Advanced Computational Econometrics: Machine Learning

Chapter 5: Tree-based and ensemble methods

Thierry Denœux

July 2019

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 decision-tree 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 interpretation.

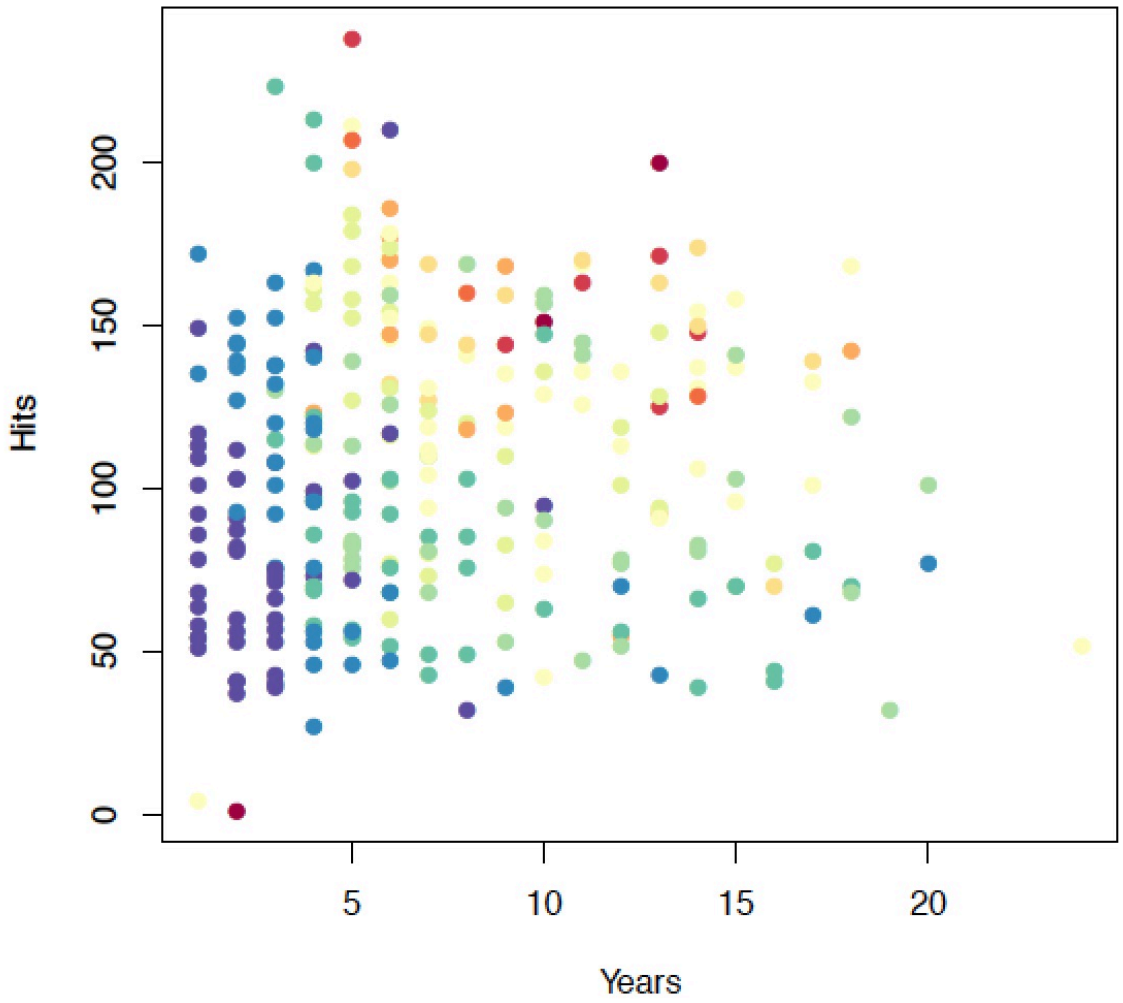
Regression/Classification Trees

- 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.

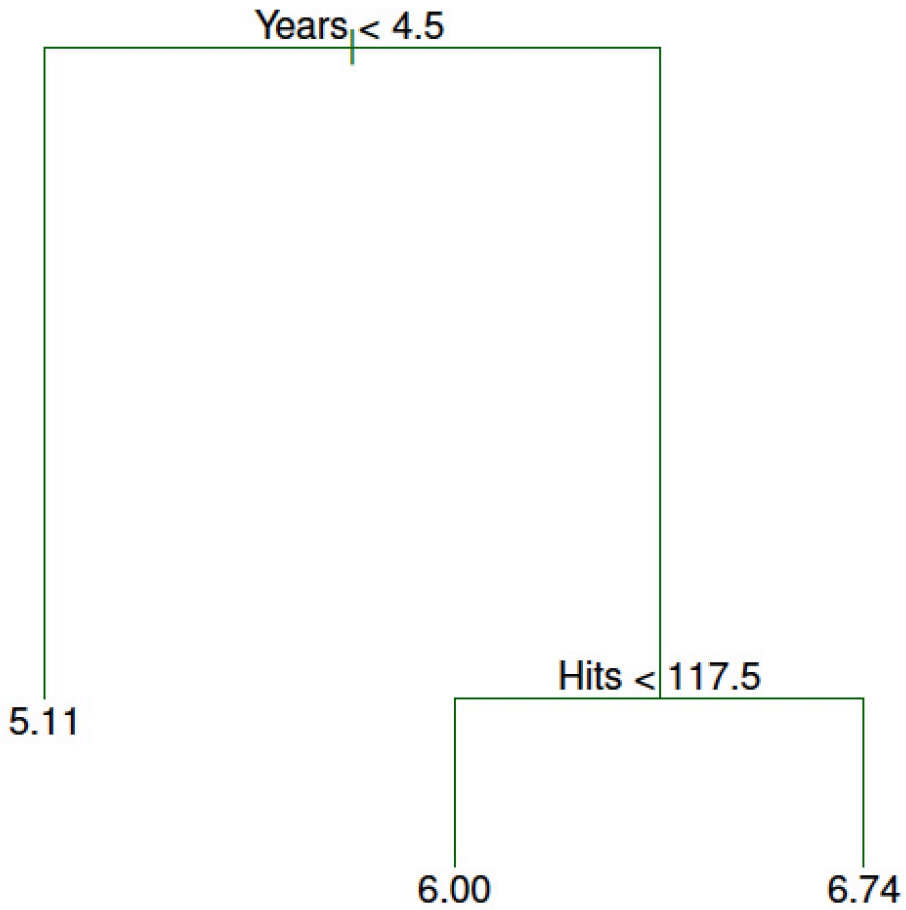
Overview

Baseball salary data: how would you stratify it?

