

Computational statistics

Lecture 6: Tree-based methods

Thierry Denœux

7 March, 2016



Tree-based methods

- Here we describe **tree-based** methods 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.



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.

The Basics of Decision Trees

- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.



Overview

Introductory example

Learning a regression tree

Tree building process

Pruning

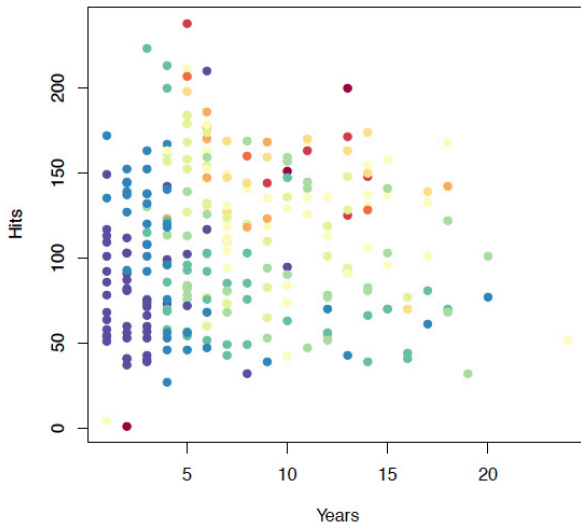
Classification trees

Combining trees

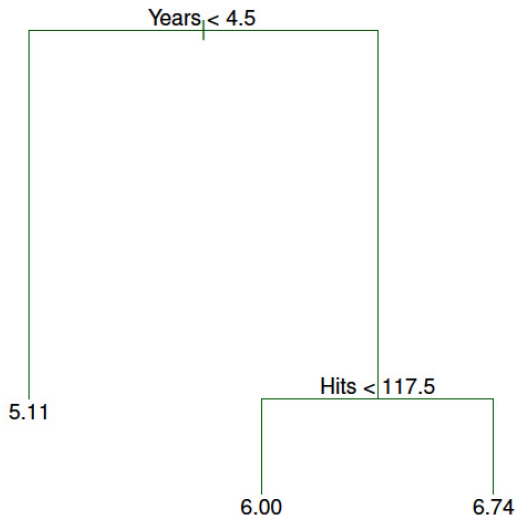


Baseball salary data: how would you stratify it?

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



Decision tree for these data



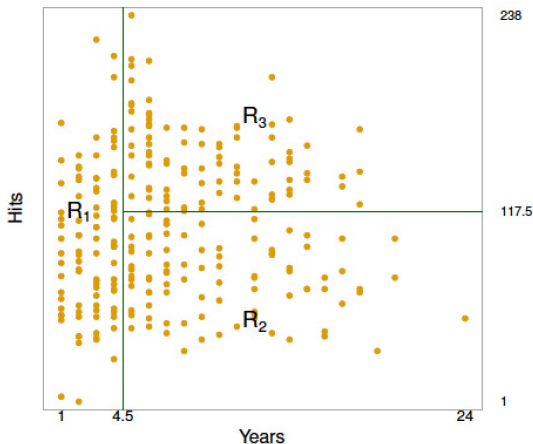
Details of previous figure

- 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* \geq 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: $R_1 = \{X | \text{Years} < 4.5\}$, $R_2 = \{X | \text{Years} \geq 4.5, \text{Hits} < 117.5\}$, and $R_3 = \{X | \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$.



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

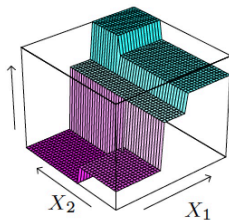
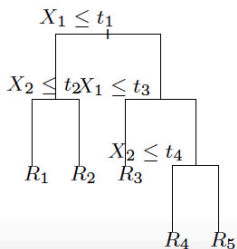
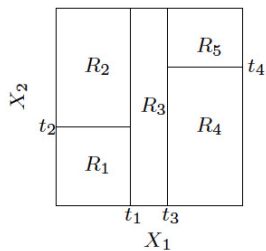
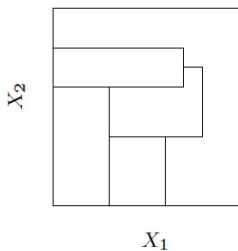


