T-61.3050 Machine Learning: Basic Principles

Decision Trees

Kai Puolamäki

Laboratory of Computer and Information Science (CIS)
Department of Computer Science and Engineering
Helsinki University of Technology (TKK)

Autumn 2007

Outline

k-means Clustering

Lloyd's algorithm

LLOYDS($\mathcal{X}$,k) {Input: $\mathcal{X}$, data set; k, number of clusters. Output: $\{\mathbf{m}_i\}_{i=1}^k$, cluster prototypes.}
Initialize $\mathbf{m}_i$, $i = 1, \ldots, k$, appropriately for example, in random.
repeat
  for all $t \in \{1, \ldots, N\}$ **do** {E step}

$$b_i^t \leftarrow \begin{cases} 1 & , \quad i = \arg\min_i \left|\left|\mathbf{x}^t - \mathbf{m}_i\right|\right| \\ 0 & , \quad \text{otherwise} \end{cases}$$

  end for
  for all $i \in \{1, \ldots, k\}$ **do** {M step}

$$\mathbf{m}_i \leftarrow \frac{\sum_t b_i^t \mathbf{x}^t}{\sum_t b_i^t}$$

  end for
until the error $\mathcal{E}(\{\mathbf{m}_i\}_{i=1}^k \mid \mathcal{X})$ does not change
return $\{\mathbf{m}_i\}_{i=1}^k$

k-means Clustering

Lloyd's algorithm

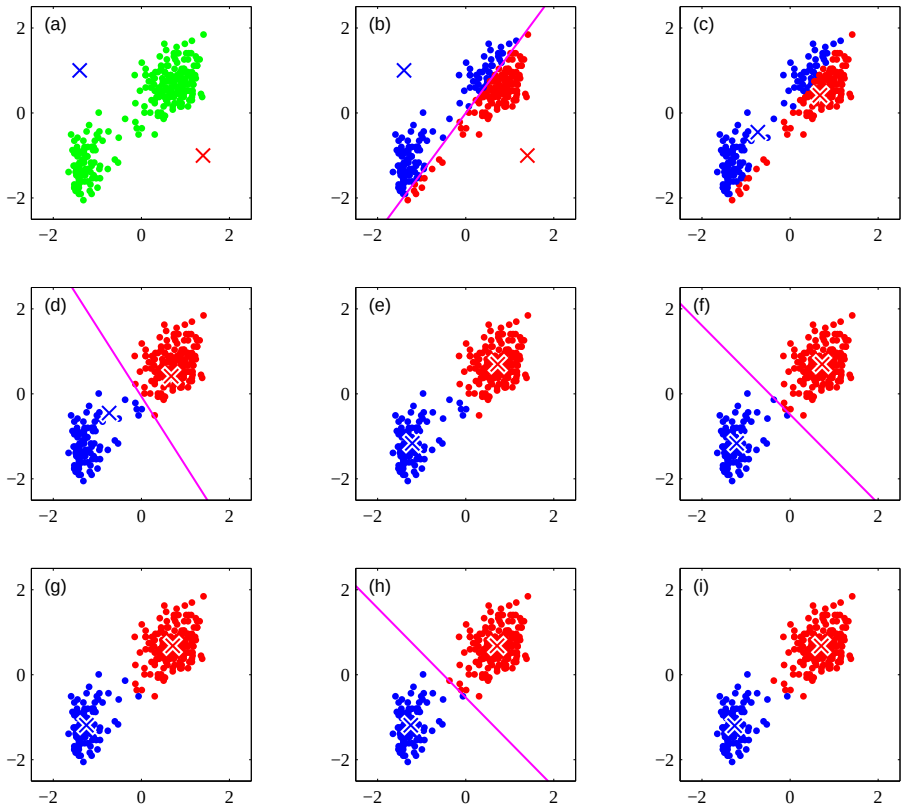


Figure 9.1 of Bishop (2006)

k-means Clustering
Lloyd's algorithm

Observations:

- Iteration cannot increase the error $\mathcal{E}(\{\mathbf{m}_i\}_{i=1}^k \mid \mathcal{X})$.
- There are finite number, k^N, of possible clusterings.
- It follows that the algorithm always stops after a finite time. (It can take no more than k^N steps.)
- Usually k-means is however relatively fast. "In practice the number of iterations is generally much less than the number of points." (Duda & Hart & Stork, 2000)
- Worst-case running time with really bad data and really bad initialization is however $2^{\Omega(\sqrt{N})}$ — luckily this usually does not happen in real life (David A, Vassilivitskii S (2006) How slow is the k-means method? In Proc 22nd SCG.)

