DATA11002 Introduction to Machine Learning

Kai Puolamäki

18 November 2020

Algorithmic supervised learning

Algorithmic supervised learning

- To be covered:
  - k-nearest neighbor (k-NN) classifier / regression
  - decision trees / regression trees
- "Algorithmic supervised learning"
  - compared to probabilistic, "spectral"
- Probabilistic models covered earlier:
  - Sec. 4 ("Classification"), including **logistic regression**, **LDA**, and **QDA**
  - In addition, we discussed **Naive Bayes** (NB)

Nearest neighbor classifiers

Similarity

- Think of suitable similarity measures for:
  - handwritten digits
  - segments of dna (can be thought of strings “TCGATTGC” etc.)
  - text documents (e.g., news articles)

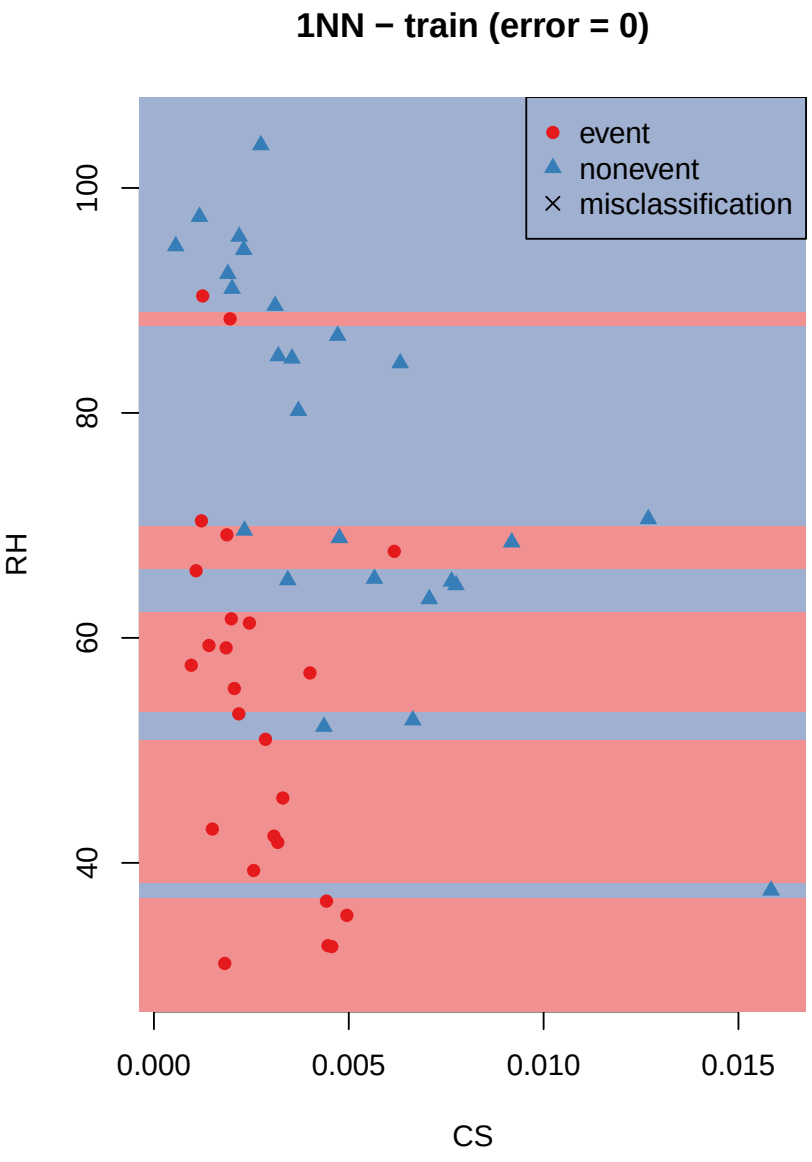
Similarity and dissimilarity

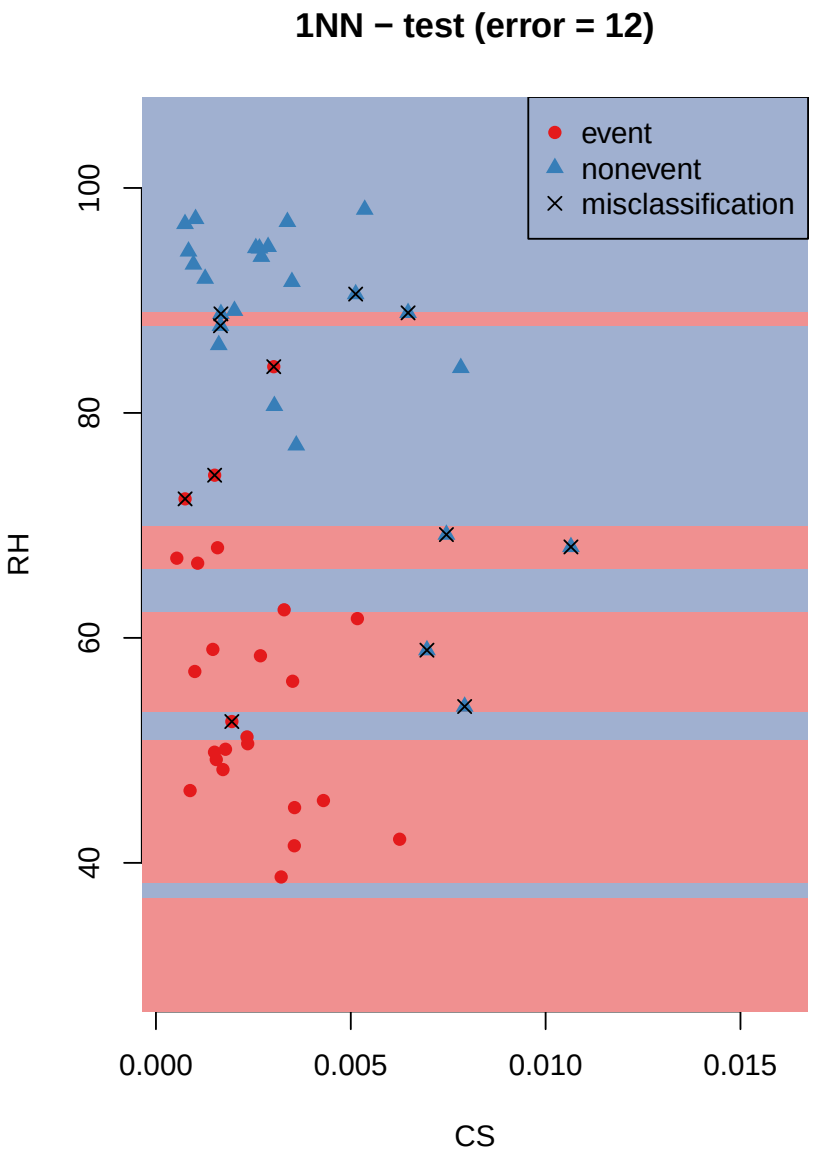
- **Similarity:**
  - Numerical measure of the degree to which two objects are alike
  - Higher for objects that are alike
  - Typically between 0 (no similarity) and 1 (completely similar)
- **Dissimilarity:**
  - Numerical measure of the degree to which two objects are different
  - Higher for objects that are different
  - Typically between 0 (no difference) and inf (completely different)

Nearest neighbor classifier

- Nearest-neighbor classifier is a simple geometric model based on distances:
  - store all the training data
  - to classify a new data point, find the closest one in the training set and use its class
- More generally, k-nearest-neighbor classifier finds k nearest points in the training set and uses the majority class (ties are broken arbitrarily)
- Different notions of distance can be used, but Euclidean is the most obvious