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



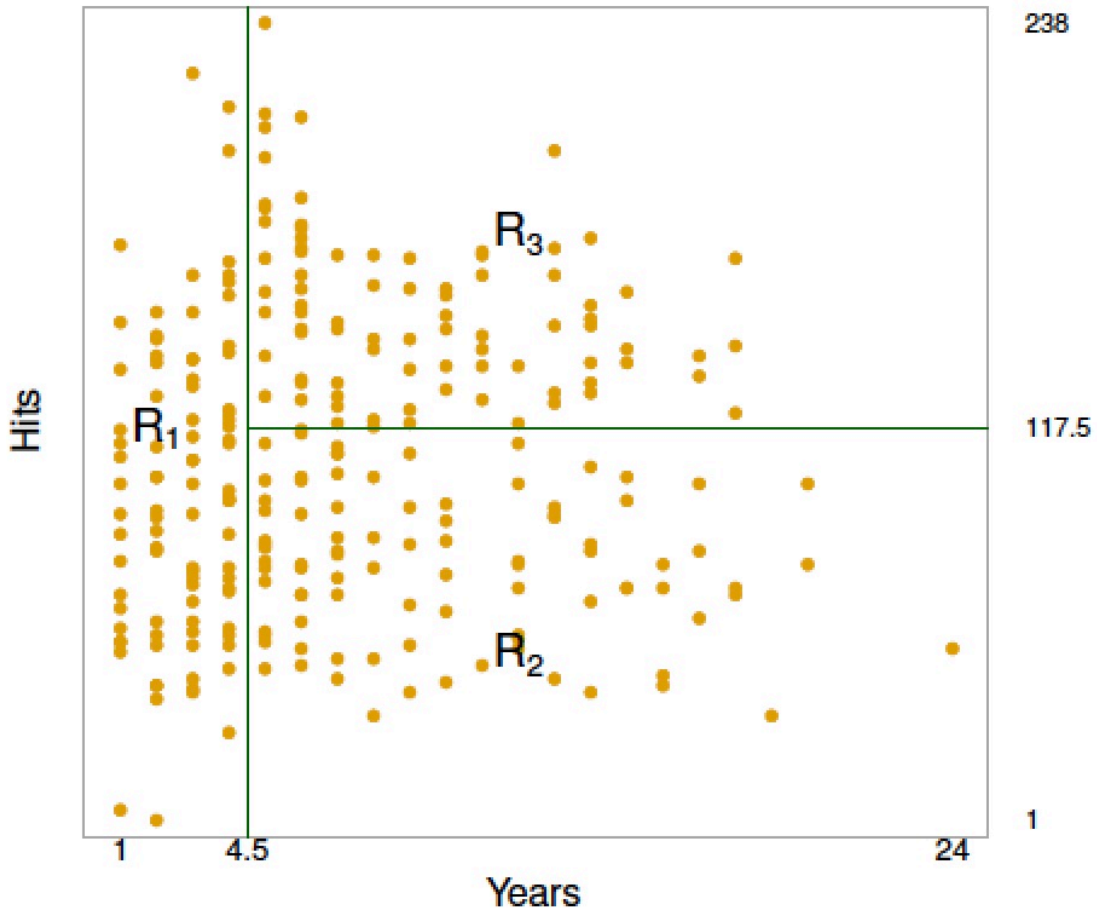
Regression tree for these data



- At a given internal node, the label (of the form $X_j < t_k$) 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.

Results

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



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 $Hits < 117.5$.

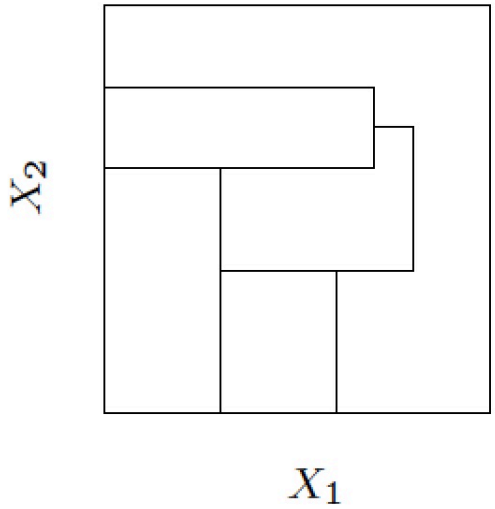
Interpretation of Results

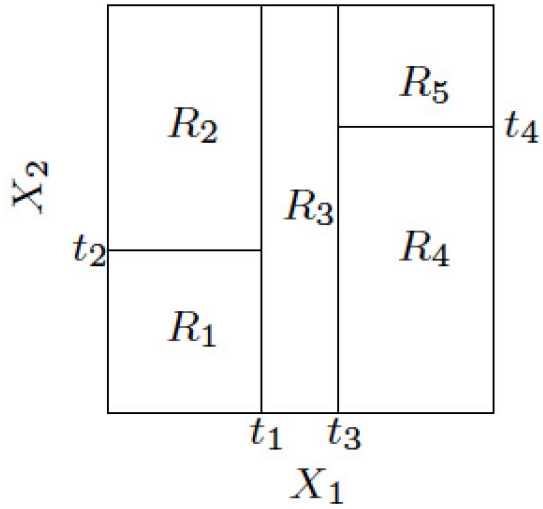
- Years is the most important factor in determining Salary, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in his Salary.
- But among players 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.

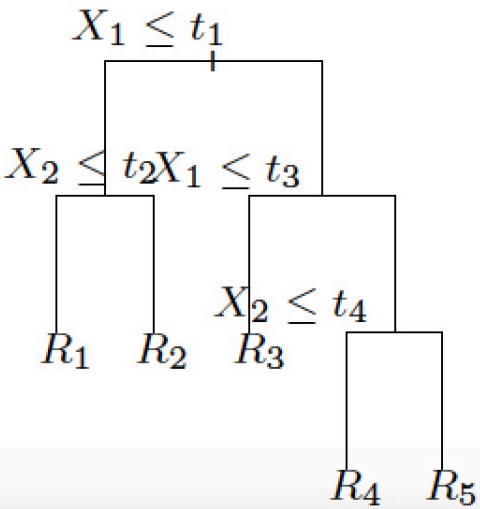
Predictions

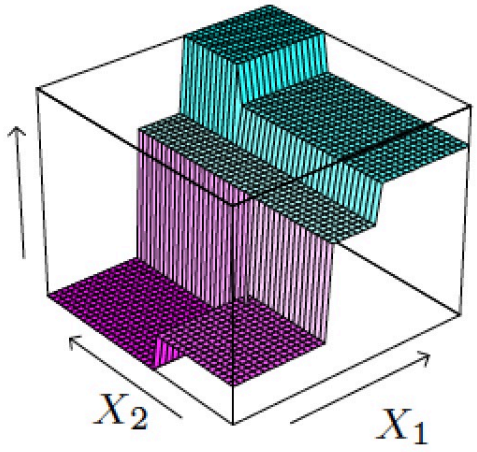
- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in the next slide.

