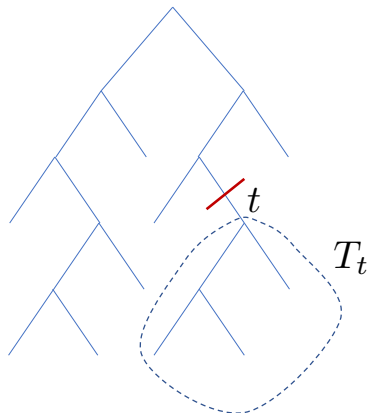
# Cost-complexity pruning

- The idea is to find, for each  $\lambda$ , the subtree  $T(\lambda) \subseteq T_0$  that minimizes  $C_\lambda(T)$ .
- The tuning parameter  $\lambda \geq 0$  governs the tradeoff between tree size and its goodness of fit to the data. Larger values of  $\lambda$  result in smaller trees  $T(\lambda)$ .
- For  $\lambda = 0$ , the solution is the full tree  $T_0$ .
- Questions:
  - 1 For given  $\lambda$ , how to find a tree that minimizes  $C_\lambda(T)$ ?
  - 2 How to choose  $\lambda$ ?



# Weakest link

- We start from the full tree  $T_0$ .
- For any internal node  $t$ , let  $T_t$  be the branch of  $T$  with root  $t$ .





## Weakest link (continued)

- If we prune  $T_t$ , the cost-complexity criterion becomes smaller if

$$\text{RSS}(t) + \lambda < \text{RSS}(T_t) + \lambda|\tilde{T}_t| \Leftrightarrow \lambda > \frac{\text{RSS}(t) - \text{RSS}(T_t)}{|\tilde{T}_t| - 1} = g_0(t)$$

- The **weakest link**  $t_0$  in  $T_0$  is the node such that  $g_0(t_0) = \min_t g_0(t)$ . Let  $\lambda_1 = g_0(t_0)$ .
- Meaning: if we increase  $\lambda$  starting from 0,  $t_0$  is the first node  $t$  such that pruning  $T_t$  improves the cost-complexity criterion.
- Let  $T_1 = T_0 - T_{t_0}$ . We again find the weakest link  $t_1$  in  $T_1$ , etc.



# Sequence of optimal trees

- By iterating the above process until the tree is reduced to the root node  $t_{root}$ , we get a decreasing sequence of trees

$$T_0 \supset T_1 \supset \dots \supset t_{root},$$

and an increasing sequence of  $\lambda$  values,  $0 = \lambda_0 < \lambda_1 < \lambda_2 < \dots$

- We can show that, for all  $k \geq 0$  and all  $\lambda \in [\lambda_k, \lambda_{k+1})$ , the optimum tree  $T(\lambda)$  is equal to  $T_k$ .



# Choosing $\lambda$

- If we have a lot of data, it is easy to estimate the sum-of-squares error of each subtree in the sequence  $T_0 \supset T_1 \supset \dots \supset t_{root}$  using a **validation set**. We choose the tree  $T_k$  with minimum validation error.
- Otherwise, we use **cross-validation**.



# Cross-validation

- Using the whole training set, we get a sequence of trees,  $T_0 \supset T_1 \supset \dots \supset t_{root}$ , where  $T_k$  is the best tree for  $\lambda_k \leq \lambda < \lambda_{k+1}$ .
- For  $k = 0, 1, 2, \dots$ , set  $\beta_k = \sqrt{\lambda_k \lambda_{k+1}}$
- Assume we use  $K$ -fold cross validation: we partition the training data in  $K$  subsets of approximately equal size.
- We construct  $K$  sequences of trees by leaving each of  $K$  subsets out and building the trees using the  $K - 1$  remaining subsets. Let  $T_0^{(r)} \supset T_1^{(r)} \supset \dots \supset t_{root}^{(r)}$  be the sequence of trees obtained by leaving subset  $r$  out.
- Compute the cross-validated error  $RSS_{cv}(T_k)$  by averaging the errors of trees  $T^{(r)}(\beta_k)$ ,  $r = 1, \dots, K$ .
- Select the tree  $T_k$  corresponding to the minimum cross-validated error.