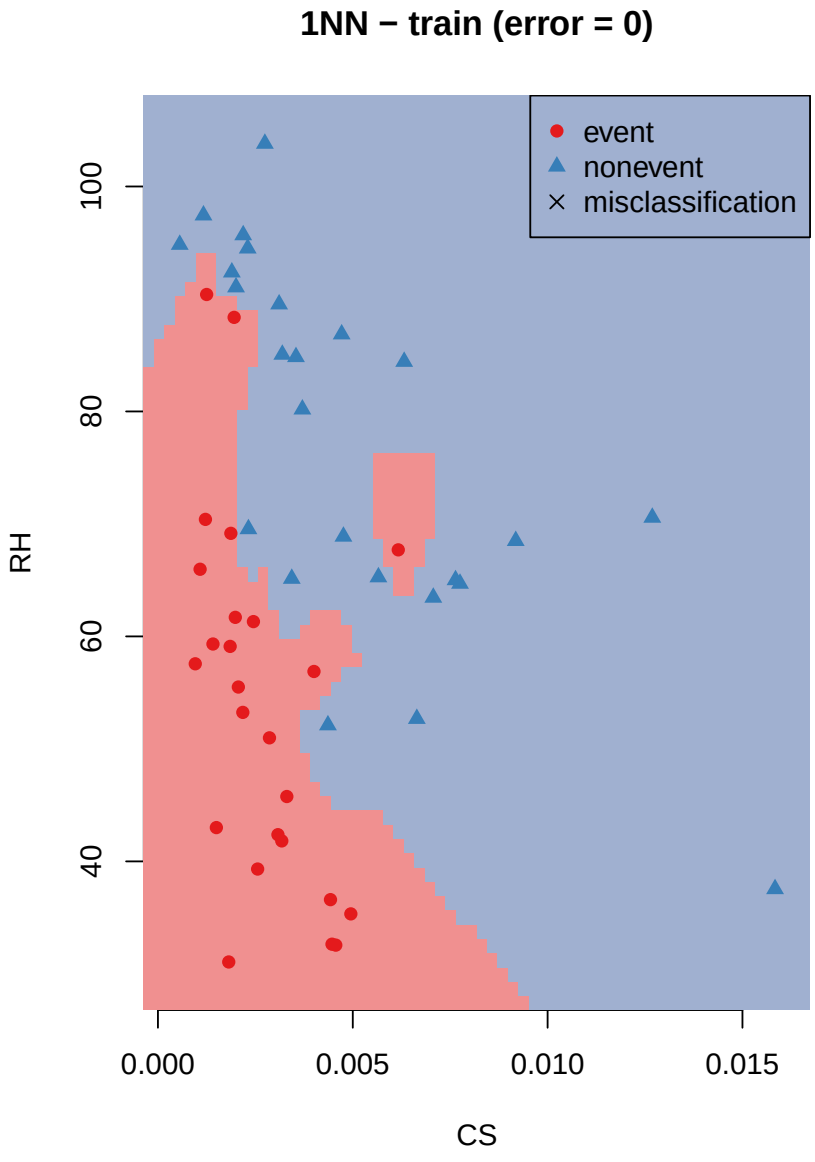
Nearest neighbor classifier

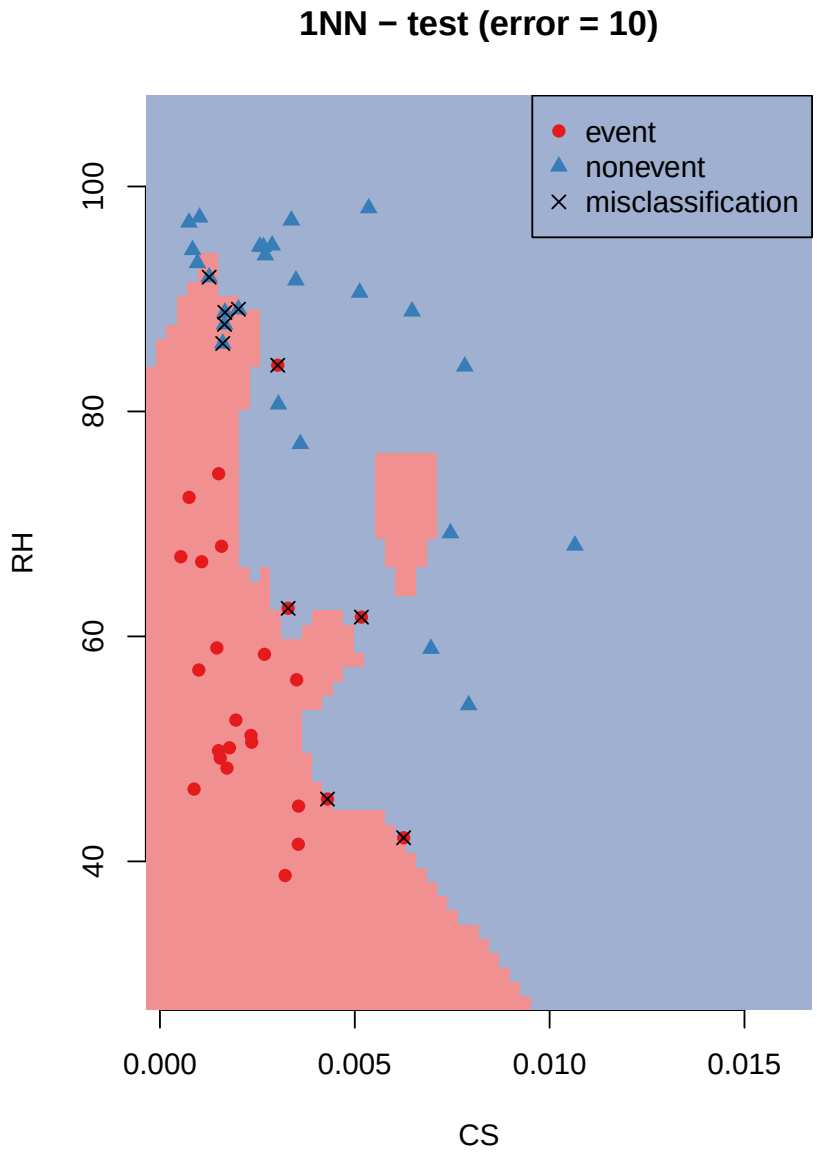




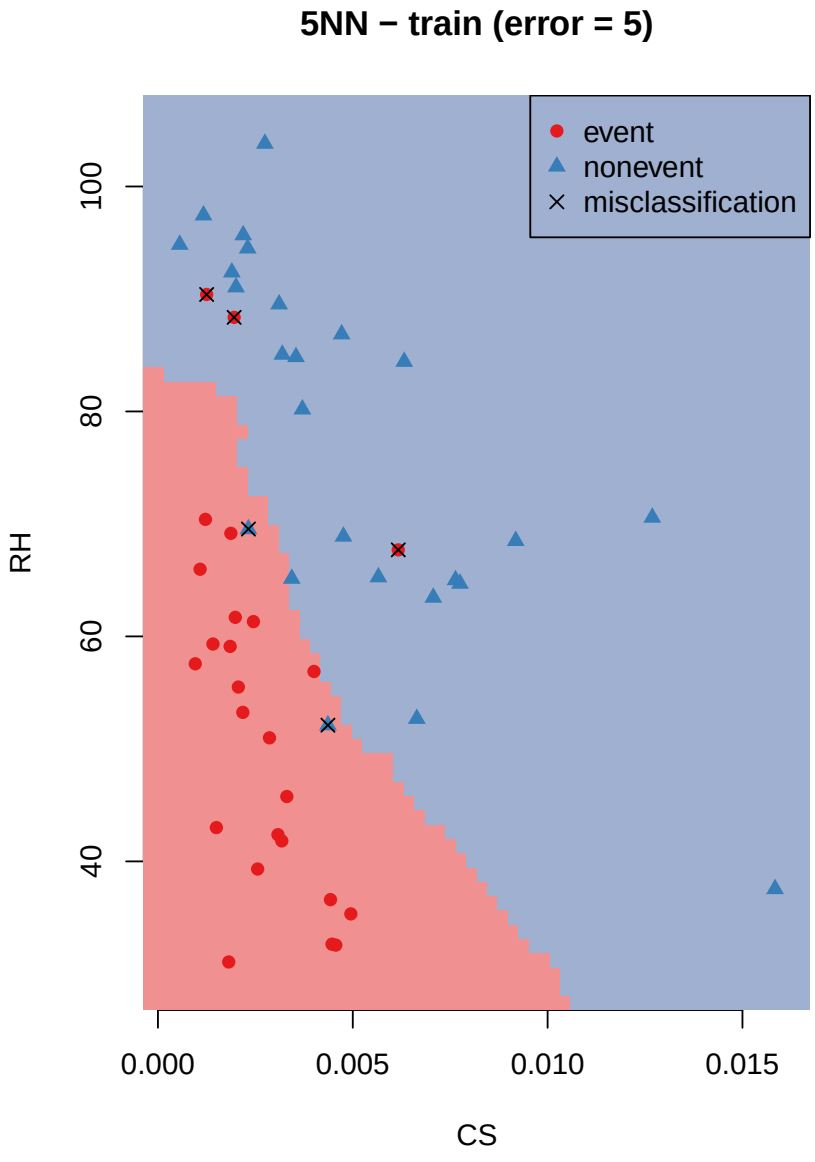
Did something go wrong here. . . ?