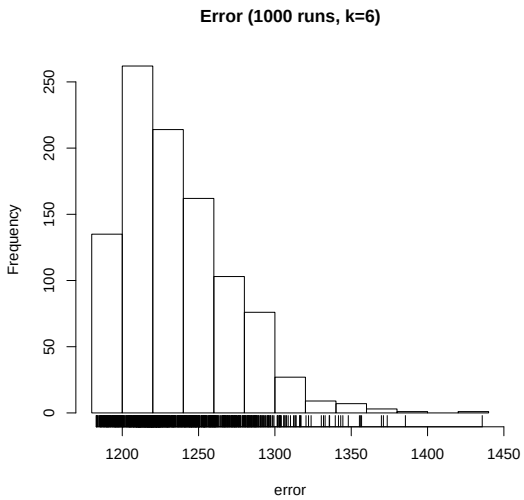
k-means Clustering
Lloyd's algorithm

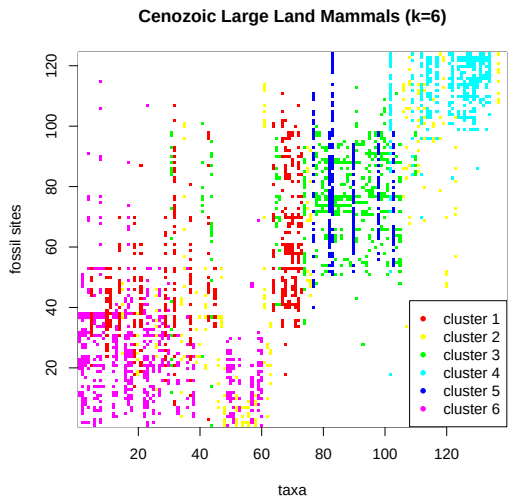
Observations:

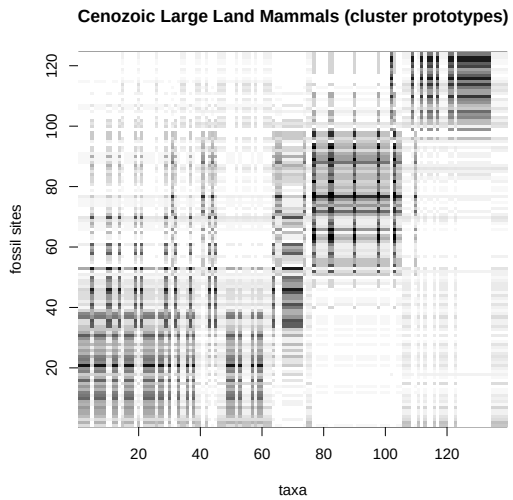
- The result can in the worst case be really bad.
- Example:
  - Four data vectors ($N = 4$) from $\mathbb{R}^d$ in $\mathcal{X}$: $\mathbf{x}^1 = (0, 0, \ldots, 0)^T$, $\mathbf{x}^2 = (1, 0, \ldots, 0)^T$, $\mathbf{x}^3 = (0, 1, \ldots, 1)^T$ and $\mathbf{x}^4 = (1, 1, \ldots, 1)^T$.
  - Optimal clustering into two ($k = 2$) is given by the prototype vectors $\mathbf{m}_1 = (0.5, 0, \ldots, 0)^T$ and $\mathbf{m}_2 = (0.5, 1, \ldots, 1)^T$, error being $\mathcal{E}(\{\mathbf{m}_i\}_{i=1}^k \mid \mathcal{X}) = 1$.
  - Lloyd's algorithm can however converge also to $\mathbf{m}_1 = (0, 0.5, \ldots, 0.5)^T$ and $\mathbf{m}_2 = (1, 0.5, \ldots, 0.5)^T$, error being $\mathcal{E}(\{\mathbf{m}_i\}_{i=1}^k \mid \mathcal{X}) = d - 1$. (Check that iteration stops here!)

k-means Clustering
Lloyd's algorithm

- Example: cluster taxa into $k = 6$ clusters 1000 times with Lloyd's algorithm.
- The error $\mathcal{E}(\{\mathbf{m}_i\}_{i=1}^k \mid \mathcal{X})$ is different for different runs!
- You should try several random initializations, and choose the solution with smallest error.
- For a cool initialization see Arthur D, Vassilivitskii S (2006) k-means++: The Advantages of Careful Seeding.