# Pruning a regression tree in R

```
fit<-rpart(salary~.,data=baseball,subset=train,method="anova",  
control = rpart.control(xval = 10, minbucket = 2, cp = 0))
```

```
printcp(fit)
```

```
plotcp(fit)
```



# Pruning a regression tree in R

```
> printcp(fit)
```

Regression tree:

```
rpart(formula = salary ~ ., data = baseball, subset = train,
      method = "anova", control = rpart.control(xval = 10, minbucket = 2,
      cp = 0))
```

Variables actually used in tree construction:

```
[1] arbitration average doubles errors freeagent hits
[7] hitspererror hitsperso homeruns hrsperso obp rbis
[13] rbisperso runs runsperso sbsobp sbsruns sos
[19] soserros triples walks walksperso
```

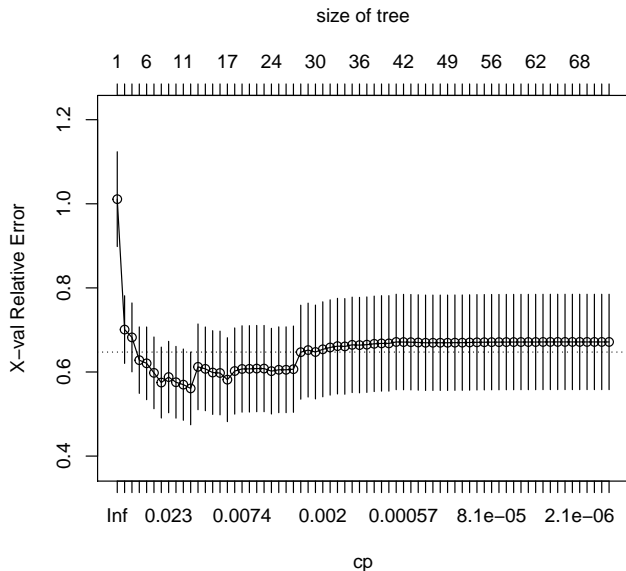
Root node error: 364623518/224 = 1627784

n= 224

	CP	nsplit	rel error	xerror	xstd
1	3.6931e-01	0	1.000000	1.01101	0.112712
2	1.2284e-01	1	0.630689	0.70087	0.080434
3	6.0838e-02	2	0.507850	0.68227	0.081909
4	4.2082e-02	4	0.386175	0.62816	0.078994
5	3.6111e-02	5	0.344093	0.62059	0.086438
6	3.3694e-02	6	0.307982	0.59791	0.085306
7	2.5680e-02	7	0.274289	0.57487	0.084283
8	2.1447e-02	8	0.248609	0.58795	0.084931
9	1.6639e-02	9	0.227162	0.57559	0.085300
10	1.5463e-02	10	0.210522	0.56992	0.084713
11	1.5064e-02	11	0.195059	0.56094	0.086454
12	1.4133e-02	12	0.179996	0.61223	0.101958
13	1.3472e-02	13	0.165862	0.60748	0.099526
14	1.2588e-02	14	0.152390	0.59866	0.099545
15	1.2339e-02	15	0.139802	0.59761	0.099555

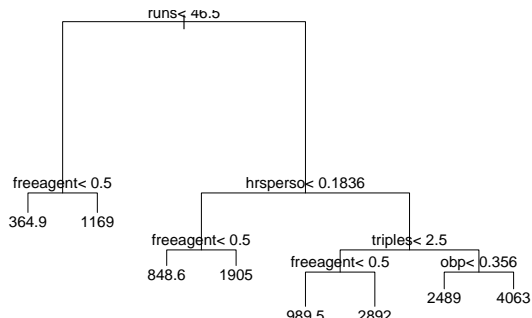


# Pruning a regression tree in R



# Pruning a regression tree in R

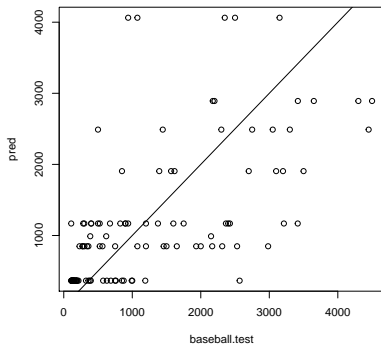
```
pruned_tree<-prune(fit, cp=2.5680e-02)
plot(pruned_tree)
text(pruned_tree, pretty=0)
```





# Prediction with a regression tree in R

```
yhat=predict(pruned_tree,newdata=baseball[-train,])  
baseball.test=baseball[-train,"salary"]  
plot(baseball.test,yhat)  
abline(0,1)
```



# Overview

- 1 Introductory example
- 2 Learning a regression tree
  - Tree building process
  - Cost-complexity pruning
- 3 Classification trees
- 4 Combining trees
  - Bootstrap
  - Bagging
  - Random Forests



# Classification trees

- If the response is a categorical variable (factor) taking  $c$  values indexed by  $1, 2, \dots, c$ , we have a **classification problem**.
- The only changes needed in the tree-growing algorithm concern the criteria for splitting nodes and pruning the tree.
- In classification, each split will aim at obtaining nodes as “**pure**” as possible (a node is pure if it contains observations from only one class). For that, we need to define a suitable **impurity measure**.