Predictions

- 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



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

Introductory example

Learning a regression tree

Tree building process

Pruning

Classification trees

Combining trees



Overview

Introductory example

Learning a regression tree

Tree building process

Pruning

Classification trees

Combining trees



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 $i = 1, 2, \dots, N$, with $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})$.
- 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, \dots, R_M , and we model the response as a constant c_m in each region:

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

- If we adopt as our criterion minimization of the sum of squares $\sum (y_i - f(x_i))^2$, it is easy to see that the best c_m is just the average of y_i in region R_m :

$$c_m = \text{ave}(y_i | x_i \in R_m).$$



Algorithm

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

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

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

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

- For any choice j and s , the inner minimization is solved by

$$c_1 = \text{ave}(y_i | x_i \in R_1(j, s)) \text{ and } c_2 = \text{ave}(y_i | x_i \in R_2(j, s))$$

- For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the 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(tree)
baseball <- read.table("baseball.dat",header=TRUE)
N<-nrow(baseball)

train = sample(1:N, N/2)
tree.baseball=tree(salary~.,baseball,subset=train)
```



Regression trees in R

```
summary(tree.baseball)
```

Regression tree:

```
tree(formula = salary ~ ., data = baseball, subset = train)
```

Variables actually used in tree construction:

"runs" "freeagent" "arbitration" "homeruns" "rbis"

"hrsperror" "sbsobp" "hrsperso" "obppererror"

Number of terminal nodes: 11

Residual mean deviance: 305800 = 48010000 / 157

Distribution of residuals:

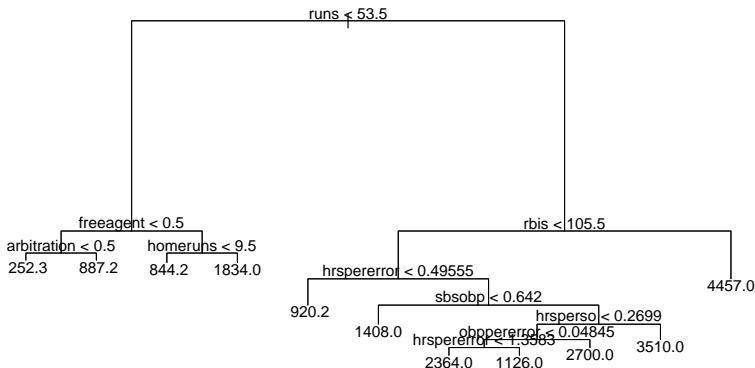
Min. 1st Qu. Median Mean 3rd Qu. Max.

-1725.00 -146.80 -69.78 0.00 219.70 1766.00



Regression trees in R

```
plot(tree.baseball)
text(tree.baseball,pretty=0)
```



Overview

Introductory example

Learning a regression tree

Tree building process

Pruning

Classification trees

Combining trees



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 \tilde{T} denote the set of terminal nodes in T . Letting

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

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

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

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



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_α , 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

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

- 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 tree

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

and an increasing sequence of α values, $0 = \alpha_0 < \alpha_1 < \alpha_2 < \dots$

- 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 .



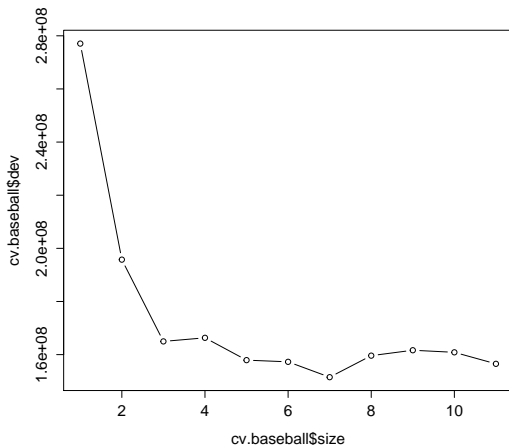
Choosing α

- Estimation of α is achieved by **five- or tenfold cross-validation**: we choose the value $\hat{\alpha}$ to minimize the cross-validated sum of squares. Our final tree is $T(\hat{\alpha})$.
- More precisely,
 - Using the whole training set, we get a sequence of trees, $T_0 \succ T_1 \succ \dots \succ t_{root}$, where T_k is the best tree for $\alpha_k \leq \alpha < \alpha_{k+1}$.
 - For $k = 0, 1, 2, \dots$, set $\alpha'_k = \sqrt{\alpha_k \alpha_{k+1}}$ and compute the cross-validated error for the trees $T^{(r)}(\alpha'_k)$.
 - Select the tree T_k corresponding to the minimum cross-validated error.



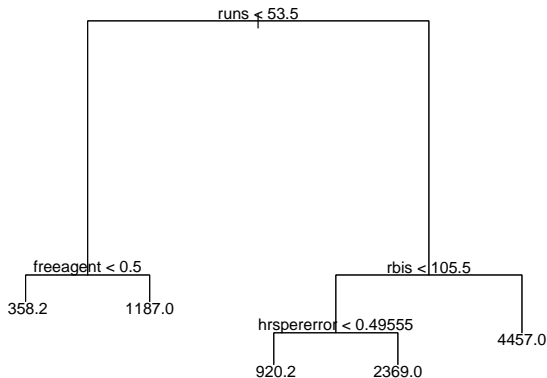
Pruning a regression tree in R