Outline

Greedy algorithm

- Task: solve $\arg\min_\theta \mathcal{E}(\theta \mid \mathcal{X})$.
- $0 \leq \mathcal{E}(\theta \mid \mathcal{X}) < \infty$
- Assume that the cost/error $\mathcal{E}(\theta \mid \mathcal{X})$ can be evaluated in polynomial time $O(N^k)$, given an instance of parameters θ and a data set $\mathcal{X}$, where N is the size of the data set and k is some constant.
- Often, no polynomial time algorithm to minimize the cost is known.
- Assume that for each instance parameter values θ there exists a candidate set $C(\theta)$ such that $\theta \in C(\theta)$.
- Assume $\arg\min_{\theta' \in C(\theta)} \mathcal{E}(\theta' \mid \mathcal{X})$ can be solved in polynomial time.

Greedy algorithm

GREEDY($\mathcal{E}$,C,ϵ,$\mathcal{X}$) {Input: $\mathcal{E}$, cost function; C, candidate set; $\epsilon \geq 0$, convergence cutoff; $\mathcal{X}$, data set. Output: Instance of parameter values θ.}
Initialize θ appropriately, for example, in random.
repeat

$$\theta \leftarrow \arg \min_{\theta' \in C(\theta)} \mathcal{E}(\theta' \mid \mathcal{X})$$

until the change in $\mathcal{E}(\theta \mid \mathcal{X})$ is no more than ϵ
return θ

Greedy algorithm

- Examples of greedy algorithms:
  - Forward and backward selection.
  - Lloyd's algorithm.
  - Optimizing a cost function using gradient descent and line search.

Greedy algorithm
Observations

- Each step (except the last) reduces the cost by more than ϵ.
- Each step can be done in polynomial time.
- The algorithm stops after a finite number of steps (at least if $\epsilon > 0$).
- Difficult parts:
  - What is a good initialization?
  - What is a good candidate set $C(\theta)$?
- θ is a global optimum if $\theta = \arg\min_\theta \mathcal{E}(\theta \mid \mathcal{X})$.
- θ is a local optimum if $\theta = \arg\min_{\theta' \in C(\theta)} \mathcal{E}(\theta' \mid \mathcal{X})$.
- Algorithm always finds a local optimum, but not necessarily a global optimum. (Interesting sidenote: greedoid.)

Greedy algorithm
Approximation ratio

- Denote $\mathcal{E}^* = \min_\theta \mathcal{E}(\theta \mid \mathcal{X})$, $\theta_{ALG} = \mathrm{GREEDY}(\mathcal{E}, C, \epsilon, \mathcal{X})$ and $\mathcal{E}_{ALG} = \mathcal{E}(\theta_{ALG} \mid \mathcal{X})$
- $1 \leq \alpha < \infty$ is an approximation ratio if $\mathcal{E}_{ALG} \leq \alpha \mathcal{E}^*$ is always satisfied for all $\mathcal{X}$.
- $1 \leq \alpha < \infty$ is an expected approximation ratio if $E[\mathcal{E}_{ALG}] \leq \alpha \mathcal{E}^*$ is always satisfied for all $\mathcal{X}$ (expectation is over instances of the algorithm).
- Observation: if approximation ratio exists, then the algorithm always finds the zero cost solution if such a solution exists for a given data set.
- Sometimes the approximation ratio can be proven; often one can only run algorithm several times and observe the distribution of costs.
- For kmeans with approximation ratio $\alpha = O(\log k)$ and references see Arthur D, Vassilivitskii S (2006) k-means++: The Advantages of Careful Seeding.

Greedy algorithm
Running times

- We can usually easily say that the running time of one step is polynomial.
- Often, the number of steps the algorithm takes is also polynomial, hence the algorithm is often polynomial (at least in practice).
- Proving the number of steps required until convergence is often quite difficult, however. Again, the easiest is to run algorithm several times and observe the distribution of the number of steps.