# Notations

- In a node  $t$ , let

$$\hat{p}_{tk} = \frac{1}{n_t} \sum_{x_i \in R_t} I(y_i = k)$$

be the proportion of class  $k$  observations in node  $t$ .

- We have  $\sum_{k=1}^c \hat{p}_{tk} = 1$ .
- We classify the observations in node  $t$  to the **majority class** in that node:

$$k(t) = \arg \max_k \hat{p}_{tk},$$



# Impurity measures

- Different measures  $Q_t$  of node impurity include the following:  
Misclassification error:

$$Q_t^{\text{mis}} = \frac{1}{n_t} \sum_{x_i \in R_t} I(y_i \neq k(t)) = 1 - \hat{p}_{tk(t)}$$

Gini index:

$$Q_t^{\text{Gini}} = \sum_{k=1}^c \hat{p}_{tk}(1 - \hat{p}_{tk})$$

Entropy:

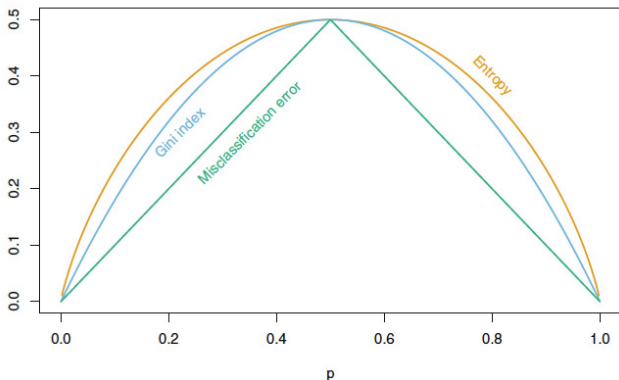
$$Q_t^{\text{ent}} = - \sum_{k=1}^c \hat{p}_{tk} \log \hat{p}_{tk}$$

- All three criteria are equal to 0 when  $\hat{p}_{tk} = 1$  for some  $k$ , and are maximum when  $\hat{p}_{tk} = 1/c$  for all  $k$ .



# Comparison between impurity measures

- Plot as a function of the proportion  $p$  of one class in the case  $c = 2$ :



- All three are similar, but entropy and the Gini index are differentiable and hence more amenable to numerical optimization.



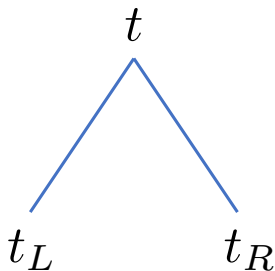
## Comparison between impurity measures

- In addition, entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate.
- For example, in a two-class problem with 400 observations in each class (denote this by  $(400, 400)$ ), suppose one split created nodes  $(300, 100)$  and  $(100, 300)$ , while the other created nodes  $(200, 400)$  and  $(200, 0)$ .
- Both splits produce a misclassification rate of 0.25, but the second split produces a pure node and is probably preferable. Both the Gini index and entropy are lower for the second split. For this reason, **either the Gini index or entropy should be used when growing the tree.**
- To guide cost-complexity pruning, any of the three measures can be used, but typically we use the misclassification rate.



# Selecting the best split

- Consider a node  $t$  with size  $n_t$  with impurity  $Q_t$ .
- For some variable  $j$  and split point  $s$ , we split  $t$  in two nodes,  $t_L$  and  $t_R$ , with sizes  $n_{t_L}$  and  $n_{t_R}$ , and with impurities  $Q_{t_L}$  and  $Q_{t_R}$ .