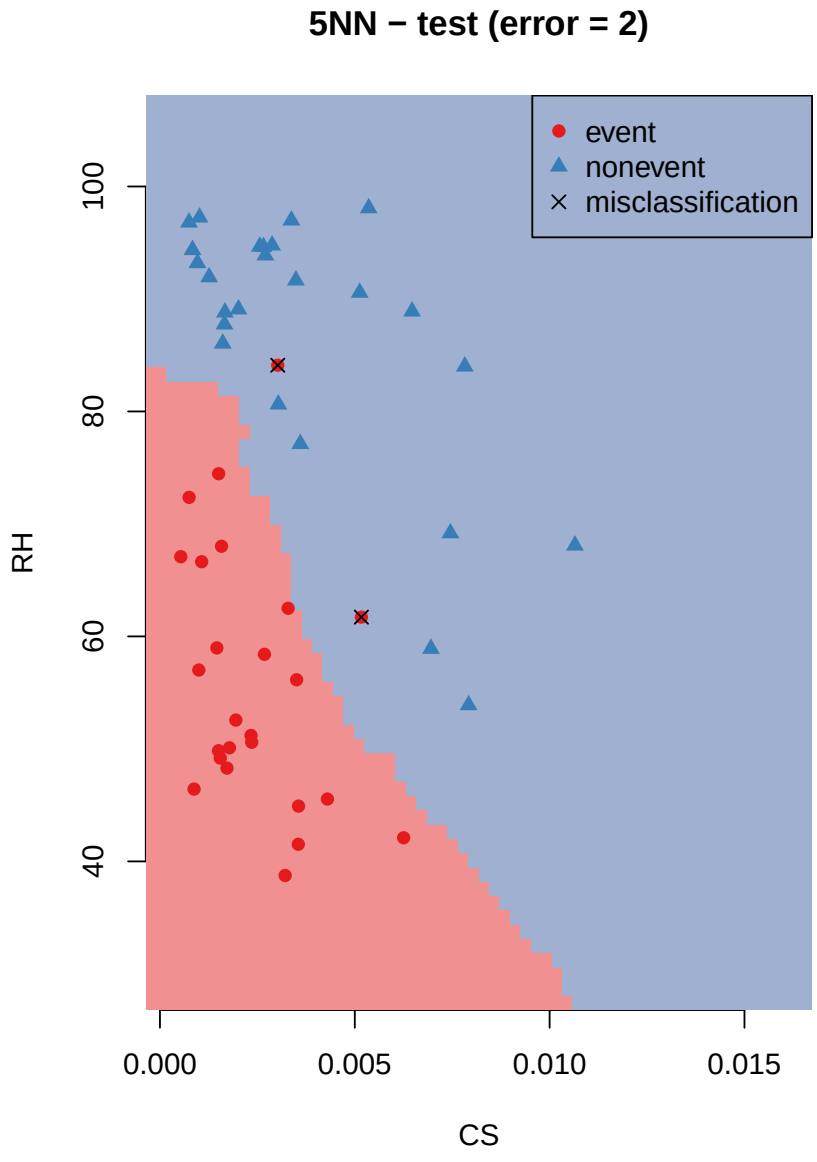
Nearest neighbor classifier



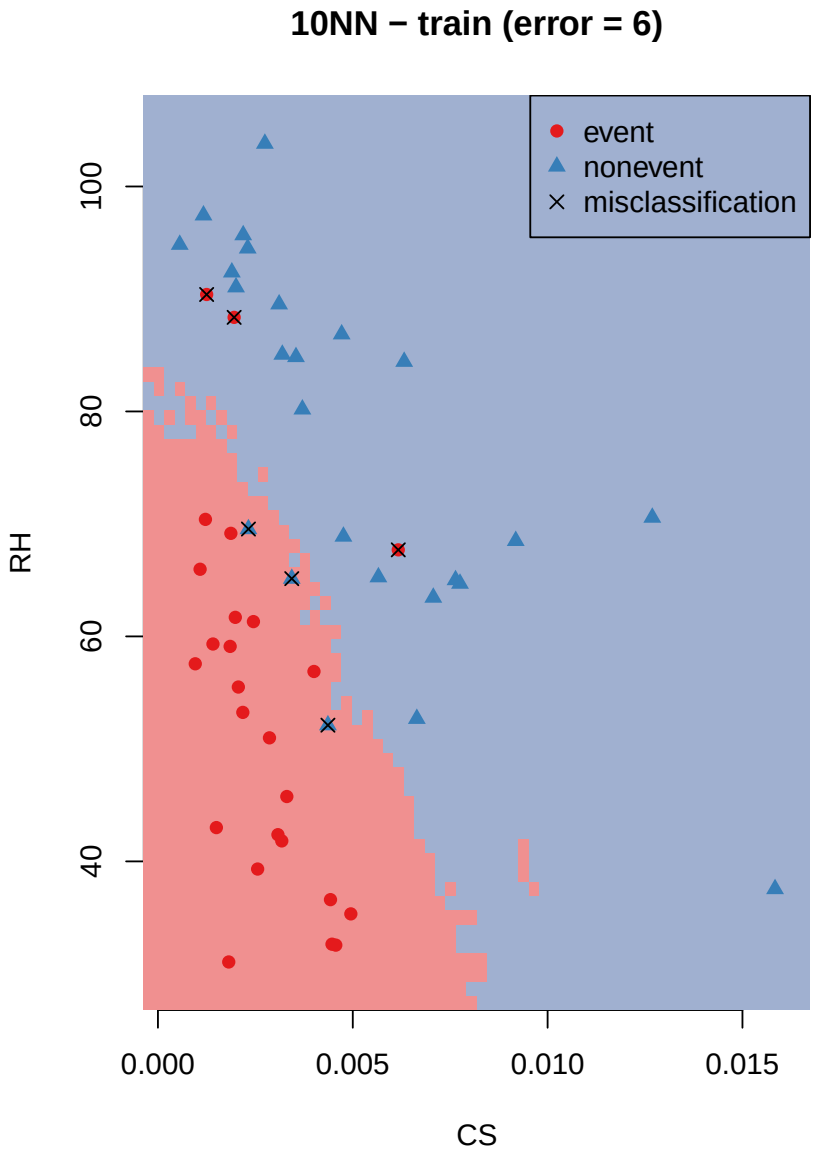


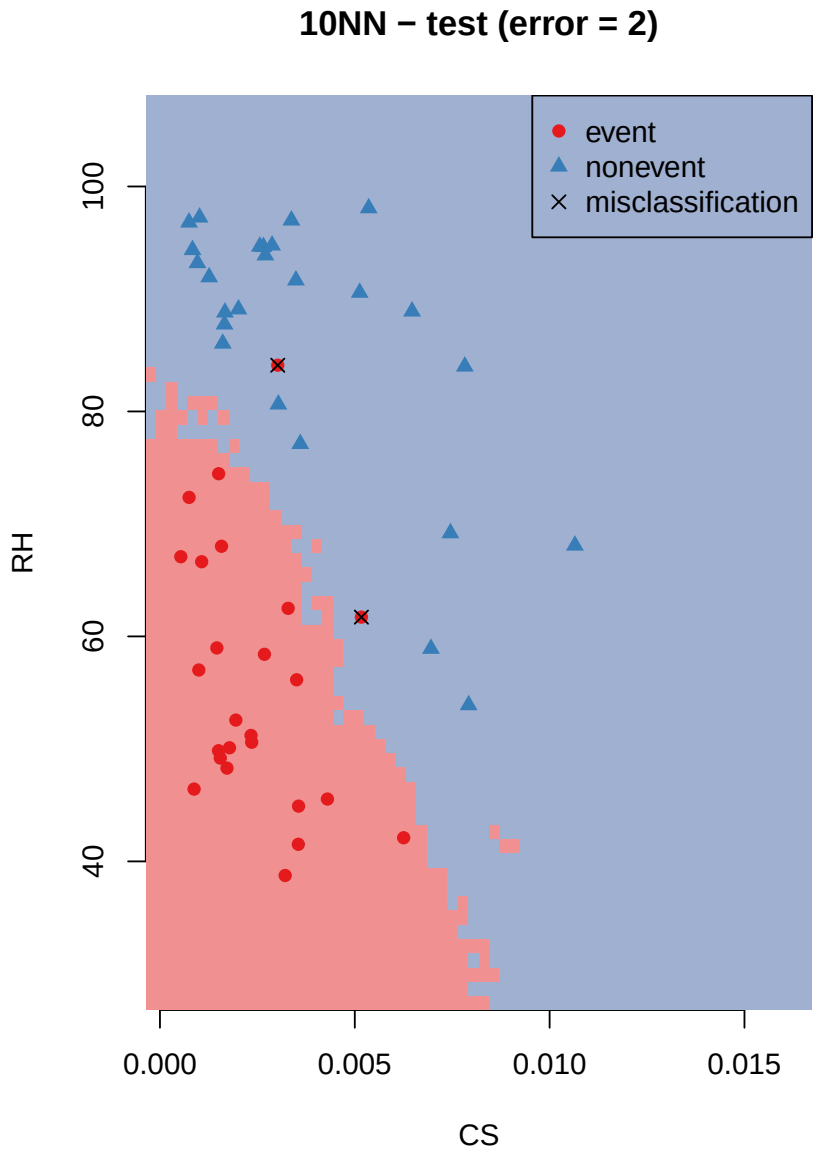
Nearest neighbor classifier





Nearest neighbor classifier





Bayes risk

- Data $\{(y_i, x_i)\}_{i=1}^n$, $y_i \in \{0, 1\}$, sampled i.i.d. from some distribution.
- **Assumption:** We have complete knowledge of the underlying distributions.
- We know (or can compute) $P(y \mid x)$.
- **Optimal classifier:** $\hat{y}(x) = \arg\max_j P(y = j \mid x)$.
- Denote by $\theta(x) = \max_j P(y = j \mid x)$
- *Bayes risk* is the probability of mis-classification, or $\mathcal{E}^*(x) = 1 - \theta(x)$.
- No classifier can have, on average, lower risk.

k-NN theory (extra)

- Assume there exists a distance $d(x, x')$ in covariate space
- **Assumption:** We have no knowledge of underlying distribution (except that we see the n data points).
- Denote the nearest neighbor of x by x_N and its class by y_N, $(y_N, x_N) = \arg\min_{\{(y_i,x_i)\}_{i=1}^n} d(x, x_i)$.
- **Nearest Neighbor Classifier:** $\hat{y}(x) = y_N$. (i.e., class of x is predicted to be the class of x_N)
- Let risk, or probability of misclassification, given x, of NN be $\mathcal{E}(x)$
- **Claim:** $\mathcal{E}(x) \leq 2\mathcal{E}^*(x)$ when $n \to \infty$.

k-NN theory (extra)

- **Assumption:** $n \to \infty$.
- **Assumption:** $P(y \mid x) \to P(y \mid x')$ when $d(x, x') \to 0$.
- **Assumption:** $P(y, y' \mid x, x') = P(y \mid x)P(y' \mid x')$ (i.i.d. assumption)
- $\mathcal{E}(x) = 1 - \sum_{i=0}^{1} P(y = i, y_N = i \mid x, x_N)$ [probability of misclassification]
- $\mathcal{E}(x) = 1 - \sum_{i=0}^{1} P(y = i \mid x)P(y_N = i \mid x_N)$ [iid]
- $\mathcal{E}(x) = 1 - \sum_{i=0}^{1} P(y = i \mid x)^2$ [as $n \to \infty$ $d(x, x_N) \to 0$]
- $\mathcal{E}(x) = 1 - \theta(x)^2 - (1 - \theta(x))^2 = 2\theta(x)(1 - \theta(x))$ [use $\theta(x) = \max_j P(y = j \mid x)$]
- $2\mathcal{E}^*(x) - \mathcal{E}(x) = 2(\theta(x) - 1)^2 \geq 0$.
- See, e.g., Cover et al. (1967) Nearest Neighbour Pattern Classification. IEEE Transactions on Information Theory.

k-NN theory (extra)

- Find k nearest neighbors of x
- Predict the class of x to be the majority class of the k nearest neighbors
- If $k \to \infty$ and $k/n \to 0$ as $n \to \infty$, then the error rate of k-NN approaches $\mathcal{E} \to \mathcal{E}^*$.

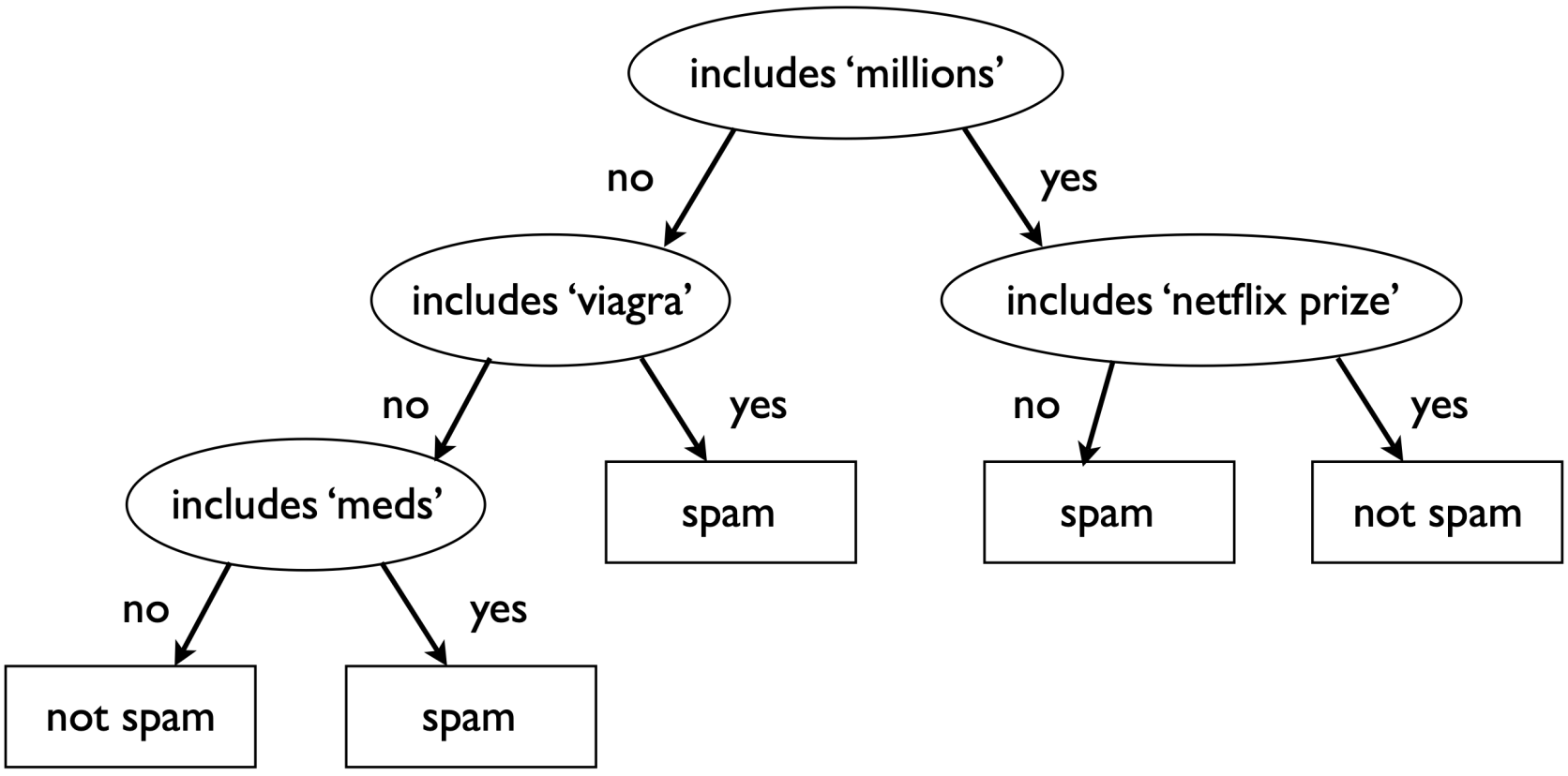
Pros and cons

- (k-)NN classifiers are simple
- They have nice theoretical properties
  - e.g., they are optimal classifiers asymptotically under some weak assumptions when $n \to \infty$, $k \to \infty$, and $k/n \to 0$
- They are quite powerful
- They have a large time and memory complexity (for example, you must store the training data)
- There is no model (sometimes the model parameters tell something of the data)