Greedy algorithm

Questions to ask about a greedy algorithm

- Does the definition of the cost function make sense in your application? Should you use some other cost, for example, some utility?
- There may be several solutions with small cost. Do these solutions have similar parameters, for example, prototype vectors (interpretation of the results)?
- How efficient is the optimization step involving $C(\theta)$? Could you find better $C(\theta)$?
- If there exists a zero-cost solution, does your algorithm find it?
- Is there an approximation ratio?
- Can you say anything about number of steps required?
- What is the empirical distribution of the error $\mathcal{E}_{ALG}$ and the number of steps taken, in your typical application?

Outline

EM Algorithm

- Expectation-Maximization algorithm (EM): greedy algorithm that finds soft cluster assignments
- Probabilistic interpretation, that is, we are maximizing a likelihood.

EM Algorithm

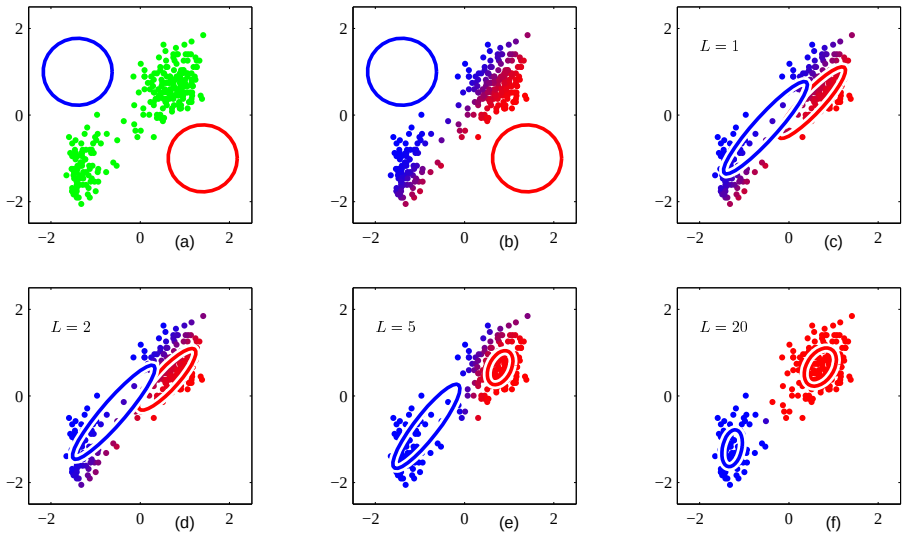
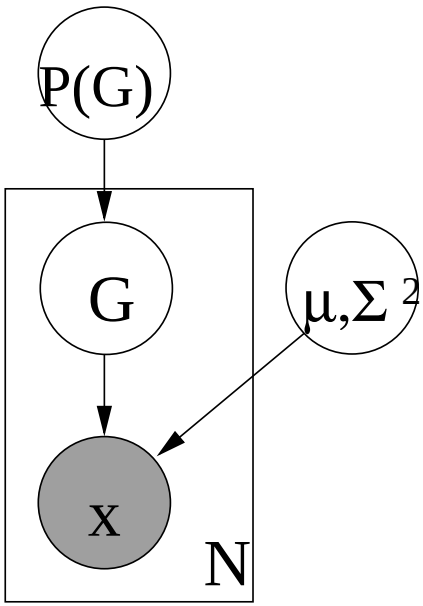


Figure 9.8 of Bishop (2006)

- EM algorithm is like k-means, except cluster assignments are "soft": each data point is a member of a given cluster with certain probability.
- $b_i^t \in \{0, 1\} \longrightarrow h_i^t \in [0, 1]$.

EM Algorithm

- Find maximum likelihood solution of the mixture model $\mathcal{L} = \log \prod_{t=1}^{N} p(\mathbf{x}^t \mid \theta)$, where the parameters θ are μ_i, Σ_i and $\pi_i = P(G_i)$.
- Maximum likelihood solution is found by the EM algorithm (which is essentially generalization of the Lloyd's algorithm to soft cluster memberships)
- Idea: iteratively find the membership weights of each data vector in clusters, and the parameter values. Continue until convergence.
- End result is intuitive.



EM Algorithm

Example: soft Gaussian mixture, fixed shared diagonal covariance matrix $\Sigma_i = s^2\mathbf{1}$, $P(G_i) = \pi_i = 1/k$.