# Selecting the best split

- The **average decrease of impurity** is

$$\Delta(j, s) = Q_t - \left( \frac{n_{tL}}{n_t} Q_{tL} + \frac{n_{tR}}{n_t} Q_{tR} \right)$$

- If  $Q_t$  is the entropy, then  $\Delta(j, s)$  is interpreted as an **information gain**.
- We select at each step the splitting variable  $j$  and the split point  $s$  that maximizes  $\Delta(j, s)$  or, equivalently, that minimizes the average impurity

$$\frac{n_{tL}}{n_t} Q_{tL} + \frac{n_{tR}}{n_t} Q_{tR}$$



# Categorical predictors

- When splitting a predictor having  $q$  possible unordered values, there are  $2^{q-1} - 1$  possible partitions of the  $q$  values into two groups.
- All the dichotomies can be explored for small  $q$ , but the computations become prohibitive for large  $q$ .
- In the 2-class case, this computation simplifies. We order the predictor levels according to the proportion falling in class 1. Then we split this predictor as if it were an ordered predictor. One can show this gives the optimal split, in terms of entropy or Gini index, among all possible  $2^{q-1} - 1$  splits.
- The partitioning algorithm tends to favor categorical predictors with many levels  $q$ ; the number of partitions grows exponentially in  $q$ , and the more choices we have, the more likely we can find a good one for the data at hand. This can lead to severe overfitting if  $q$  is large, and such variables should be avoided.



## Example: Heart data

- A retrospective sample of males in a coronary heart disease (CHD) high-risk region of the Western Cape, South Africa.
- There are roughly two controls per positive case of CHD.
- Variables:
  - sbp: systolic blood pressure
  - tobacco: cumulative tobacco (kg)
  - ldl: low density lipoprotein cholesterol
  - adiposity
  - famhist: family history of heart disease (Present, Absent)
  - typea: type-A behavior
  - obesity
  - alcohol: current alcohol consumption
  - age: age at onset
  - chd: response, coronary heart disease



# Tree growing in R

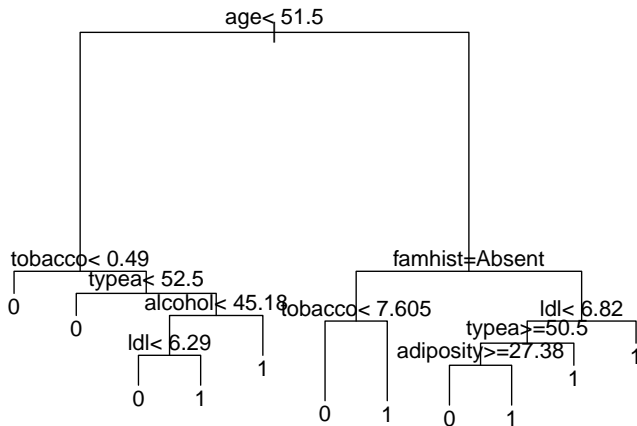
```
heart<-read.table(file = "SAheart.data",sep="," ,header=T,
                  row.names=1)
n<-nrow(heart)

train = sample(n, 2*n/3)
fit <- rpart(chd ~ ., data = heart, method="class",
             subset=train, parms = list(split = 'gini'))
plot(fit,margin = 0.05)
text(fit,pretty=0,cex=0.8)

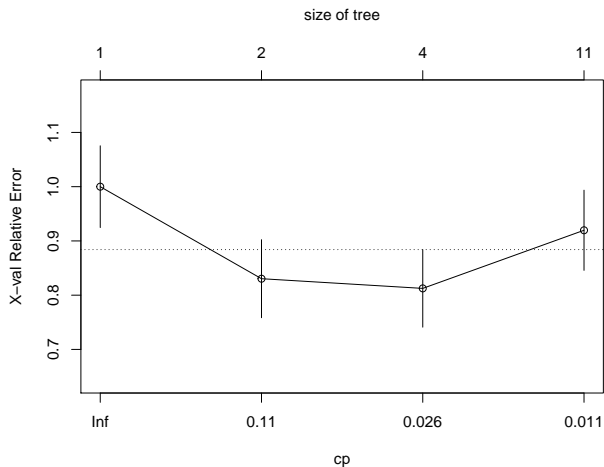
plotcp(fit)
```