Tree based classifiers

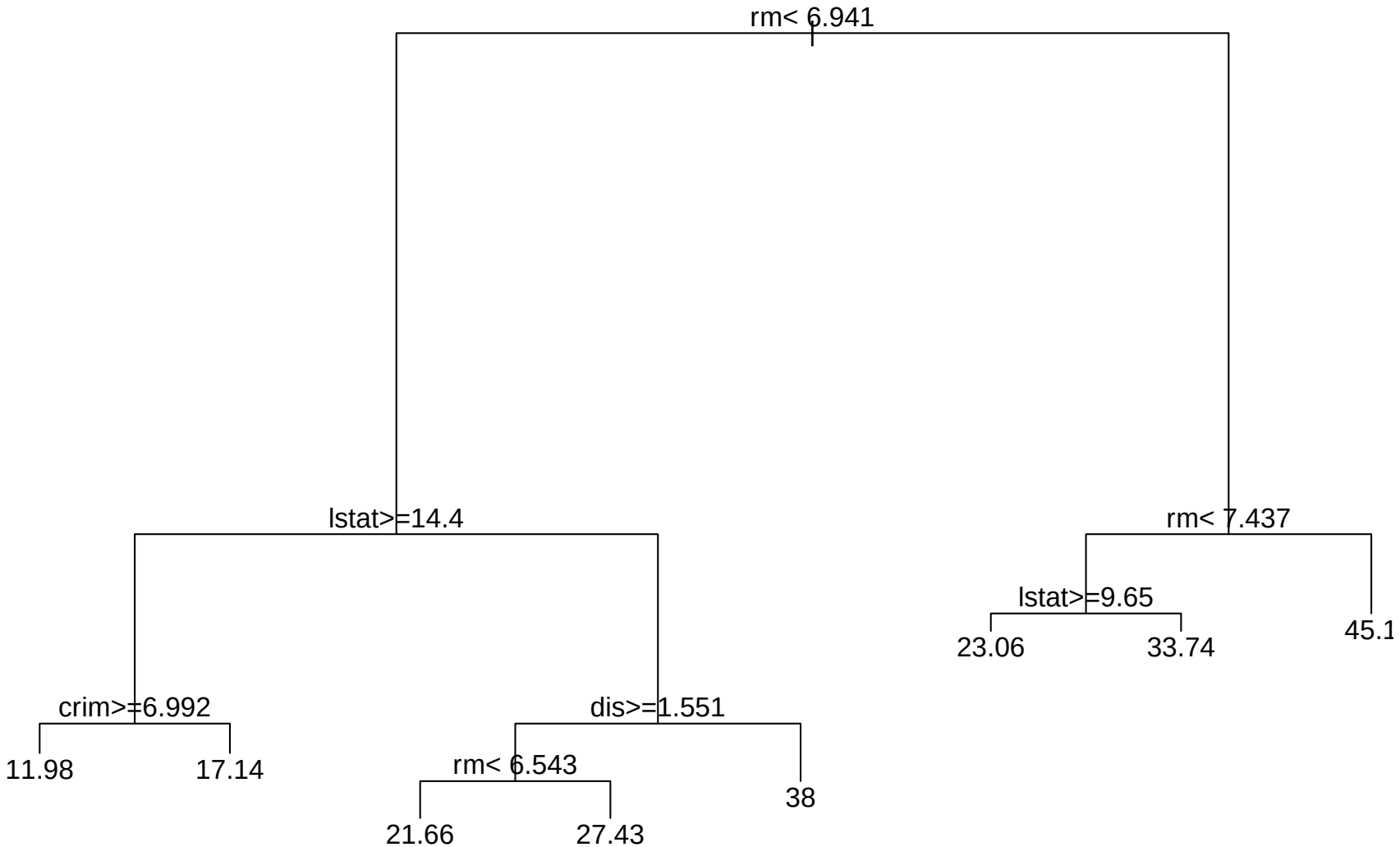
Decision trees

- Idea: Ask a sequence of questions to infer the class



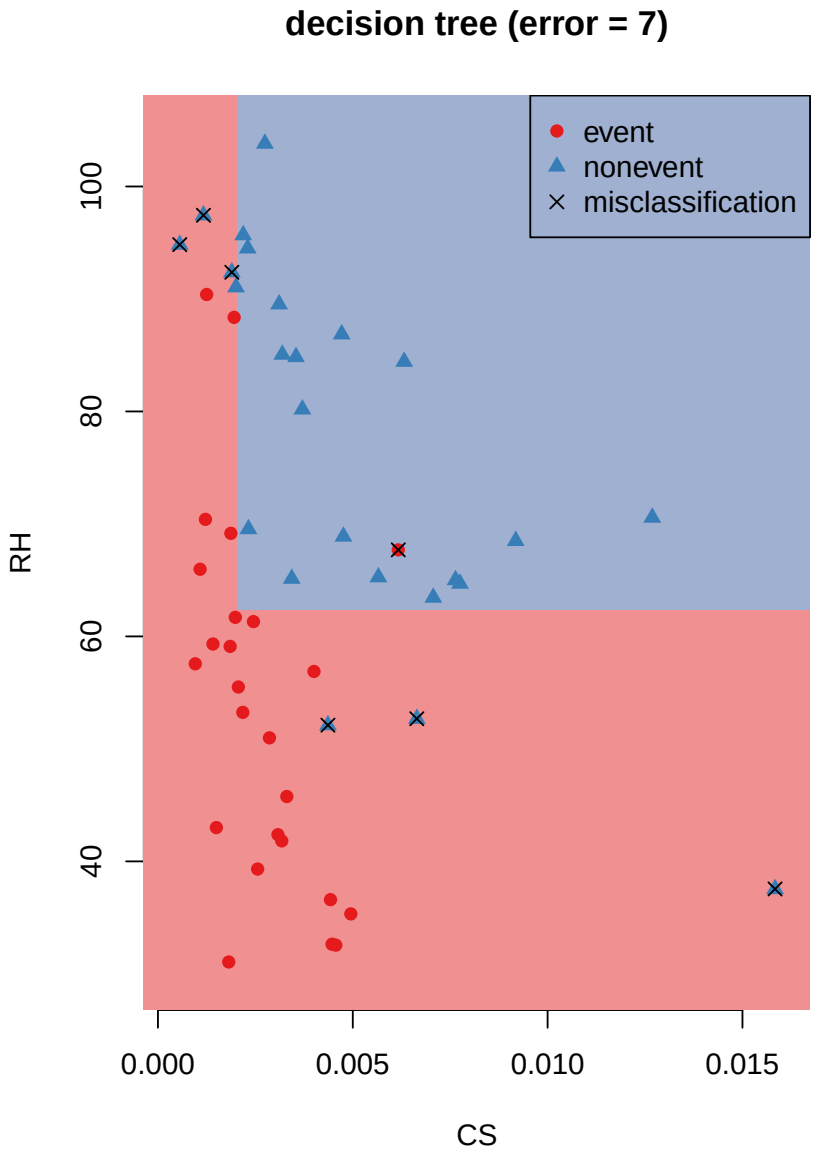
Regression trees

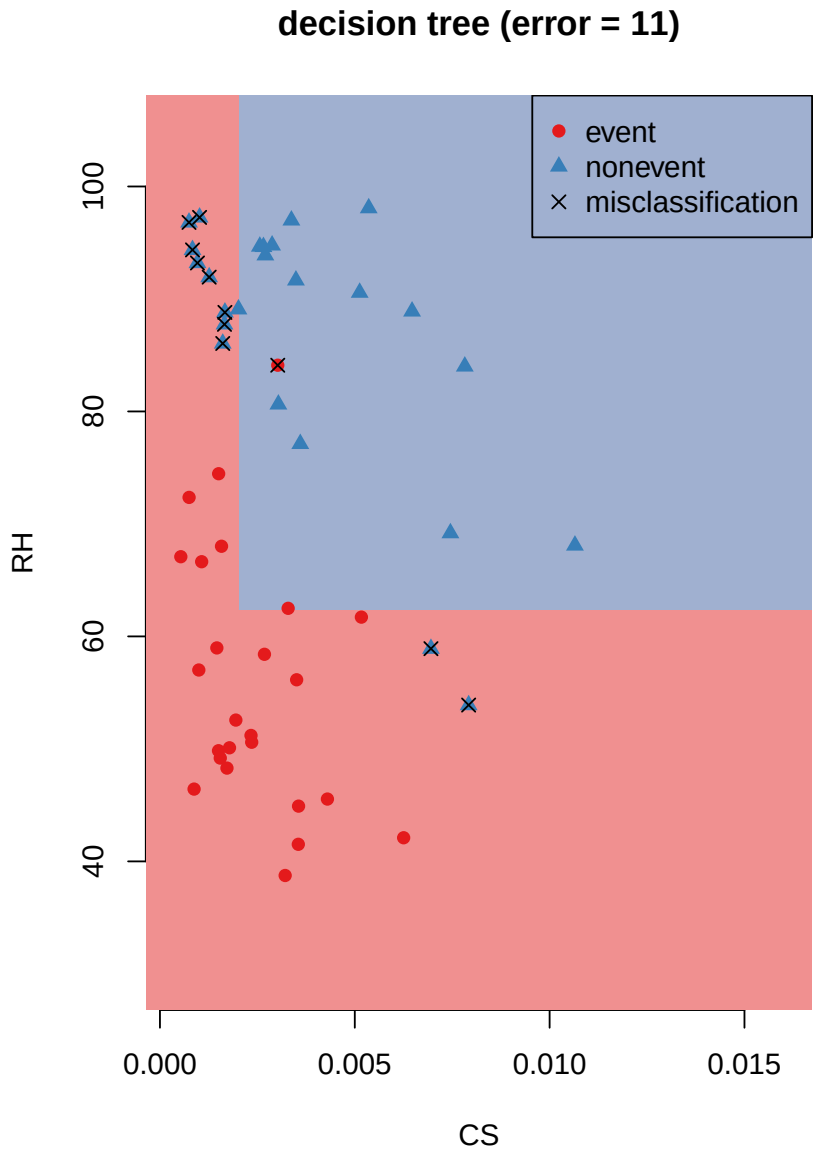
```
library(MASS); library(rpart)
m <- rpart(medv ~ .,Boston)
plot(m) ; text(m)
```



Boston dataset

Decision trees



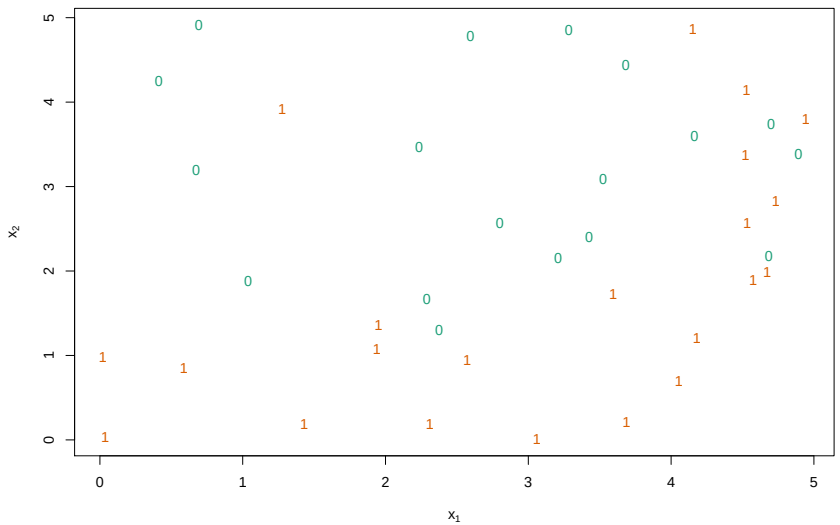


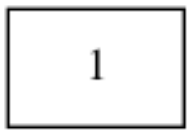
Decision trees: structure

- Structure of the tree:
  - A single *root node* with no incoming edges, and zero or more outgoing edges (where edges go downwards)
  - *Internal nodes*, each of which has exactly one incoming edge and two or more outgoing edges
  - *Leaf* or *terminal nodes*, each of which has exactly one incoming edge and no outgoing edges
- Node contents:
  - Each terminal node is assigned a prediction (here, for simplicity: a definite class label).
  - Each non-terminal node defines a test, with the outgoing edges representing the various possible results of the test (here, for simplicity: a test only involves a single feature)