EM($\mathcal{X}$,k) {Input: $\mathcal{X}$, data set; k, number of mixture components. Output: $\{\mathbf{m}_i\}_{i=1}^k$, mixture components.}
Initialize $\mathbf{m}_i$, $i = 1, \ldots, k$, for example using some kmeans algorithm.
repeat
  for all $t \in \{1, \ldots, N\}$ **do** {E step}

$$h_i^t \leftarrow \frac{\exp\left[-\frac{1}{2s^2}\left|\left|\mathbf{x}^t - \mathbf{m}_i\right|\right|^2\right]}{\sum_j \exp\left[-\frac{1}{2s^2}\left|\left|\mathbf{x}^t - \mathbf{m}_j\right|\right|^2\right]}$$

  end for
  for all $i \in \{1, \ldots, k\}$ **do** {M step}

$$\mathbf{m}_i \leftarrow \frac{\sum_t h_i^t \mathbf{x}^t}{\sum_t h_i^t}$$

  end for
until convergence
return $\{\mathbf{m}_i\}_{i=1}^k$

EM Algorithm

- For derivation, see Alpaydin (2004), section 7.4 (pages 139–144); for an alternative derivation, see Bishop (2006), section 9.4 (pages 450–455). A sketch follows.
- Task: find an ML solution of a likelihood function given by $p(\mathbf{X} \mid \theta) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z} \mid \theta)$.

$$\begin{aligned} \sum_t \log p(\mathbf{x}^t \mid \theta) &\geq \sum_t \log p(\mathbf{x}^t \mid \theta) - \sum_t KL(h_i^t \mid\mid p(\mathbf{z}^t \mid \mathbf{x}^t, \theta)) \\ &= \sum_t \sum_i h_i^t \log p(\mathbf{x}^t, \mathbf{z}^t \mid \theta) + \sum_t H(h_i^t), \end{aligned}$$

where we have used the Kullback-Leibler (KL) divergence $KL(q(i) \mid\mid p(i)) = \sum_i q(i) \log (q(i)/p(i))$. KL divergence is always non-negative and it vanishes only when the distributions q and p are equal. The entropy is given by $H(q(i)) = -\sum_i q(i) \log q(i)$.

EM Algorithm

- Expectation step (E Step): find h_i^t by minimizing the KL divergence.
- Maximization step (M Step): find θ by maximizing the expectation.