## Tree



# Cross-validation error

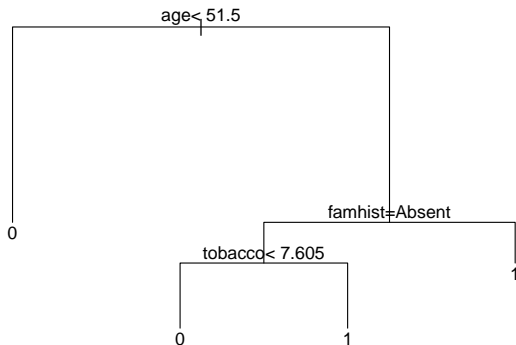


# Pruning

```
pruned_tree<-prune(fit,cp=0.026)
plot(pruned_tree,margin = 0.05)
text(pruned_tree,pretty=0)
```



## Pruned tree





# Test error rate estimation

```
yhat=predict(pruned_tree,newdata=heart[-train,],type='class')
y.test=heart[-train,"chd"]
table(y.test,yhat)
err<-1-mean(y.test==yhat)
```

- Confusion matrix:

	prediction	
true class	0	1
0	95	11
1	31	17

- Test error rate: 0.27



# Advantages and disadvantages of trees

- 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 modern regression and classification approaches.
- However, by **aggregating** many decision trees, the predictive performance of trees can be substantially improved.
- We will see two combination methods, both based on the same resampling technique: the **bootstrap**.



# Overview

- 1 Introductory example
- 2 Learning a regression tree
  - Tree building process
  - Cost-complexity pruning
- 3 Classification trees
- 4 Combining trees
  - Bootstrap
  - Bagging
  - Random Forests



# Overview

- 1 Introductory example
- 2 Learning a regression tree
  - Tree building process
  - Cost-complexity pruning
- 3 Classification trees
- 4 Combining trees
  - Bootstrap
  - Bagging
  - Random Forests



# The bootstrap

- The bootstrap is a flexible and powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method.
- For example, it can provide an estimate of the **standard error** of an estimator, or a **confidence interval** for a parameter.



# A simple problem

- Let  $X \sim F_\theta$  be a random variable whose distribution depends on a parameter  $\theta$ .
- Assume that we have an iid sample  $X_1, \dots, X_n$  and an estimator  $\hat{\theta}(X_1, \dots, X_n)$  of  $\theta$ .
- How to estimate the standard error  $\text{se}(\hat{\theta})$  of  $\hat{\theta}$ ?
- In simple cases a closed-form expression of  $\text{se}(\hat{\theta})$  can be derived, but most of the times it cannot.



## Ideal solution

If we could draw a large number  $N$  of datasets from the same distribution, we could compute the corresponding realizations of  $\hat{\theta}$ :

$$\begin{aligned}x_1^{(1)}, \dots, x_n^{(1)} &\longrightarrow \hat{\theta}^{(1)} \\ &\vdots \\ x_1^{(N)}, \dots, x_n^{(N)} &\longrightarrow \hat{\theta}^{(N)}\end{aligned}$$

and estimate  $\text{se}(\hat{\theta})$  by the empirical standard deviation:

$$\widehat{\text{se}}(\hat{\theta}) = \sqrt{\frac{1}{N} \sum_{r=1}^N (\hat{\theta}^{(r)} - \bar{\hat{\theta}})^2} \xrightarrow{P} \text{se}(\hat{\theta})$$



## Real life solution

- In real life, we only have one realization  $x_1, \dots, x_n$  of the sample and we do not know the true distribution.
- The bootstrap idea is to **replace the true distribution by the empirical distribution**, with a mass  $1/n$  on each observation  $x_i$ .
- To draw a bootstrap dataset  $x_1^*, \dots, x_n^*$ , we randomly draw  $n$  observations from  $x_1, \dots, x_n$  with replacement.
- For instance, if  $n = 5$ , the sample is  $(0.5, 1.2, -0.3, 0.8, -1.4)$  and  $\hat{\theta}$  is the mean, we have the following  $B = 3$  bootstrap samples and corresponding values of  $\hat{\theta}^*$ :

$$(-0.30, 0.50, 0.50, 0.80, -0.30) \longrightarrow 0.24$$

$$(0.8, -0.3, 0.5, 0.8, -0.3) \longrightarrow 0.3$$

$$(1.20, 1.20, 0.80, 0.80, 1.20) \longrightarrow 1.04$$





# Bootstrap estimation of the standard error

- To estimate the standard error using the bootstrap, we generate  $B$  bootstrap datasets  $x_1^{*b}, \dots, x_n^{*b}$ ,  $b = 1, \dots, B$  and we compute the corresponding estimates  $\hat{\theta}^{*b} = \hat{\theta}(x_1^{*b}, \dots, x_n^{*b})$ ,  $b = 1, \dots, B$ .
- The bootstrap estimate of standard error is then