Learning a decision tree: general idea

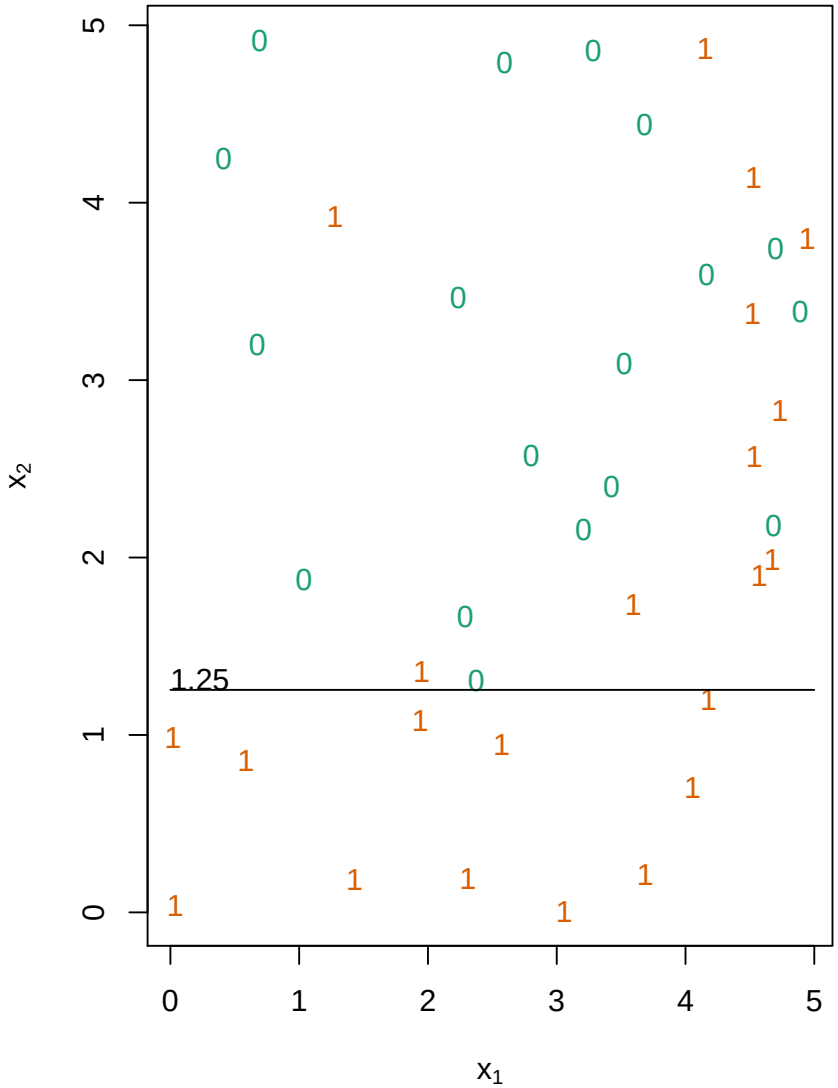
- Simple idea: recursively divide up the space into pieces which are as *pure* as possible

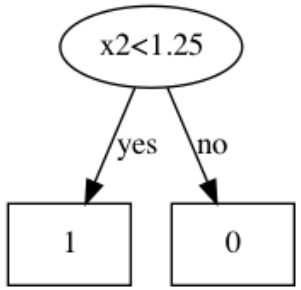




Learning a decision tree: general idea

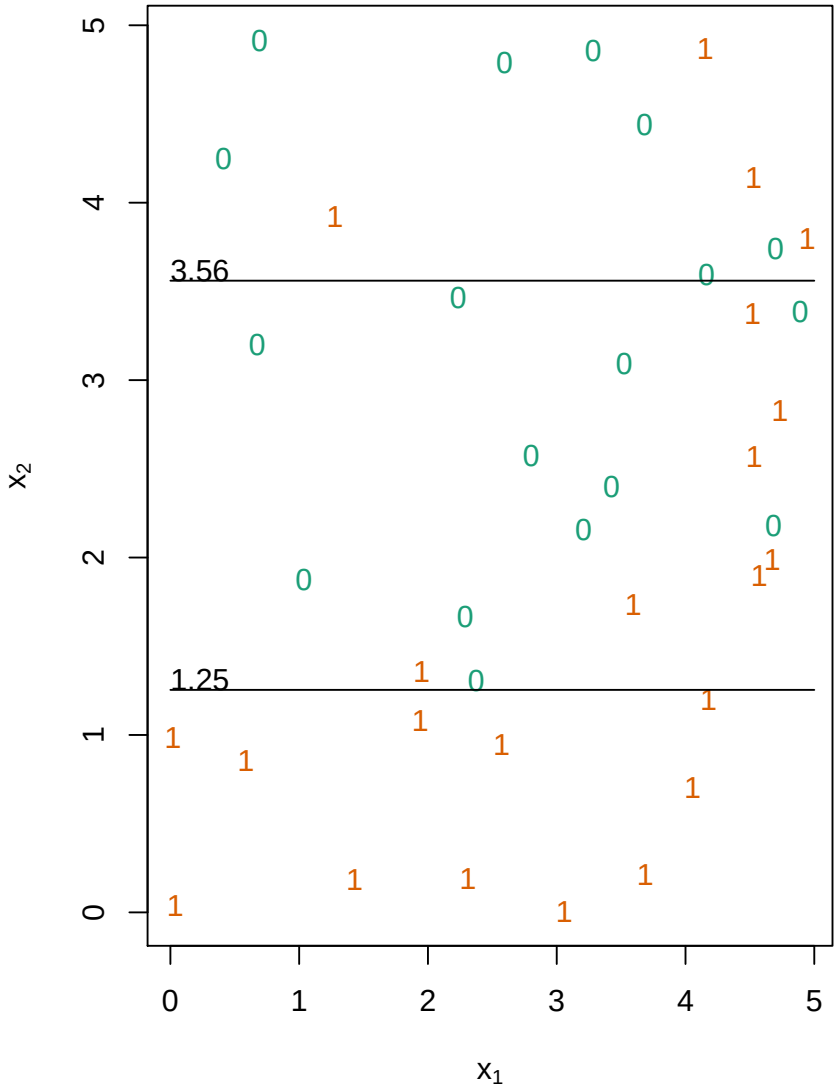
- Simple idea: recursively divide up the space into pieces which are as *pure* as possible

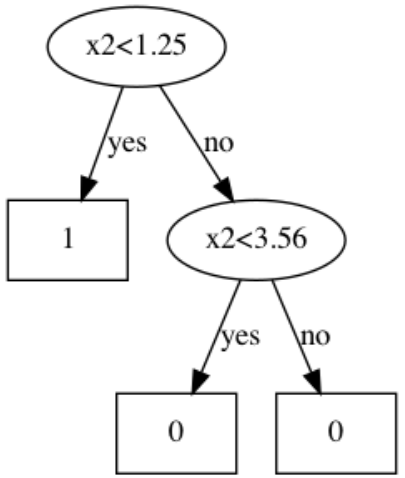




Learning a decision tree: general idea

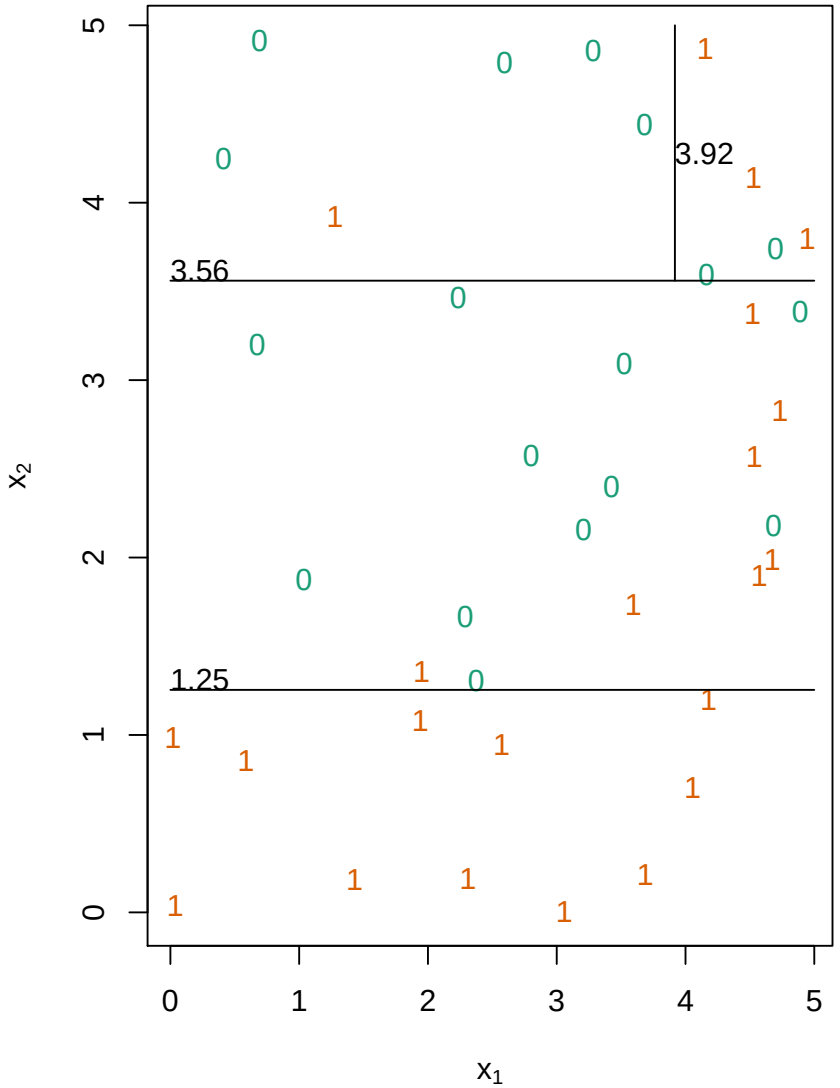
- Simple idea: recursively divide up the space into pieces which are as *pure* as possible