```
cv.baseball=cv.tree(tree.baseball)  
plot(cv.baseball$size,cv.baseball$dev,type='b')
```



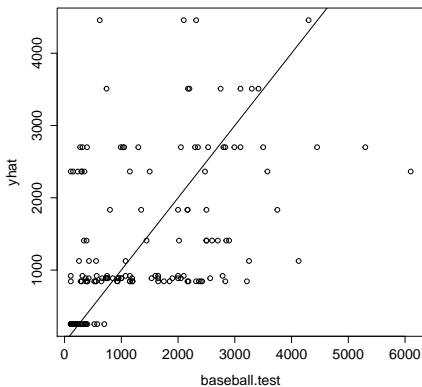
Pruning a regression tree in R

```
prune.baseball=prune.tree(tree.baseball,best=5)  
plot(prune.baseball)  
text(prune.baseball,pretty=0)
```



Prediction with a regression tree in R

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



Overview

Introductory example

Learning a regression tree

Tree building process

Pruning

Classification trees

Combining trees



Classification trees

- If the target is a classification outcome taking values $1, 2, \dots, K$, 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 squared-error node impurity measure Q_t ,

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

but this is not suitable for classification.



Impurity measures

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

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

the proportion of class k observations in node t .

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

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

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

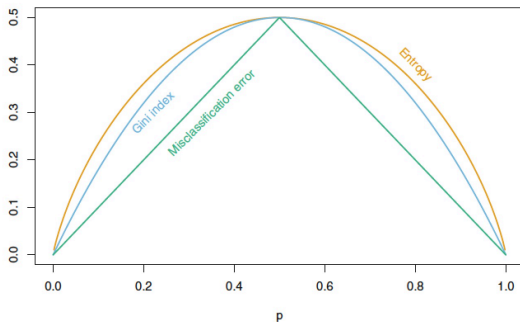
Cross-entropy or deviance: $-\sum_{k=1}^K \hat{p}_{tk} \log \hat{p}_{tk}$



Comparison between impurity measures

Case where $k = 2$

- 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 crossentropy and the Gini index are differentiable, and hence more amenable to numerical optimization.



Comparison between impurity measures

- In addition, cross-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 cross-entropy are lower for the second split. For this reason, either the Gini index or cross-entropy should be used when growing the tree.
- To guide cost-complexity pruning, any of the three measures can be used, but typically it is the misclassification rate.



Interpretation of the Gini index

- The Gini index can be interpreted in two interesting ways.
- Rather than classify observations to the majority class in the node, we could classify them to class k with probability \hat{p}_{tk} . Then the training error rate of this rule in the node is $\sum_{k \neq k'} \hat{p}_{tk} \hat{p}_{tk'}$ – the Gini index.
- Similarly, if we code each observation as 1 for class k and zero otherwise, the variance over the node of this 0-1 response is $\hat{p}_{tk}(1-\hat{p}_{tk})$. Summing over classes k again gives the Gini index.



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 Shannon 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 classes 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 cross-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.



German credit scoring data

- 1000 cases of credit application, 300 refused.
- 20 predictors, mix of quantitative, qualitative (unordered) and qualitative (ordered)
- Examples of quantitative predictor: credit amount, age in years, number of existing credits at this bank, etc.
- Examples of unordered qualitative predictors: purpose (car, furniture/equipment, education, etc.), credit history (no credits taken/ all credits paid back duly, delay in paying off in the past, etc.)
- Example of ordered qualitative predictors: Present employment since (unemployed, less than 1 year, 1-4 years, 4-7 years, more than 7 years), etc.



Data preprocessing in R

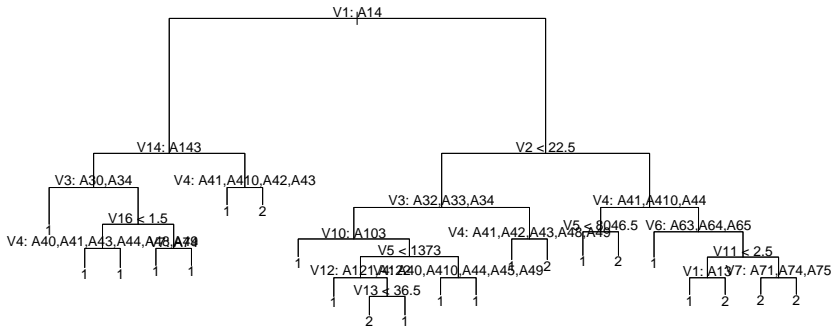
```
credit <- read.table("german.data")
credit$V7<-ordered(credit$V7)
credit$V17<-ordered(credit$V17)
credit$V21<-factor(credit$V21)