Example


X_2
X_1


R_2
R_5
t_4
X_2
R_3
t_2
R_4
R_1
t_1
t_3
X_1


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


X_2
X_1

Details of previous figure

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

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

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

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

Overview

Overview

Growing a regression tree

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

Growing a regression tree

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

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

- If we adopt as our criterion minimization of the RSS error $\sum_{i=1}^{n}(y_i - f(x_i))^2$, it is easy to see that the best c_m is just the average of y_i in region R_m:

$$c_m = \text{ave}\left\{y_i \mid x_i \in R_m\right\}.$$

Growing a regression tree

- 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.

Algorithm

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

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

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

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

with

$$\widehat{c}_1(j,s) = \text{ave}\{y_i \mid x_i \in R_1(j,s)\} \text{ and } \widehat{c}_2(j,s) = \text{ave}\{y_i \mid x_i \in R_2(j,s)\}$$

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

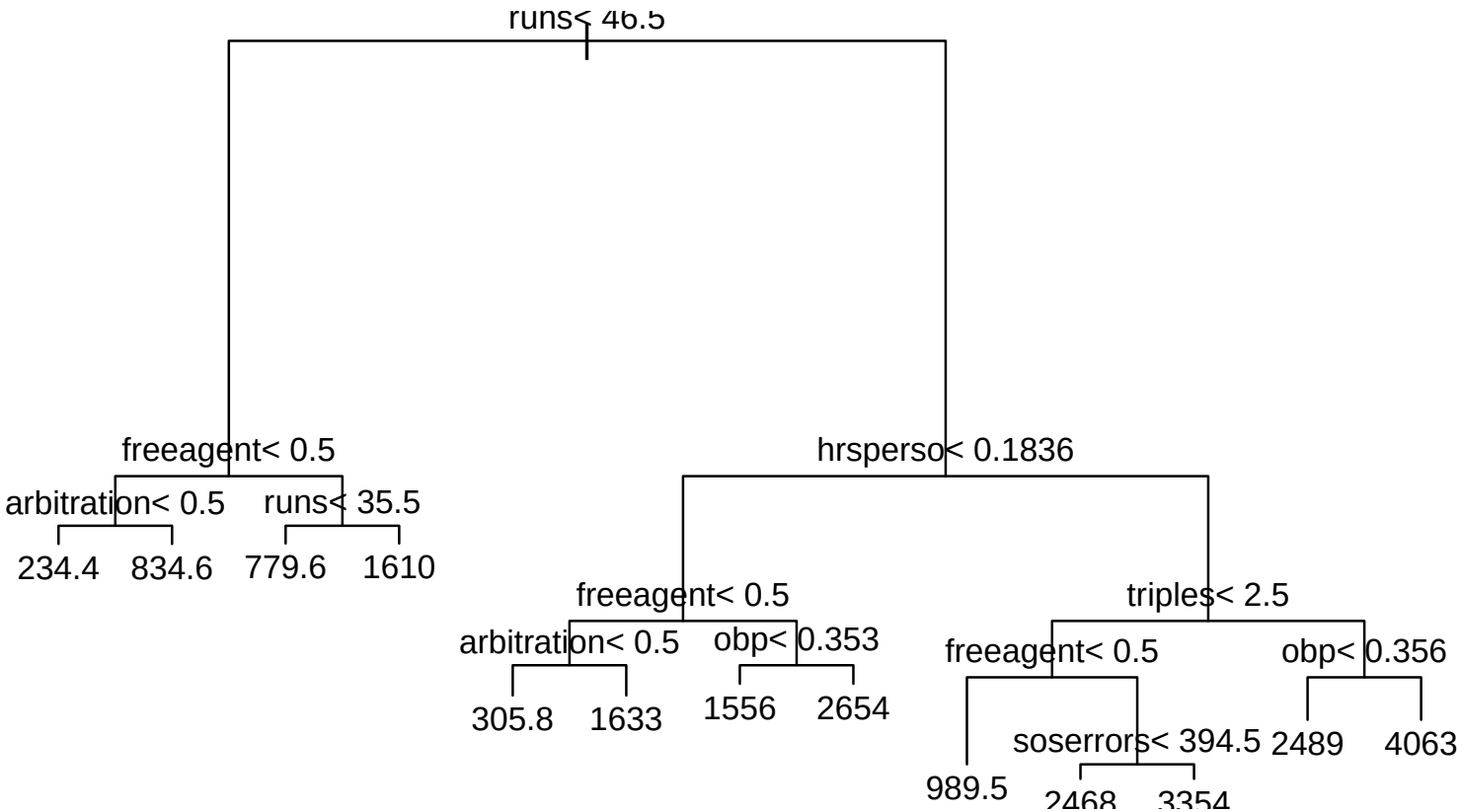
Algorithm

- 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 might overfit the data, while a small tree might not capture the important structure.

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

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.

Cost-complexity pruning

- 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 (non-terminal) nodes.
- We index terminal nodes by t, with node t representing region R_t.
- Let $\widetilde{T}$ denote the set of terminal nodes in T. Letting

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

$$Q_t = \frac{1}{n_t} \sum_{x_i \in R_t} (y_i - \widehat{c}_t)^2, \quad C(T) = \sum_{t \in \widetilde{T}} n_t Q_t$$

- We define the cost-complexity criterion

$$C_\alpha(T) = C(T) + \alpha |\widetilde{T}|.$$

Cost-complexity pruning

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

Weakest link pruning

- We start from the full tree T_0.
- For any internal node t, let T_t be the branch of T with root t.
- If we prune T_t, the cost-complexity criterion becomes smaller if

$$C(t) + \alpha < C(T_t) + \alpha|\widetilde{T}_t| \Leftrightarrow \alpha > \frac{C(t) - C(T_t)}{|\widetilde{T}_t| - 1} = g_0(t)$$

- The weakest link t_0 in T_0 is the node such that $g_0(t_0) = \min_t g_0(t)$. Let $\alpha_1 = g_0(t_0)$.
- Meaning: if we increase α 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.

Weakest link pruning

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

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

  and an increasing sequence of α values, $0 = \alpha_0 < \alpha_1 < \alpha_2 < \ldots$.
- We can show that, for all $k \geq 0$ and all $\alpha_k \leq \alpha < \alpha_{k+1}$, the optimum tree $T(\alpha)$ is equal to T_k.