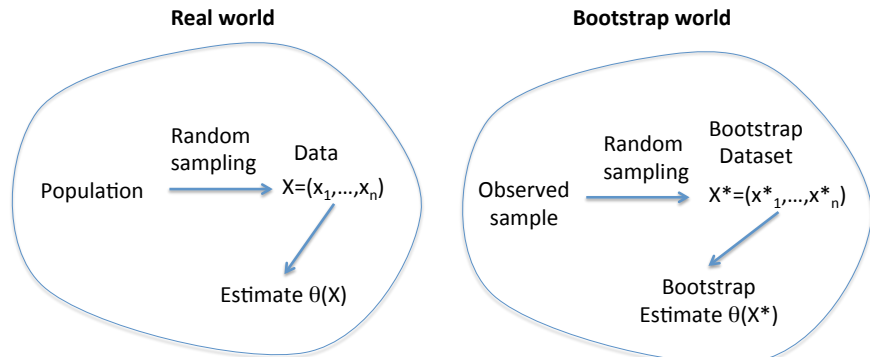
$$\widehat{\text{se}}_B(\hat{\theta}) = \sqrt{\frac{1}{B-1} \sum_{b=1}^B (\hat{\theta}^{*(b)} - \overline{\hat{\theta}^*})^2.}$$

with

$$\overline{\hat{\theta}^*} = \frac{1}{B} \sum_{b=1}^B \hat{\theta}^{*(b)}$$



# General picture of the bootstrap



# Overview

- 1 Introductory example
- 2 Learning a regression tree
  - Tree building process
  - Cost-complexity pruning
- 3 Classification trees
- 4 Combining trees
  - Bootstrap
  - **Bagging**
  - Random Forests



# 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  $X_1, \dots, X_n$ , each with variance  $\sigma^2$ , the variance of the mean  $\bar{X}$  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 – continued

- 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** and we train  $B$  decision trees, one from each bootstrap dataset.
- For regression problems, we then average all the predictions to obtain

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

where  $\hat{f}^{*b}(x)$  is the prediction at  $x$  for the  $b$ -th tree.



# Bagging classification trees

- The above prescription applied to regression trees.
- For classification trees: for each test observation, we record the class predicted by each of the  $B$  trees, and take a **majority vote**: the overall prediction is the majority class among the  $B$  predictions.
- If we are interested in the posterior class probabilities, we can rather **average the class proportions** in the terminal nodes.

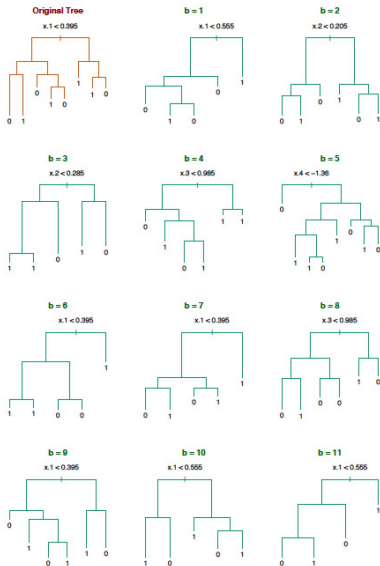


## Example

- We generated a sample of size  $n = 30$ , with two classes and  $p = 5$  predictors, each having a standard Gaussian distribution with pairwise correlation 0.95.
- The response  $Y$  was generated according to  $\mathbb{P}(Y = 1 \mid x_1 \leq 0.5) = 0.2$ ,  $\mathbb{P}(Y = 1 \mid x_1 > 0.5) = 0.8$ . The Bayes error is 0.2.
- A test sample of size 2000 was also generated from the same population.
- We fit classification trees to the training sample and to each of  $B = 200$  bootstrap samples. No pruning was used.