N<-nrow(credit)
train = sample(1:N, 700)
```



Tree growing in R

```
tree.credit=tree(as.factor(V21) ~.,credit,subset=train)
plot(tree.credit)
text(tree.credit,pretty=0)
```



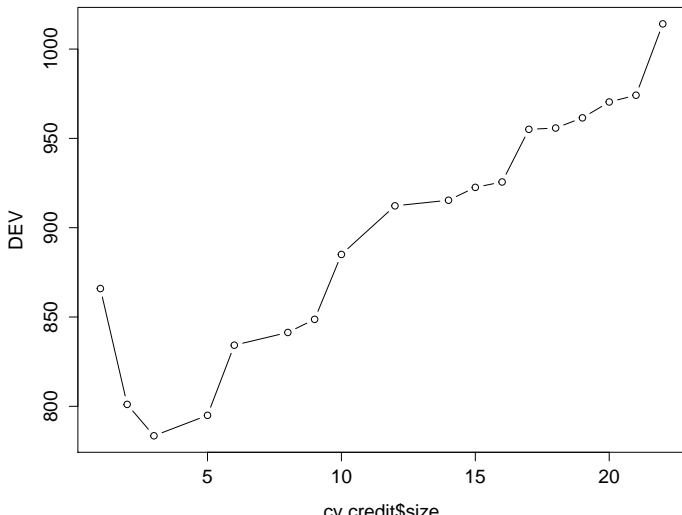
Cross-validation

```
Size<-cv.tree(tree.credit)$size
DEV<-rep(0,length(Size))
for(i in (1:10)){
  cv.credit=cv.tree(tree.credit)
  DEV<-DEV+cv.credit$dev
}
DEV<-DEV/10
plot(cv.credit$size,DEV,type='b')
```



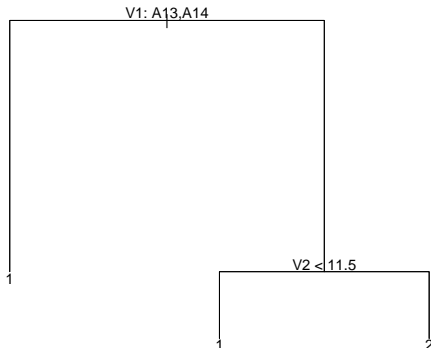
Cross-validation

Result



Pruning

```
prune.credit=prune.tree(tree.credit,best=3)  
plot(prune.credit)  
text(prune.credit,pretty=0)
```



Test error rate estimation