Choosing α

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

Cross-validation in detail

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

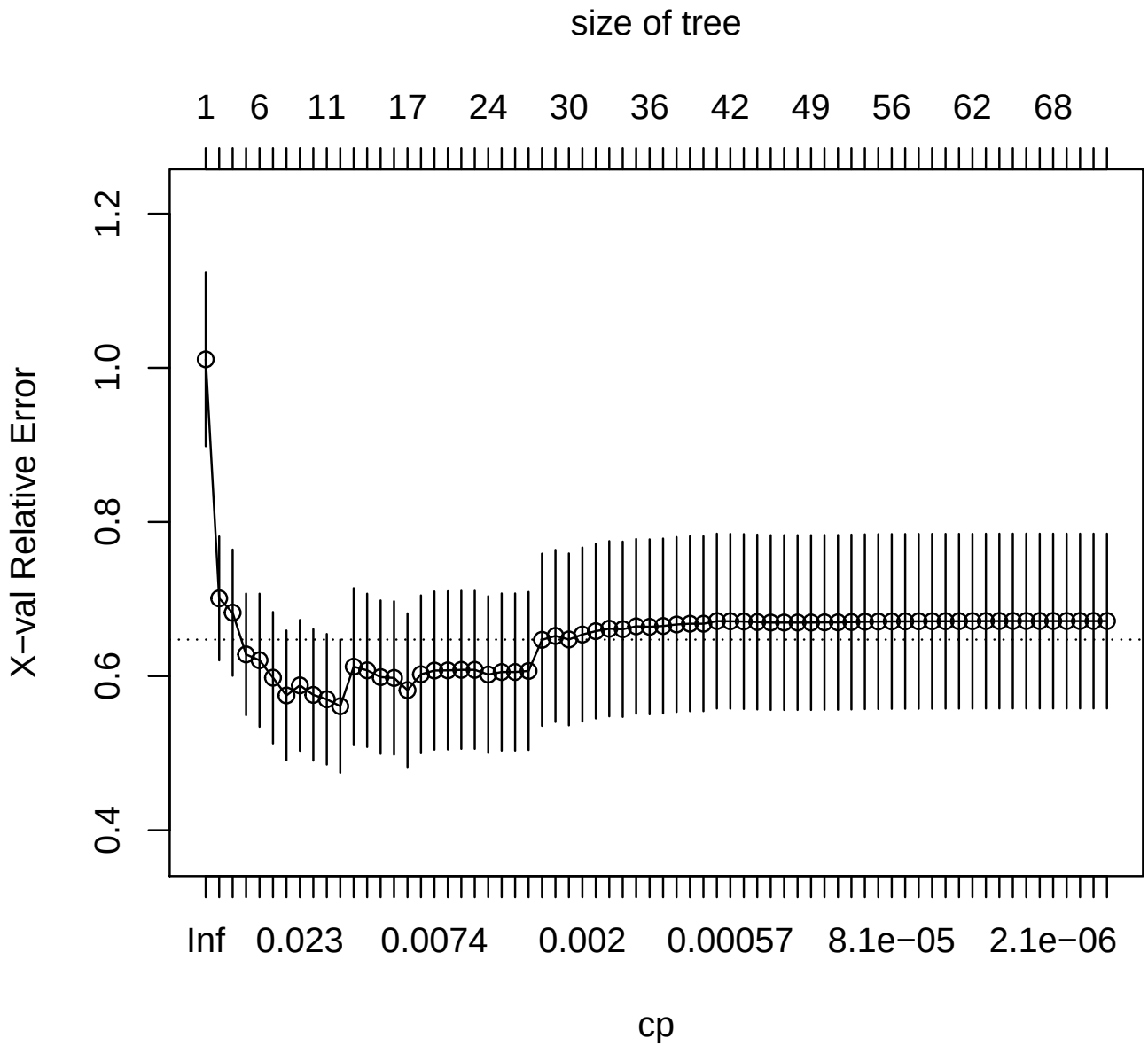
Pruning a regression tree in R

```
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

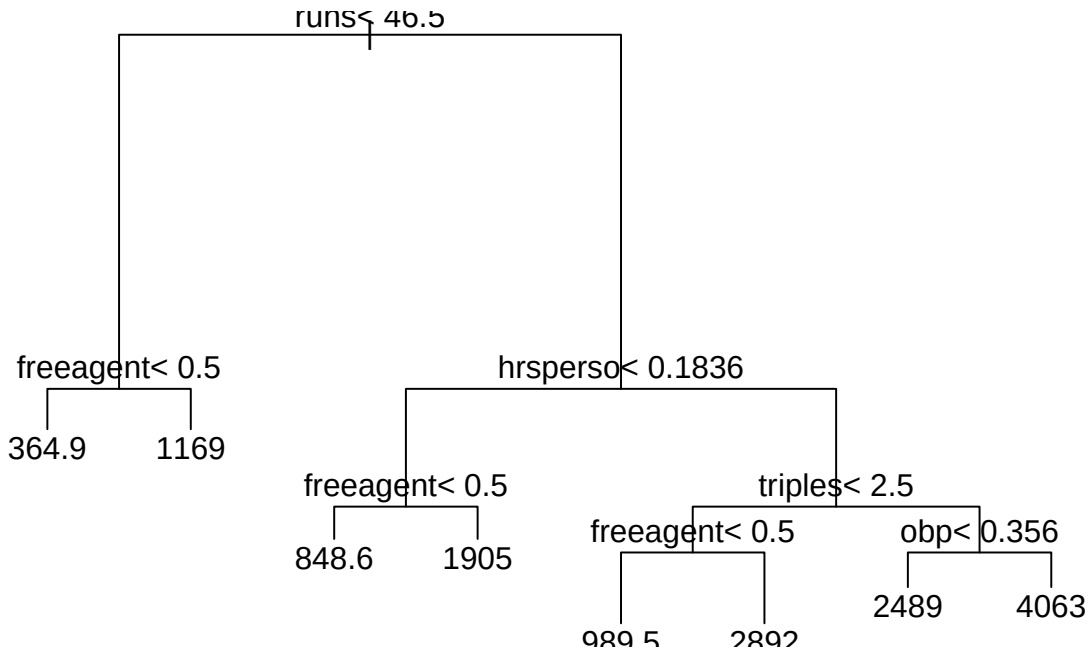
```
> printcp(fit)

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] soserrors    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
```

Pruning a regression tree in R



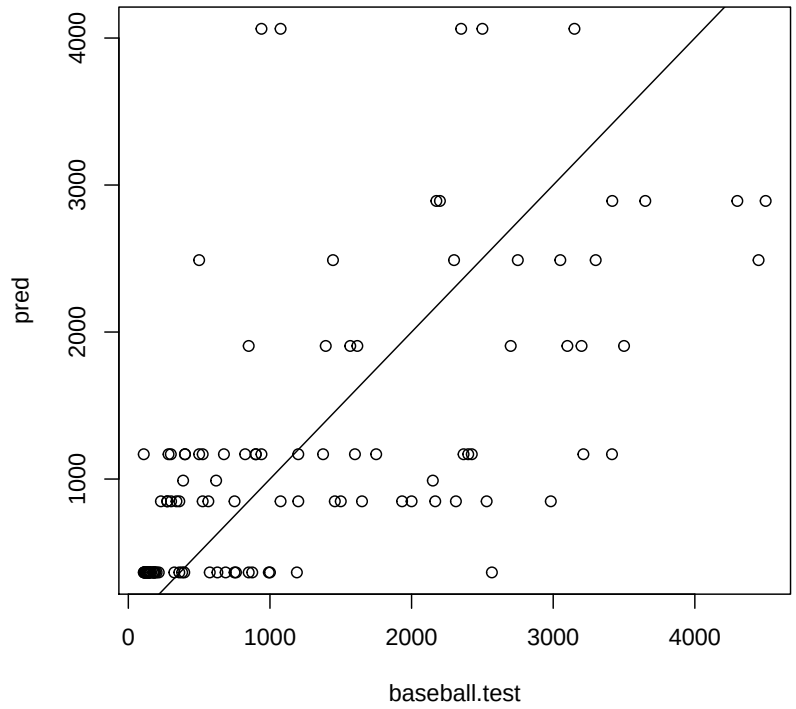
Pruning a regression tree in R