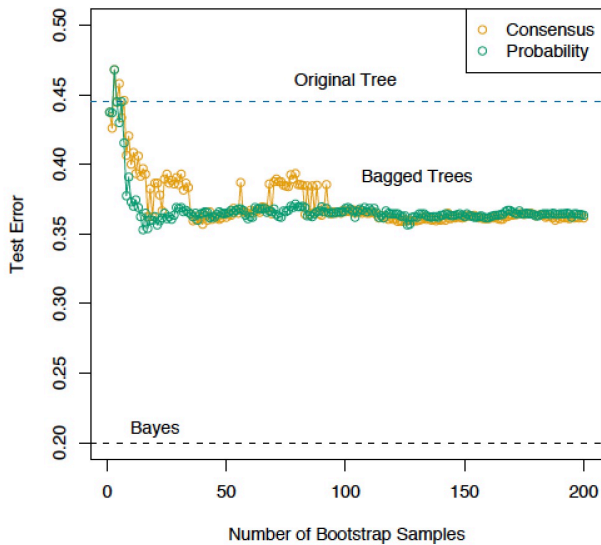


# Bagged decision trees





## Error curves



# Out-of-Bag Error Estimation

- It is a 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 all trees in which that observation was OOB. This will yield around  $B/3$  predictions for the  $i$ -th observation, which we average.



# Overview

- 1 Introductory example
- 2 Learning a regression tree
  - Tree building process
  - Cost-complexity pruning
- 3 Classification trees
- 4 Combining trees
  - Bootstrap
  - Bagging
  - 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.
- A fresh selection of  $m$  predictors is taken at each split, and typically we choose  $m \approx \sqrt{p}$ .
- When  $m = p$ , random forests boil down to bagging.

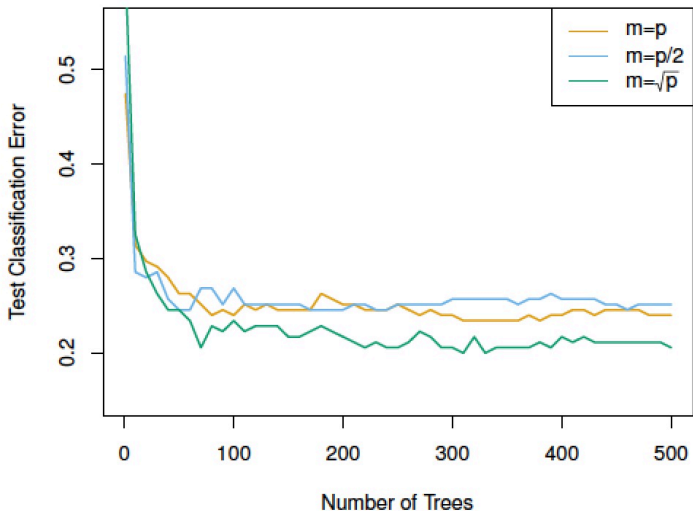


## Example: gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 500 genes measured on tissue samples from 349 patients.
- Each of the patient samples belongs to one of  $c = 15$  classes: either normal or one of 14 different types of cancer.
- We randomly divided the observations into a training set 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



# Variable importance

- Bagging improves prediction accuracy at the expense of interpretability.
- Although the collection of bagged trees is much more difficult to interpret than a single tree, one can obtain an overall summary of the **importance** of each predictor.
- There are, at least, two ways to measure the importance of a variable.



# Variable importance measures

**Mean decrease in accuracy:** For each tree, the prediction error on the OOB portion of the data is recorded (error rate for classification, MSE for regression). Then the same is done after randomly permuting (i.e., randomly shuffling the values of) each predictor variable. The differences between the two are then averaged over all trees.

**Mean decrease in node impurity:** total decrease in node impurities from splitting on the variable, averaged over all trees. For classification, the node impurity is measured by the Gini index. For regression, it is measured by RSS.





# Bagging in R

```
library(randomForest)
p<-ncol(heart)-1
bag.heart=randomForest(as.factor(chd) ~ .,data=heart,subset=train,
                       mtry=p)
yhat1=predict(bag.heart,newdata=heart[-train,],type="response")
table(y.test,yhat1)
1-mean(y.test==yhat1)
```

- Confusion matrix:

	prediction	
true class	1	2
1	81	13
2	38	22

- Test error rate: 0.33



# Random forests in R

```
library(randomForest)
RF.heart=randomForest(as.factor(chd) ~ .,data=heart,subset=train,
                      mtry=3,importance=TRUE)
yhat2=predict(RF.heart,newdata=heart[-train,],type="response")
table(y.test,yhat2)
1-mean(y.test==yhat2)
varImpPlot(RF.heart)
```

- Confusion matrix:

	prediction	
true class	1	2
1	84	10
2	36	24

- Test error rate: 0.30



# Variable importance plot

RF.heart