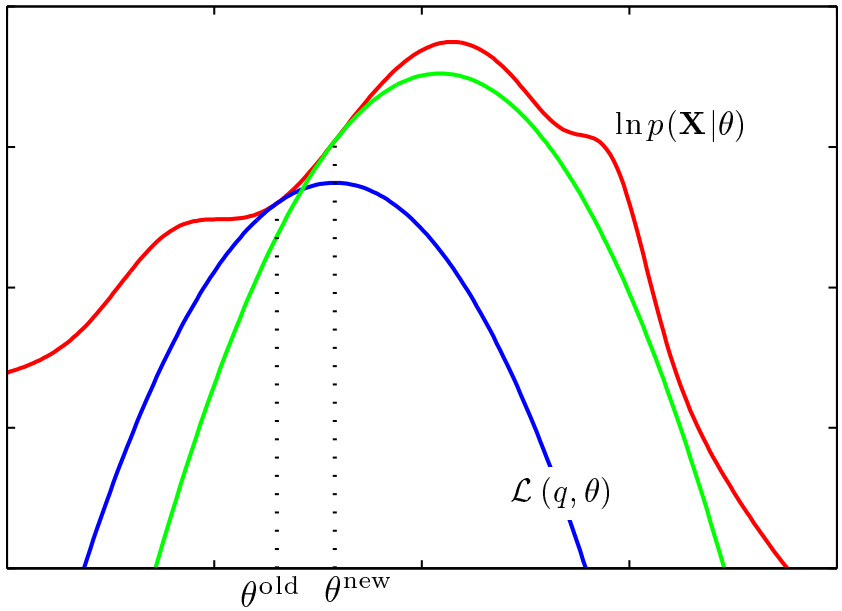
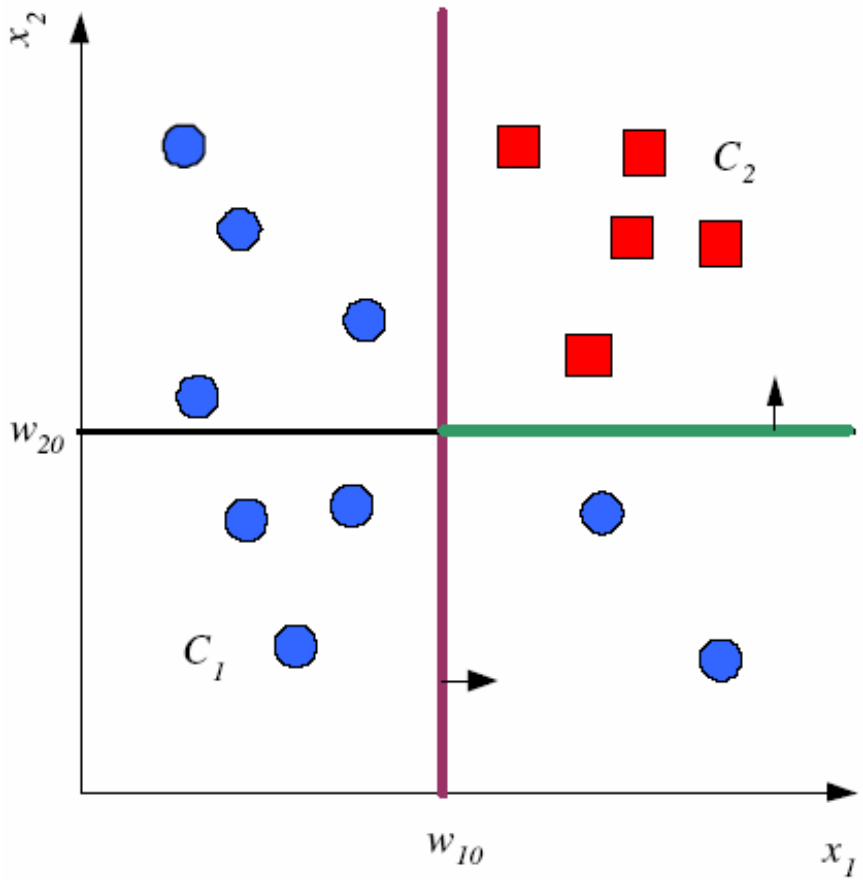
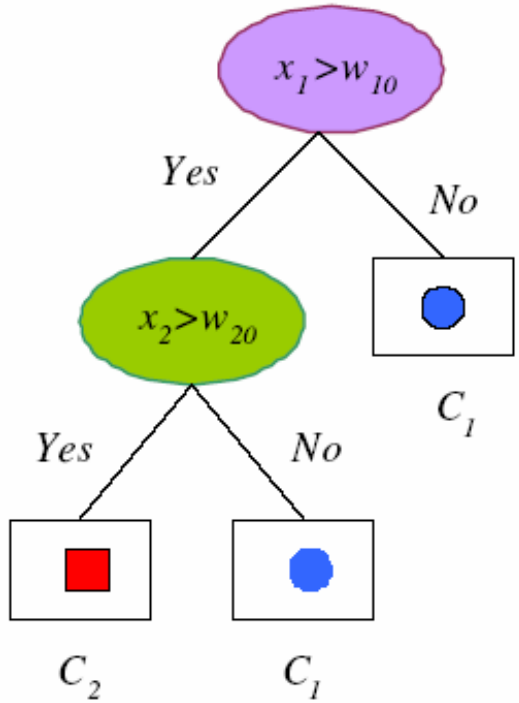


Figure 9.14 of Bishop (2006)

Outline

Decision Trees





Decision Trees

- Each internal node tests an attribute.
- Each branch corresponds to set of attribute values.
- Each leaf node assigns a classification (classification tree) or a real number (regression tree).
- The tree is usually learned using a greedy algorithm built around *ID*3, such as *C*4.5. (The problem of finding optimal tree is generally NP-hard.)
- Advantages of trees:
  - Learning and classification is fast.
  - Trees are accurate in many domains.
  - Trees are easy to interpret as sets of decision rules.
- Often, trees should be used as a benchmark before more complicated algorithms are attempted.
- For alternative discussion, see Mitchell (1997), Ch 3.