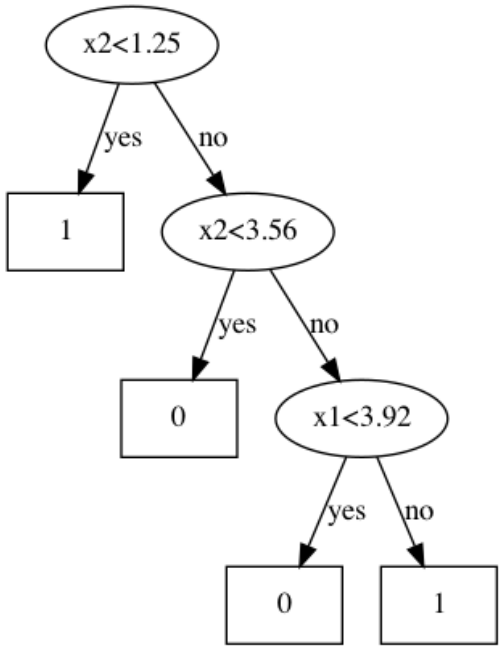




Learning a decision tree: general idea

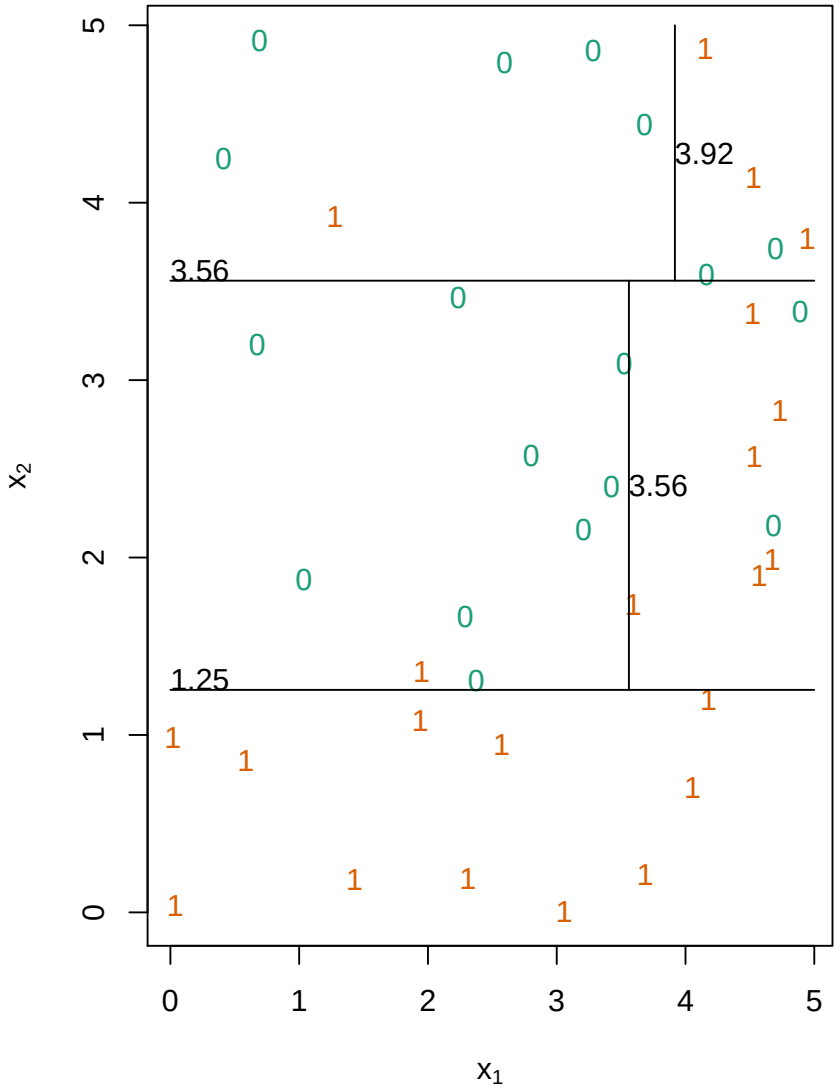
- Simple idea: recursively divide up the space into pieces which are as *pure* as possible

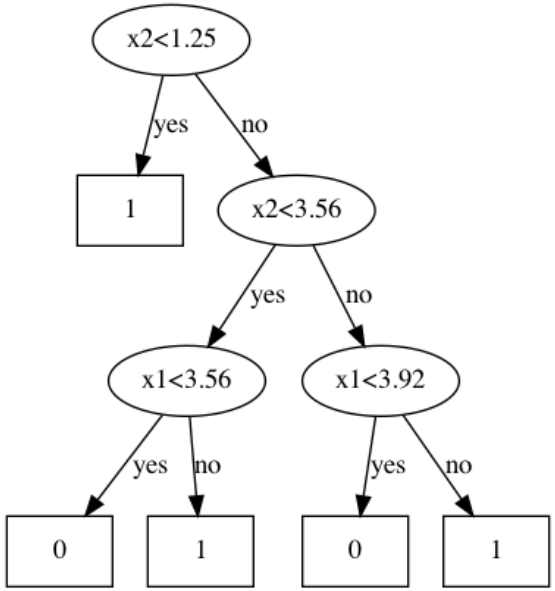




Learning a decision tree: general idea

- Simple idea: recursively divide up the space into pieces which are as *pure* as possible





Decision tree vs linear classifier (e.g., LDA)

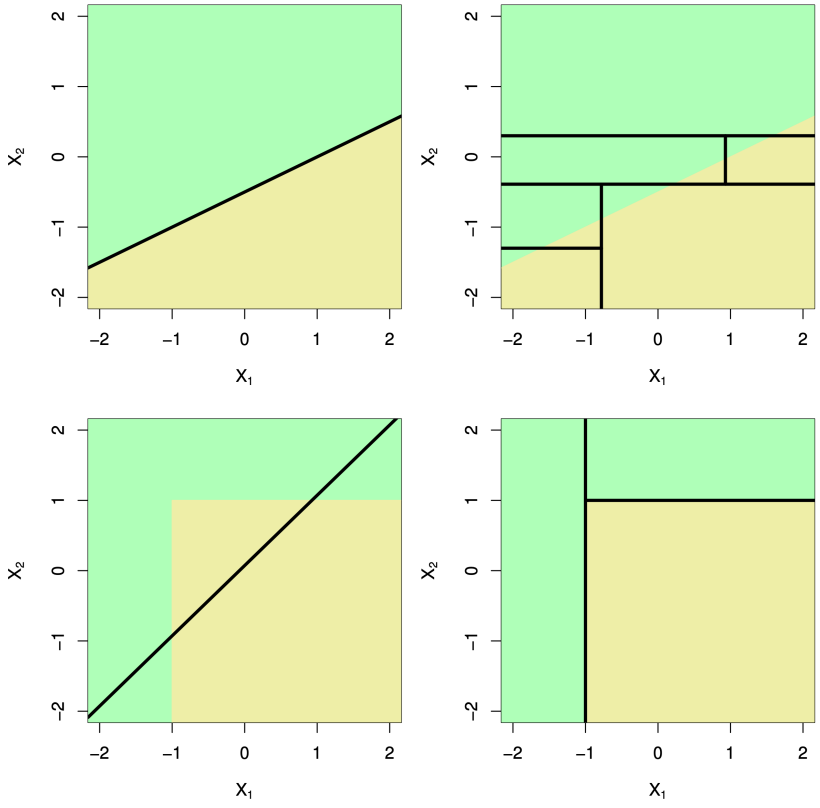


Fig. 8.7 from James et al.

- Two continuous features X_1, X_2; Top: true model linear; Bottom: true model "rectangular"
- Left: linear classifier; Right: (small) decision tree

Regression trees

- The textbook (Sec. 8) first presents regression trees and then classification trees
- The idea in regression trees is to predict a continuous outcome Y by a representative (usually the average) outcome $\overline{y}$ within each leaf
- Otherwise, the idea is the same
- (See the textbook to learn more about regression trees if you wish; not required in this course)
- We cover classification trees a some more detail than the book: e.g., splitting criteria and pruning are only superficially defined in the book

Examples: Predicting woody cover in African savannas

- Task: woody cover (% of surface covered by trees) as a function of precipitation (MAP), soil characteristics (texture, total nitrogen total and phosphorus, and nitrogen mineralization), fire and herbivory regimes.
- Result: MAP (mean average precipitation) is the most important factor.

Figure 4 | The distributions of MAP-determined ('stable') and disturbance-determined ('unstable') savannas in Africa. Grey areas represent the existing distribution of savannas in Africa according to ref. 30. Vertically hatched areas show the unstable savannas (>784 mm MAP); cross-hatched areas show the transition between stable and unstable savannas (516–784 mm MAP); grey areas that are not hatched show the stable savannas (<516 mm MAP).

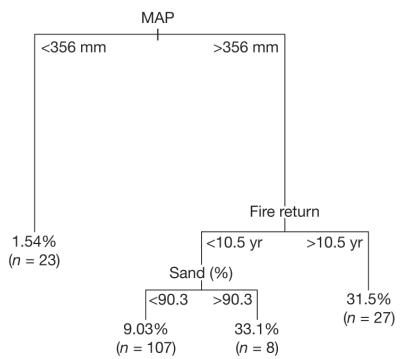


Figure 3 | Regression tree showing generalized relationships between woody cover and MAP, fire-return interval and percentage of sand. The tree is pruned to four terminal nodes and is based on 161 sites for which all data were available. No consistent herbivore effects were detected. Branches are labelled with criteria used to segregate data. Values in terminal nodes represent mean woody cover of sites grouped within the cluster. The pruned tree explained ~45.2% of the variance in woody cover, which is significantly more than a random tree ($P < 0.001$). Of this, 31% was accounted for by the first split; the second split explained an additional 10% of the variance in woody cover.

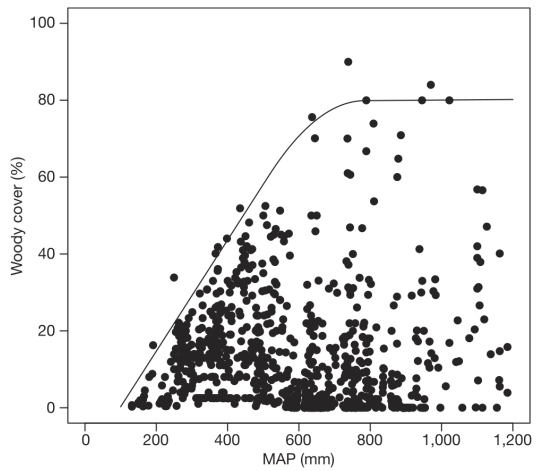


Figure 1 | Change in woody cover of African savannas as a function of MAP. Maximum tree cover is represented by using a 99th quantile piec wise linear regression. The regression analysis identifies the breakpoint (t rainfall at which maximum tree cover is attained) in the interval 650 ± 134 mm MAP (between 516 and 784 mm; see Methods). Trees are typically absent below 101 mm MAP. The equation for the line quantifyi the upper bound on tree cover between 101 and 650 mm MAP is Cover(%) = 0.14(MAP) − 14.2. Data are from 854 sites across Africa.

From Sankaran et al. (2005) Nature.

Feature test conditions

- Binary features: yes / no (left / right)
- Nominal (categorical) features with L values:
  - Pick one value (left) vs other (right)—can also do arbitrary subsets
- Ordinal features with L states:
  - Split at a given value: $X_i \leq T$ (left) vs $X_i > T$ (right)
- Continuous features:
  - Same as ordinal
- NB: We could also use multiway (not only binary) splits