```
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

Classification trees

- If the target is a classification outcome taking values $1, 2, \ldots, c$, the only changes needed in the tree-growing algorithm pertain to the criteria for splitting nodes and pruning the tree.
- For regression we used the MSE node impurity measure Q_t,

$$Q_t = \frac{1}{n_t} \sum_{x_i \in R_t} (y_i - \widehat{c}_t)^2,$$

but this is not suitable for classification.

Impurity measures

- In a node t, representing a region R_t with n_t observations, let

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

  be the proportion of class k observations in node t.
- We classify the observations in node t to class $k(t) = \arg\max_k \widehat{p}_{tk}$, the majority class in node t.
- Different measures Q_t of node impurity include the following:

  Misclassification error: $\frac{1}{n_t} \sum_{x_i \in R_t} I(y_i \neq k(t)) = 1 - \widehat{p}_{tk(t)}$

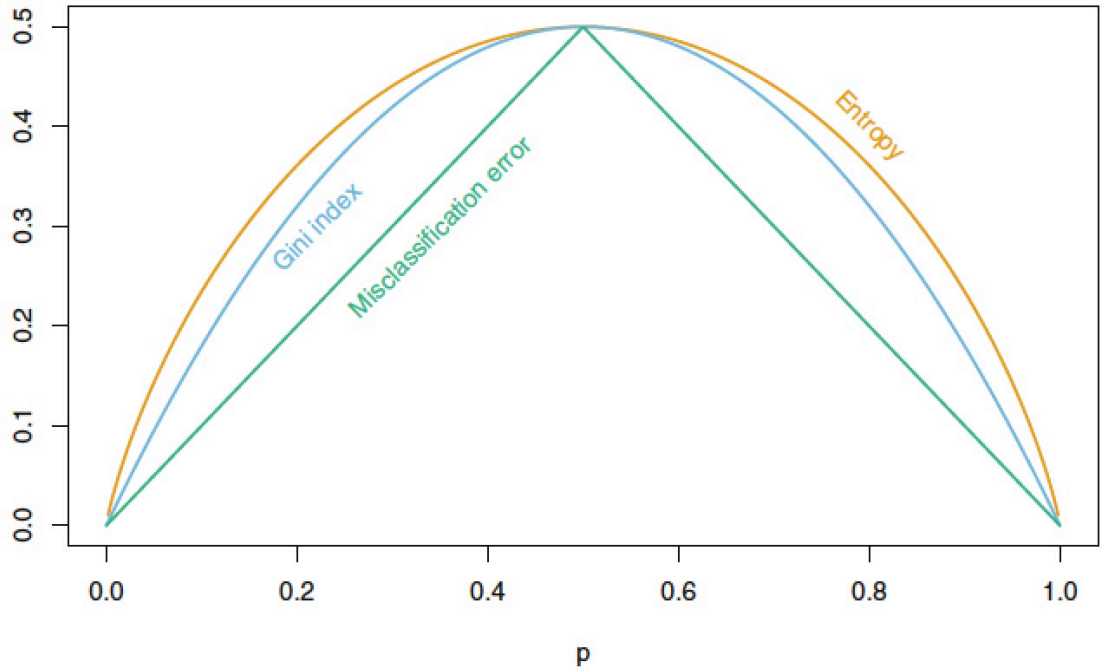
  Gini index: $\sum_{k \neq k'} \widehat{p}_{tk} \widehat{p}_{tk'} = \sum_{k=1}^{c} \widehat{p}_{tk}(1 - \widehat{p}_{tk})$

  Entropy: $-\sum_{k=1}^{c} \widehat{p}_{tk} \log \widehat{p}_{tk}$

Comparison between impurity measures

Case $c = 2$

- For two classes, if p is the proportion in the second class, these three measures are $1 - \max(p, 1-p)$, $2p(1-p)$ and $-p\log p - (1-p)\log(1-p)$, respectively.
- All three are similar, but entropy and the Gini index are differentiable, and hence more amenable to numerical optimization.



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}
- The average decrease of impurity is

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

- 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

$$\frac{n_{t_L}}{n_t} Q_{t_L} + \frac{n_{t_R}}{n_t} Q_{t_R}$$

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 outcome 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.