Outline

Example Data from Mitchell (1997)

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example: Final Decision Tree

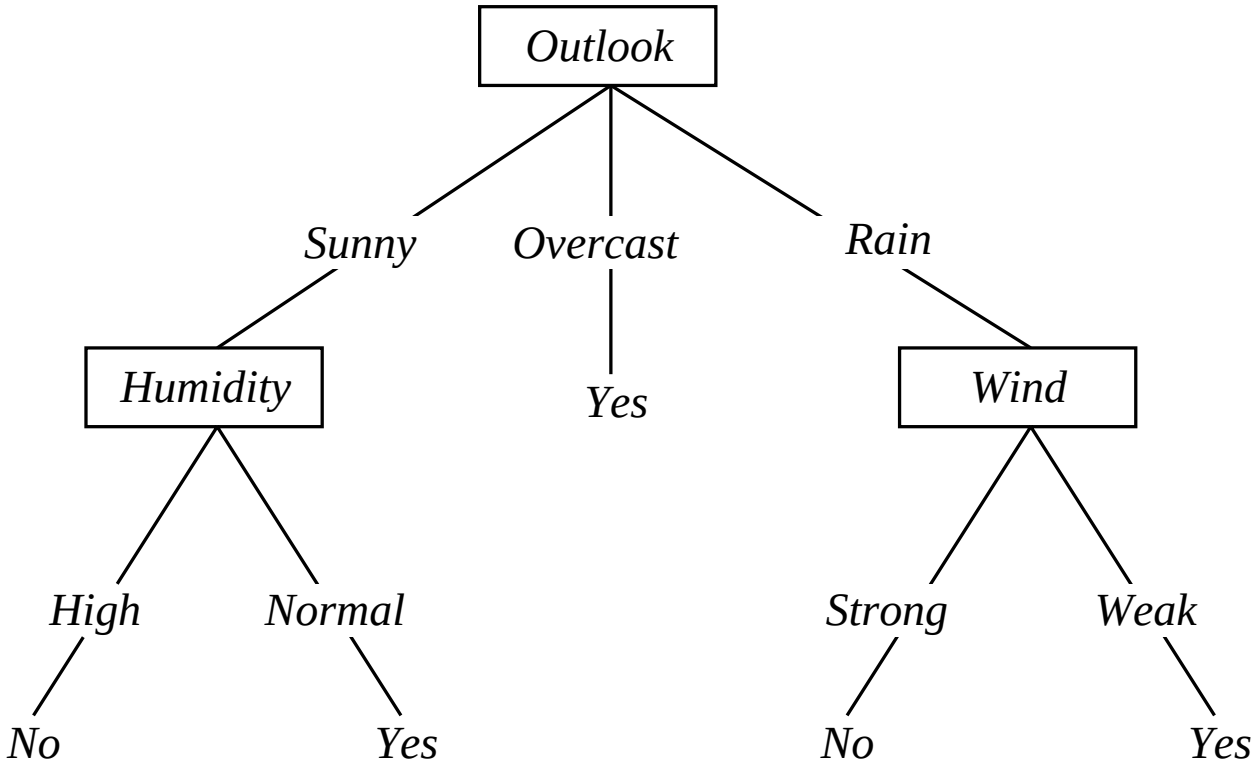


Figure 3.1 of Mitchell (1997).

ID3 algorithm for discrete attributes

ID3($\mathcal{X}$) {Input: $\mathcal{X} = \{(r^t, \mathbf{x}^t)\}_{t=1}^N$, data set with binary attributes $r^t \in \{-1, +1\}$ and a vector of discrete variables $\mathbf{x}^t$. Output: T, classification tree.}
Create *root* node for T
If all items in $\mathcal{X}$ are positive (negative), return a single-node tree with label "+" ("-")
Let A be attribute that "best" classifies the examples
for all values v of A **do**
  Let $\mathcal{X}_v$ be subset of $\mathcal{X}$ that have value v for A
  if $\mathcal{X}_v$ is empty **then**
    Below the root of T, add a leaf node with most common label in $\mathcal{X}$
  else
    Below the root of T, add subtree ID3($\mathcal{X}_v$)
  end if
end for
return T

Entropy

- $\mathcal{X}$ is a sample of training examples.
- p_+ is the proportion of positive and $p_- = 1 - p_+$ is the proportion of negative samples in $\mathcal{X}$.
- Entropy measures impurity of $\mathcal{X}$.

$$\text{Entropy}(\mathcal{X}) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