Impurity measures

Key part in choosing test for a node is purity of a subset D of training data

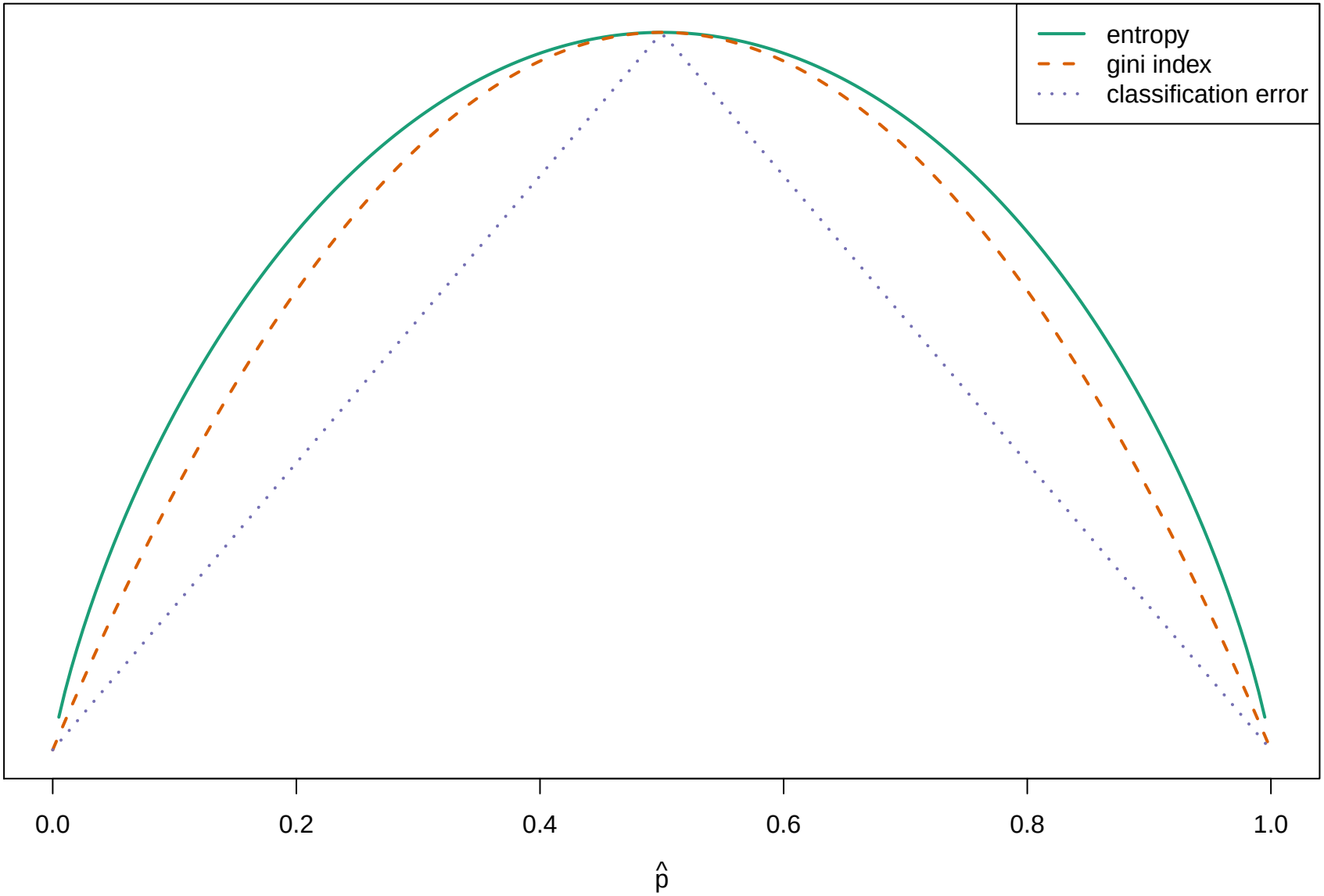
- Suppose there are K classes. Let $\hat{p}_{mc}$ be the fraction of examples of class c among all examples in node m.
- Impurity measures:

$$\text{(cross) entropy } D = -\sum_{c=1}^{K} \hat{p}_{mc} \log \hat{p}_{mc}$$

$$\text{gini index } G = \sum_{c=1}^{k} \hat{p}_{mc}(1 - \hat{p}_{mc})$$

$$\text{classification error } E = 1 - \max_c \hat{p}_{mc}$$

Impurity measures: binary classification



Gini and cross entropy are recommended and most commonly used.

Selecting the best split

- Let $Q(D)$ the **impurity of data set** D
- Assume a given features test splits D into subsets D_1 and D_2
- We define the **impurity of the split** as

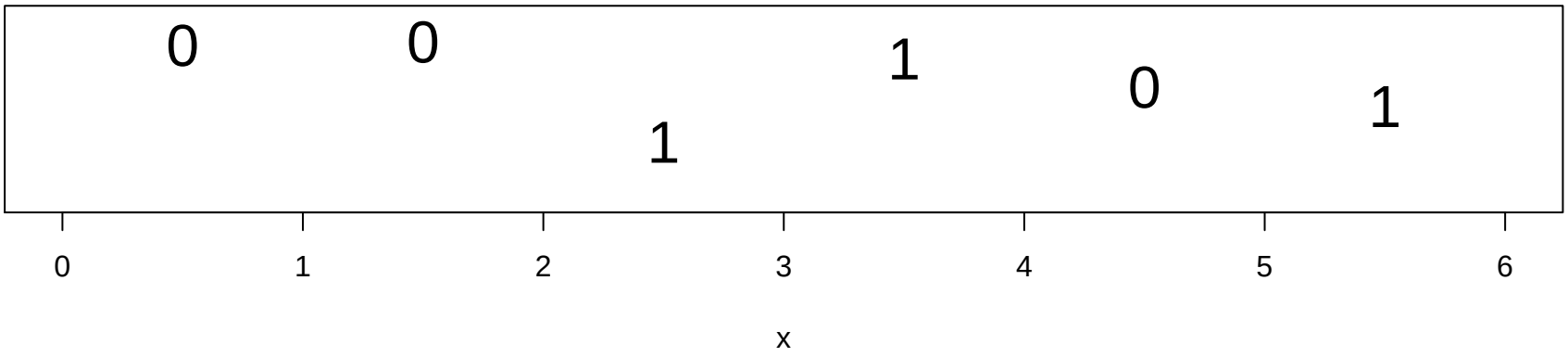
$$Q(\{D_1, D_2\}) = \sum_{i=1}^{2} Q(D_i) \frac{|D_i|}{|D|}$$

and the related **gain** as

$$Q(D) - Q(\{D_1, D_2\})$$

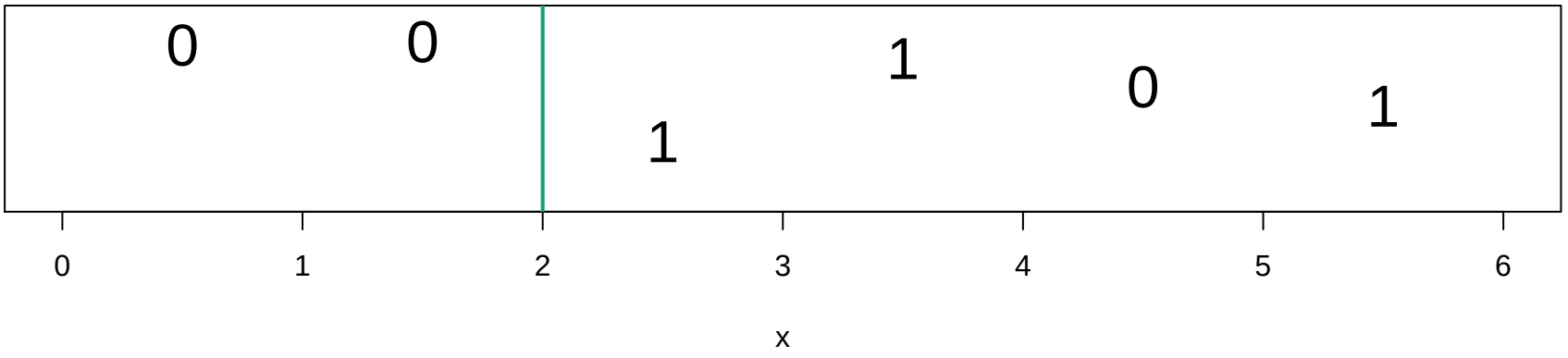
- Choose the split with the highest gain

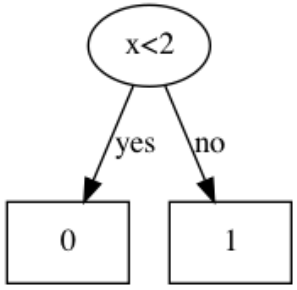
Selecting the best split: binary Gini



- Gini index for 0-1 binary data:
  $G(\hat{p}_{m1}) = \sum_{c=0}^{1} \hat{p}_{mc}(1 - \hat{p}_{mc}) = 2\hat{p}_{m1}(1 - \hat{p}_{m1})$
  - $\hat{p}_{m1}$ is the proportion of 1s within the split
- Impurity if no split: $G(3/6) = 0.5$
- *Gain* (reduction in impurity) for different splits:
  - $x = 1$: $G(3/6) - 1/6 \times G(0/1) - 5/6 \times G(3/5) = 0.10$
  - $x = 2$: $G(3/6) - 2/6 \times G(0/2) - 4/6 \times G(3/4) = 0.25$
  - $x = 3$: $G(3/6) - 3/6 \times G(1/3) - 3/6 \times G(2/3) = 0.06$
  - $x = 4$: $G(3/6) - 4/6 \times G(2/4) - 2/6 \times G(1/2) = 0.00$
  - $x = 5$: $G(3/6) - 5/6 \times G(2/5) - 1/6 \times G(1/1) = 0.10$

Selecting the best split: binary Gini





- The best gain is 0.25 with a split at $x = 2$
  - choose to split at $x = 2$
- Weighted impurity after the split:
  $2/6 \times G(0/2) + 4/6 \times G(3/4) = 0.25$ (before the split: 0.5)

Avoiding over-fitting

- The tree building process that splits until nodes have size n_{max} (e.g., 5), will overfit
- It would be possible to stop recursion once the subset D is "pure enough". This is known as pre-pruning but generally not recommended
- In contrast, post-pruning (called simply pruning in the textbook) is an additional step to simplify the tree after it has first been fully grown until $|D| \leq n_{max}$

Cost-complexity pruning

- Introduce regularization term α into the training error $R_\alpha(T) = \sum_{t \in f(T)} R(t) + \alpha |f(T)|$
  - $f(T)$ are the leaf nodes!
- Split data into training and validation (or use cross-validation) and choose the best α
- Train a model with the full data using the chosen α

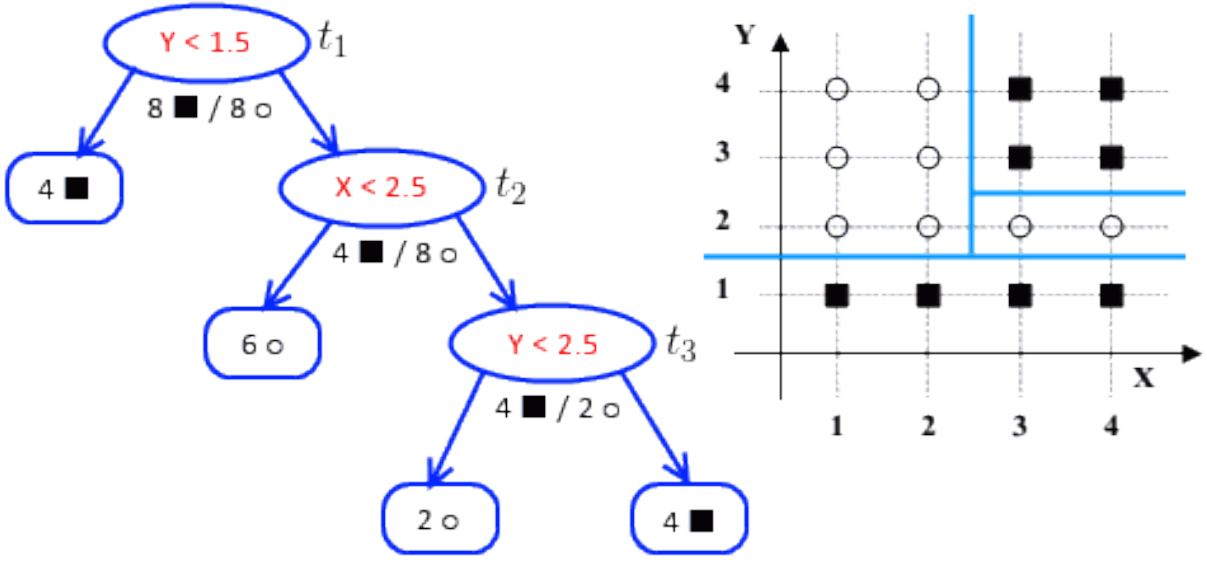


Figure from http://mlwiki.org/index.php/Cost-Complexity_Pruning

Cost-complexity pruning

- **Algorithm** to find solutions of $\arg\min_T (R(t) + \alpha|f(T)|)$ for all values of α: first find T_0 for $\alpha_0 = 0$ (e.g., using ID3).
- Then start increasing α. This results to a nested sequence of trees $T_0 \subset T_1 \subset \ldots \subset \{\text{root}\}$ (you "prune out" a subtree at each step) and corresponding sequence $\alpha_0 \leq \alpha_1 \leq \ldots \leq \ldots$.