```
yhat=predict(prune.credit,newdata=credit[-train,],type='class')
y.test=credit[-train,"V21"]
table(y.test,yhat)
err<-1-mean(y.test==yhat)
```

- Confusion matrix:

	prediction	
true class	1	2
1	172	41
2	50	37

- Test error rate: $(41+50)/300=0.303$



Advantages and Disadvantages of Trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- 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 introduce these concepts next.



Overview

Introductory example

Learning a regression tree

Tree building process

Pruning

Classification trees

Combining trees



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 σ^2 , the variance of the mean \bar{Z} 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 $\hat{f}^{*b}(x)$, the prediction at a point x . We then average all the predictions to obtain

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

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.

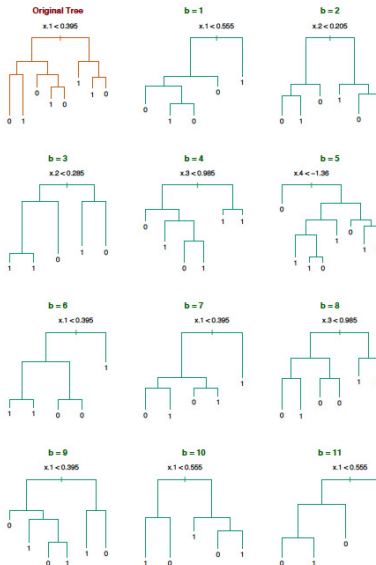


Example

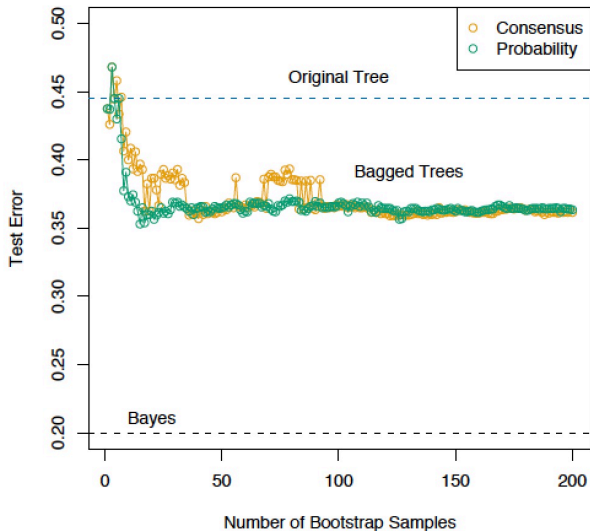
- We generated a sample of size $N = 30$, with two classes and $p = 5$ features, each having a standard Gaussian distribution with pairwise correlation 0.95.
- The response Y was generated according to $Pr(Y = 1|x_1 \leq 0.5) = 0.2$, $Pr(Y = 1|x_1 > 0.5) = 0.8$. The Bayes error is 0.2.
- A test sample 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 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 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.

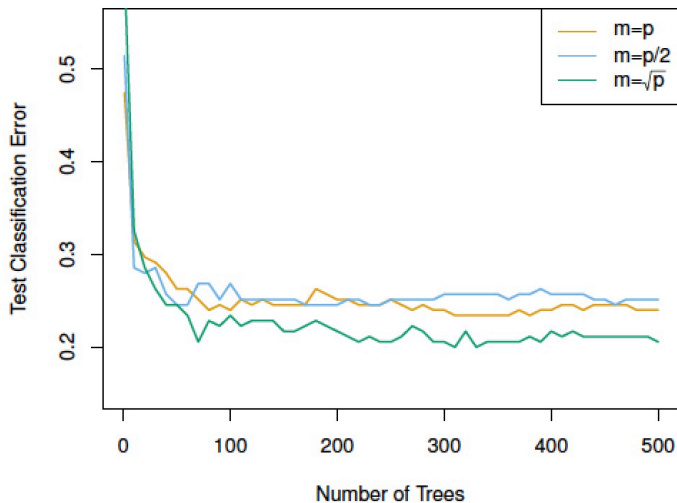


Example: gene expression data

- 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)
bag.credit=randomForest(V21 ~.,data=credit,subset=train,mtry=20)

yhat1=predict(bag.credit,newdata=credit[-train,],type='response')
table(y.test,yhat1)
1-mean(y.test==yhat1)
```

- Confusion matrix:

true class	prediction	
	1	2
1	193	20
2	48	39

- Test error rate: $(20+48)/300=0.227$



Random forests in R

```
library(randomForest)
bag.credit=randomForest(V21 ~.,data=credit,subset=train,mtry=5)

yhat2=predict(bag.credit,newdata=credit[-train,],type='response')
table(y.test,yhat2)
1-mean(y.test==yhat2)
```

- Confusion matrix:

true class	prediction	
	1	2
1	202	11
2	60	27

- Test error rate: $(11+60)/300=0.237$