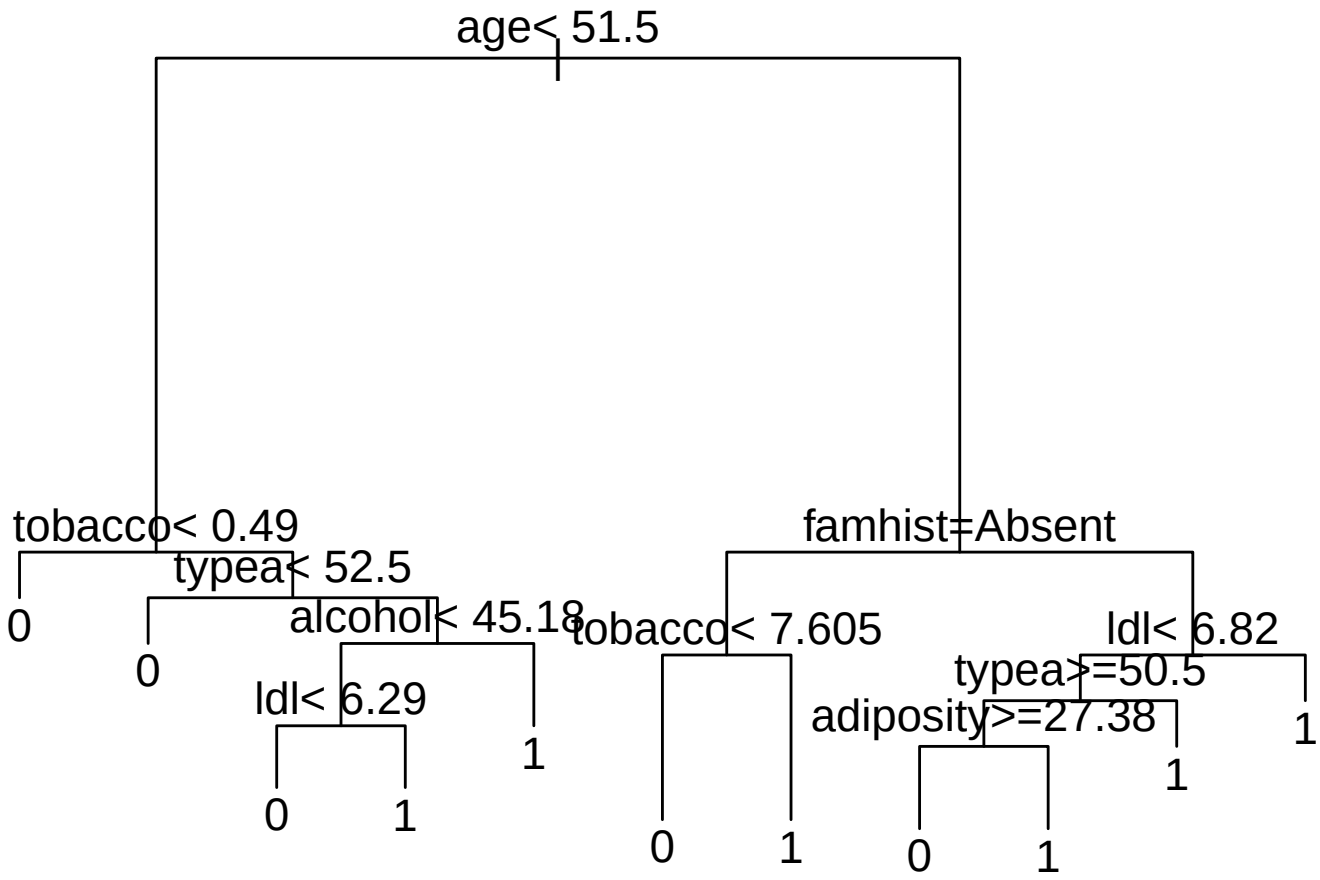
Heart data

- A retrospective sample of males in a heart-disease high-risk region of the Western Cape, South Africa.
- There are roughly two controls per 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