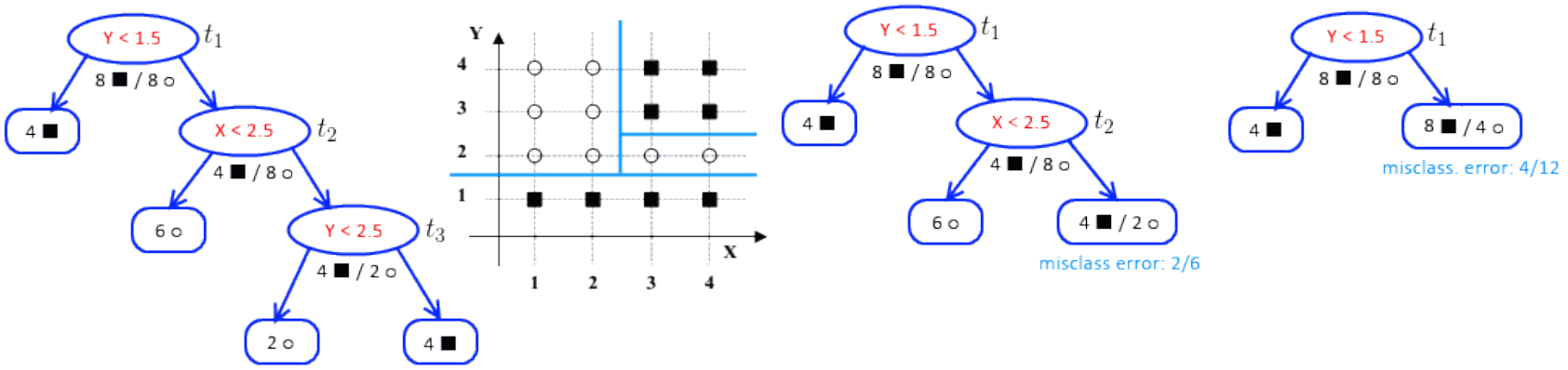
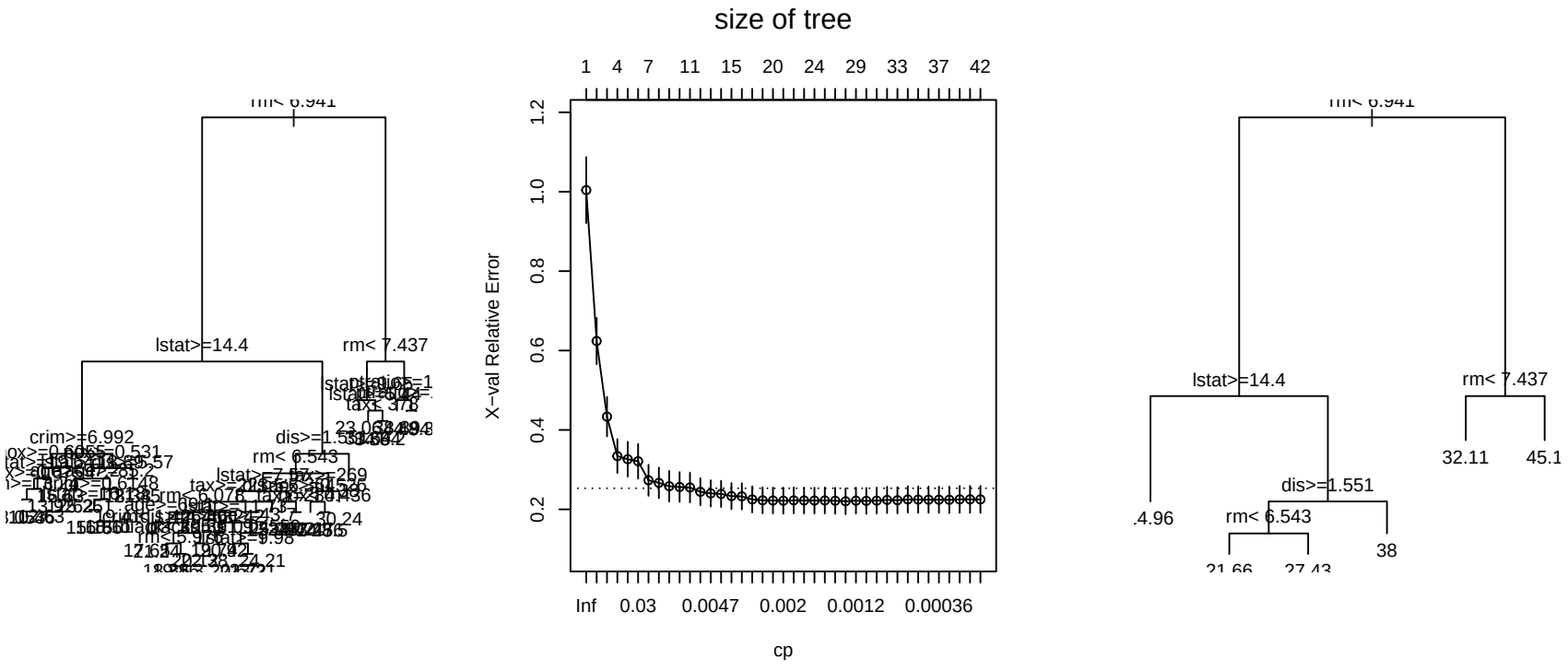


Figure from http://mlwiki.org/index.php/Cost-Complexity_Pruning

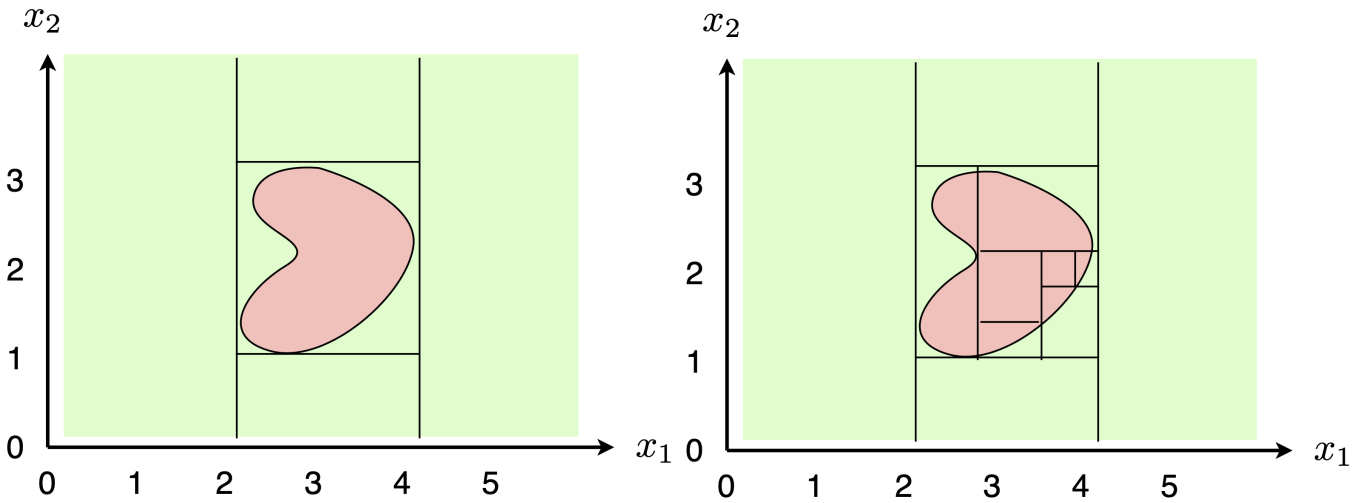
Cost-complexity pruning with Boston data and rpart

```
library(rpart); library(MASS); par(mfrow=c(1,3))
## Set alpha=0 in our initial tree (cp=0):
tt <- rpart(medv ~ ., data=Boston,control=rpart.control(cp=0))
plot(tt); text(tt) # fit & plot tree
plotcp(tt) # plot cost-complexity curve
ttp <- prune(tt,0.03) # CV results; apply "1-SE" rule, yields cp=0.03 in this case
plot(ttp); text(ttp)
```



Properties of decision trees

- Nonparametric approach:
  - If the tree size is unlimited, can in approximate any decision boundary to arbitrary precision (like k-NN)
  - So with infinite data could in principle always learn the optimal classifier
  - But with finite training data needs some way of avoiding overfitting (pruning) — bias–variance tradeoff!



Properties of decision trees

- Local, greedy learning to find a reasonable solution in reasonable time (as opposed to finding a globally optimal solution)
- Relatively easy to interpret (by experts or regular users of the system)
- Classification generally very fast (linear in depth of the tree)
- Usually competitive only when combined with ensemble methods: bagging, random forests, boosting

Ensemble methods for supervised learning

Ensemble methods for supervised learning

- Having several training samples, $D_1, \ldots, D_K$ would clearly be nice also for supervised learning
- We can combine the learned models $\hat{f}_1, \ldots, \hat{f}_K$ into an aggregate model $\hat{f}_{agg}$
- The aggregate model will have lower variance than the individual models
- If a learning method has high variance (but low bias), then $\hat{f}_{agg}$ may be a very good model
- **Bagging** = bootstrap aggregation: $\hat{f}_j = \hat{f}_j^*$ obtained from boostrap (Sec. 8.2.1)
  - *bootstrap* = sample a new dataset D_j^* ("bootstrap sample") of the same size randomly with replacement from the original data, then train a model $\hat{f}_j^*$ using the bootstrap sample

Bagging

1. Bootstrap to obtain $D_1^*, \ldots, D_K^*$ (bootstrap samples)
   - Bootstrap idea: dataset D_j^* is sampled in random with replacement from the original data
   - D_j^* is of the same size as the original dataset
2. Learn models (classifiers or regressors) $\hat{f}_1^*, \ldots, \hat{f}_K^*$ from the bootstrap samples
   - For example, unpruned decision trees (high variance, low bias)
3. Combine the predictions
   - for regression, average:

$$\hat{f}_{bag}(x) = \sum_{j=1}^{K} \hat{f}_j^*(x)/K$$

   - for classifiers, majority vote:

$$\hat{f}_{bag}(x) = \text{majority}\left(\hat{f}_1^*, \ldots, \hat{f}_K^*\right)$$

Random forest

- For decision trees, bagging tends to improve somewhat
- However, the trees are highly dependent in cases where the splits that maximize the gain are clearly better than the 2nd best splits
  - for example, always split according to X_3 first, then X_4 etc.
- The trees can be forced to use different features by only allowing splits based on a *random sample of the features*
- For example, select randomly about $\sqrt{p}$ of all features at each split: the feature of with maximum gain is usually outside this set
- Trees constructed in bagging or random forests are usually not pruned since averaging a large number of trees reduces over-fitting
- Random forests are one of the most powerful family of classifiers available (Fernandez-Delgado et al (2014) JMLR)

Random forests