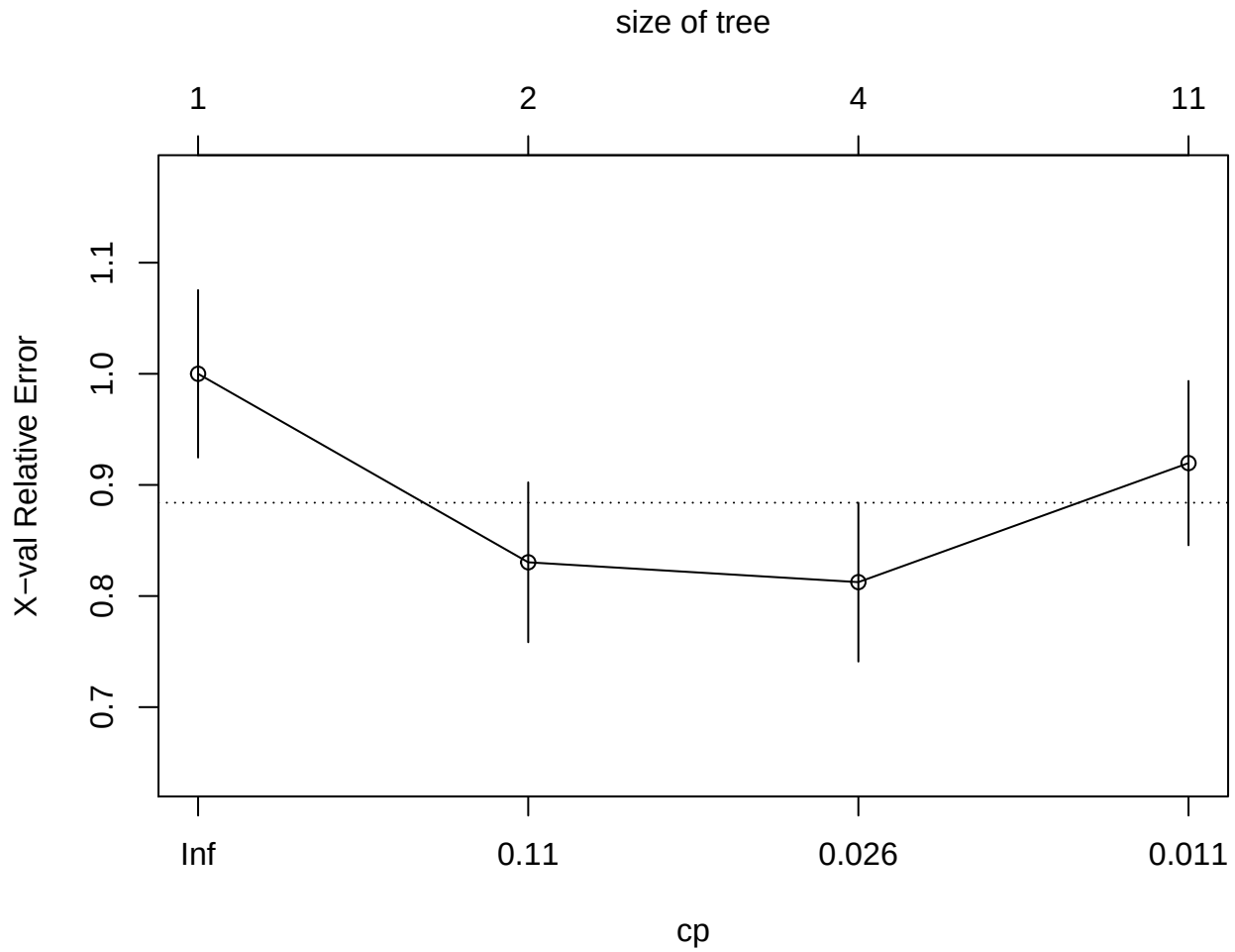
Tree growing in R

```
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


age< 51.5
tobacco< 0.49
famhist=Absent
typea< 52.5
0
alcohol< 45.18
tobacco< 7.605
ldl< 6.82
0
ldl< 6.29
typea>=50.5
1
adiposity>=27.38
1
1
0
1
0
1
0
1

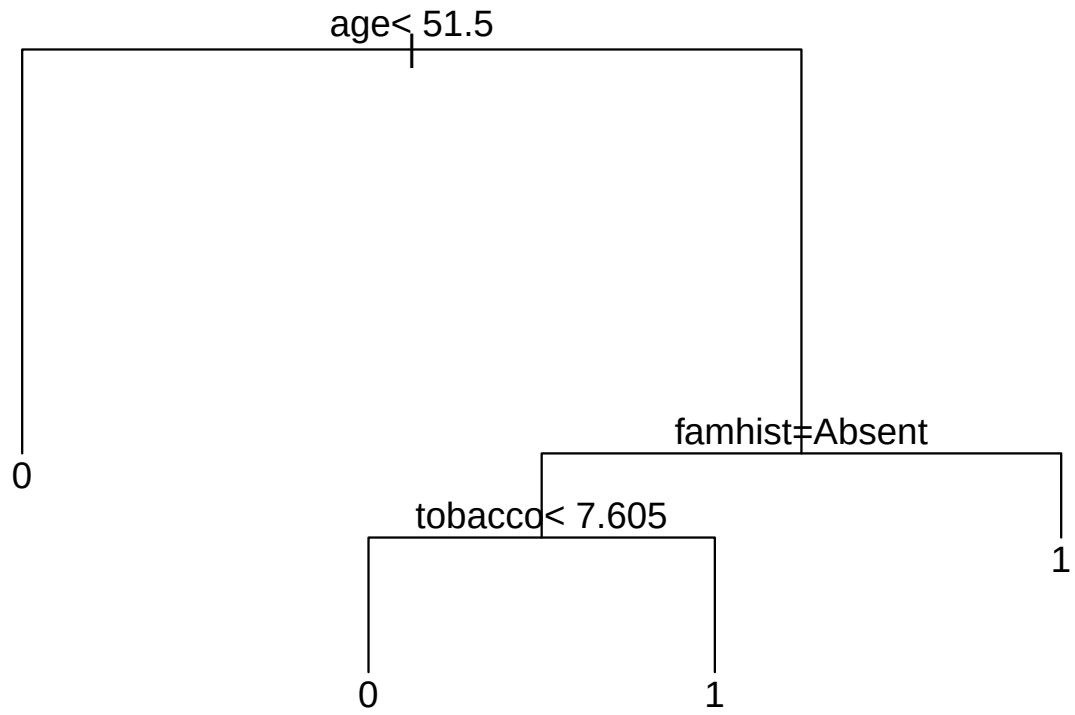
Cross-validation error



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, which we introduce first.

Overview

Overview

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 $X_1, \ldots, X_n$, each with variance σ^2, the variance of the mean $\overline{X}$ of the observations is given by σ^2/n.
- In other words, averaging a set of observations reduces variance. Of course, this is not practical because we generally do not have access to multiple training sets.

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. We then train our method on the b-th bootstrapped training set in order to get $\widehat{f}^{*b}(x)$, the prediction at a point x. We then average all the predictions to obtain

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

This is called bagging.

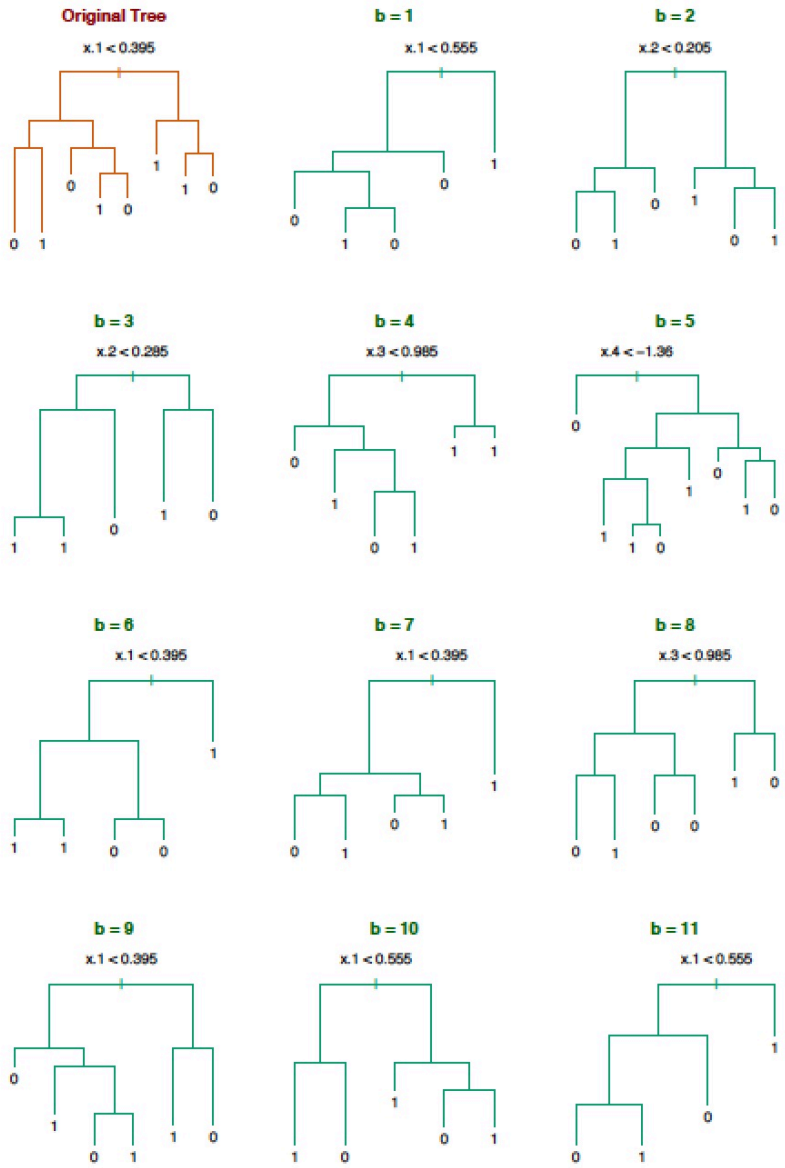
Bagging classification trees

- 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 most commonly occurring class among the B predictions.
- If we are interested in the posterior probabilities, we can rather average the class proportions in the terminal nodes.

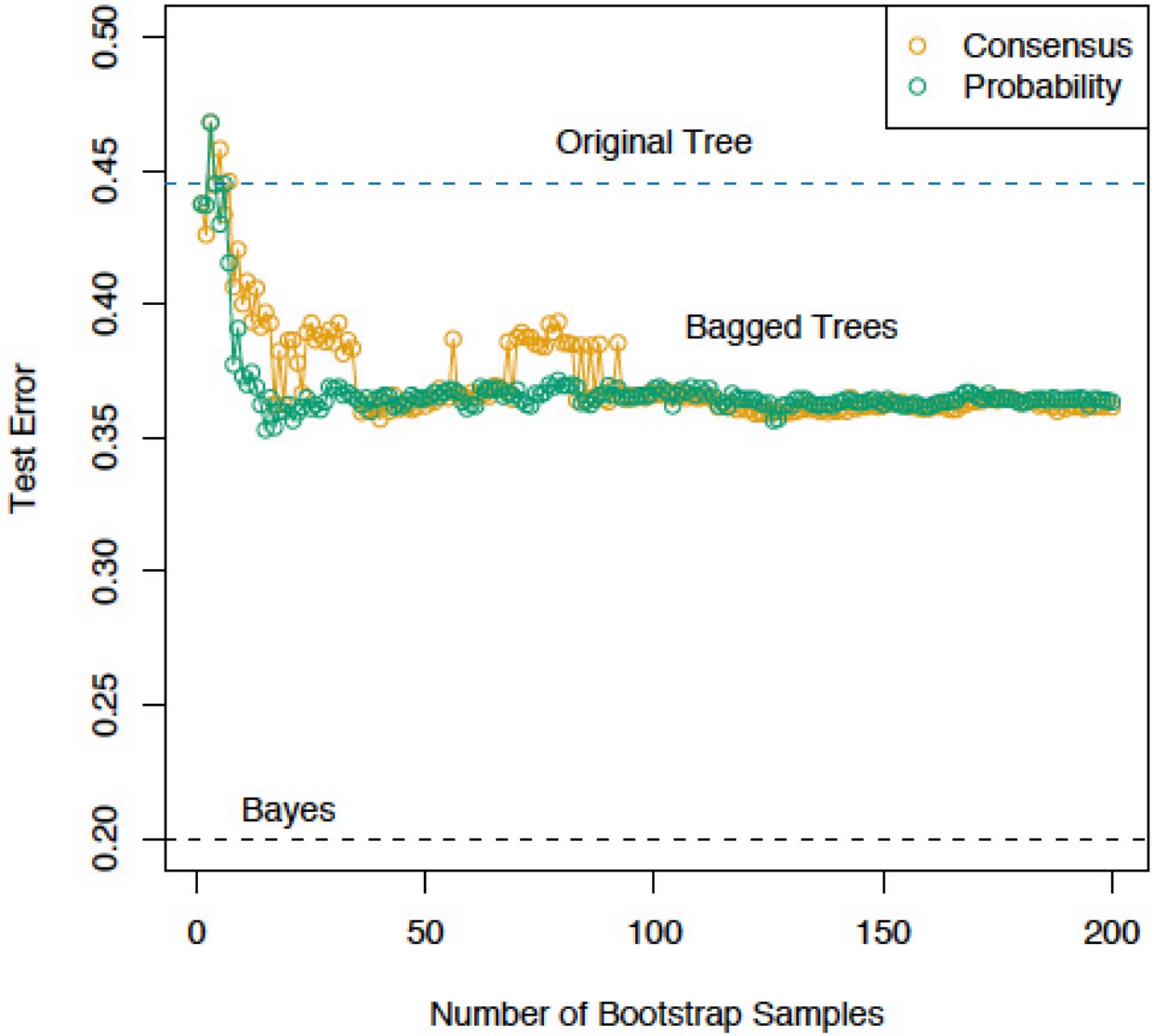
Example

- We generated a sample of size $n = 30$, with two classes and $p = 5$ features, each having a standard Gaussian distribution with pairwise correlation 0.95.
- The response Y was generated according to $Pr(Y = 1|x_1 \leq 0.5) = 0.2$, $Pr(Y = 1|x_1 > 0.5) = 0.8$. 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 200 bootstrap samples. No pruning was used.

Bagged decision trees



Error curves



Out-of-Bag Error Estimation

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

Overview

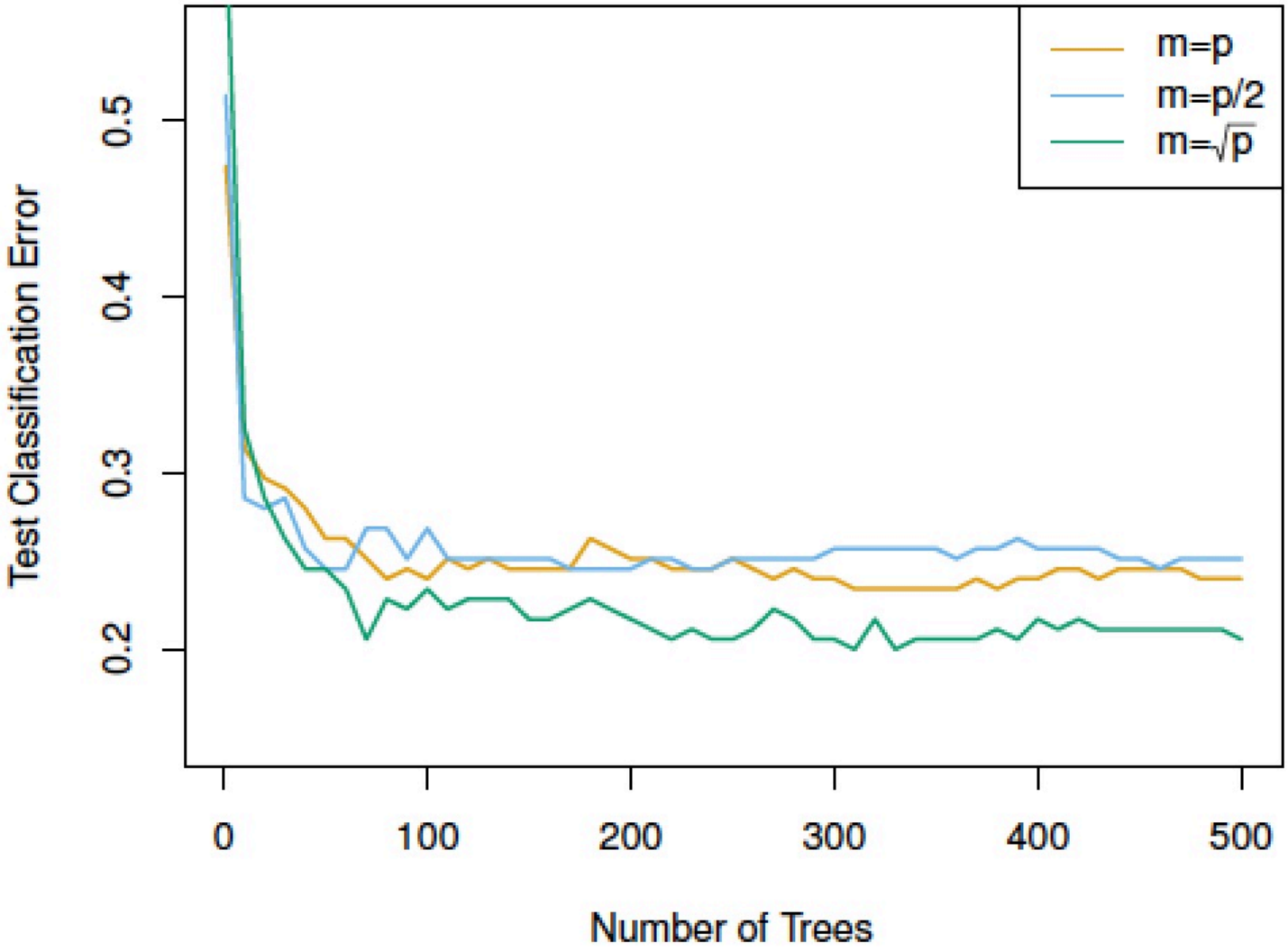
Random Forests

- 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 $m \approx \sqrt{p}$ – that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

Example: gene expression data

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

Results: gene expression data



Details of previous figure

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

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	83	23
2	28	20

- Test error rate: 0.33

Random forests in R

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

- Confusion matrix:

	prediction	
true class	1	2
1	83	17
2	30	18

- Test error rate: 0.31