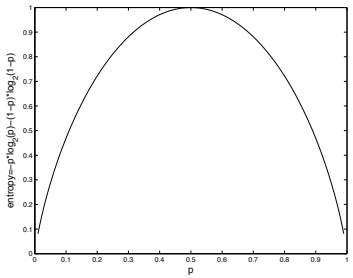


Figure 9.2: Entropy function for a two-class problem. *From: E. Alpaydın. 2004.* Introduction to Machine Learning. ©*The MIT Press.*

Entropy

- $\text{Entropy}(\mathcal{X})$ is the expected number of bits needed to encode class ($+1$ or -1) of randomly drawn member of $\mathcal{X}$ (under the optimal, shortest-length code).
- Information theory: the optimal (shortest expected coding length) code for an event with probability p is $-\log_2 p$ bits.
- Therefore, expected number of bits to encode $+1$ or -1 of a random member of $\mathcal{X}$ is

$$p_+ \left(-\log_2 p_+\right) + p_- \left(-\log_2 p_-\right).$$

$$\text{Entropy}(\mathcal{X}) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

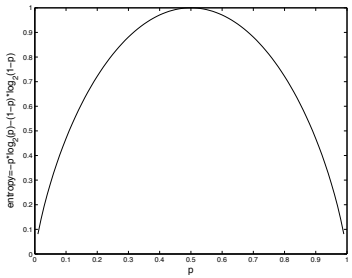


Figure 9.2: Entropy function for a two-class problem. *From: E. Alpaydın. 2004.* Introduction to Machine Learning. ©*The MIT Press.*

Information Gain

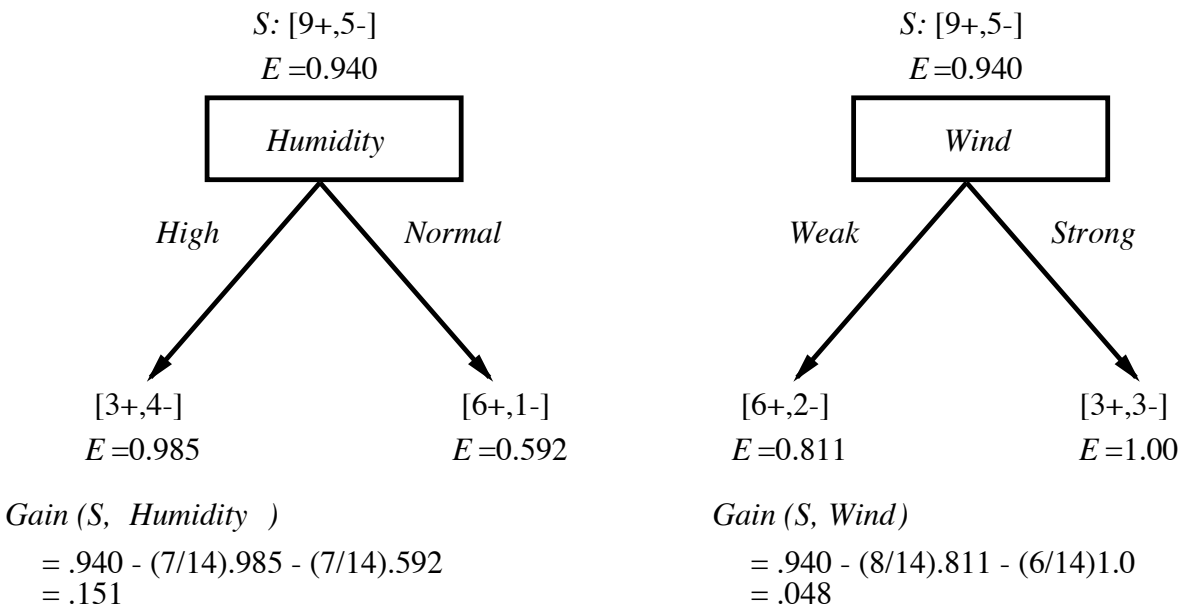
- $\mathrm{Gain}(\mathcal{X}, A)$ is the expected reduction in entropy due to sorting on A.

$$\mathrm{Gain}(\mathcal{X}, A) = \mathrm{Entropy}(\mathcal{X}) - \sum_{v \in \mathrm{values}(A)} \frac{|\mathcal{X}_v|}{|\mathcal{X}|} \mathrm{Entropy}(\mathcal{X}_v).$$

- For ID3: attribute A that has the highest gain classifies the examples $\mathcal{X}$ "best".

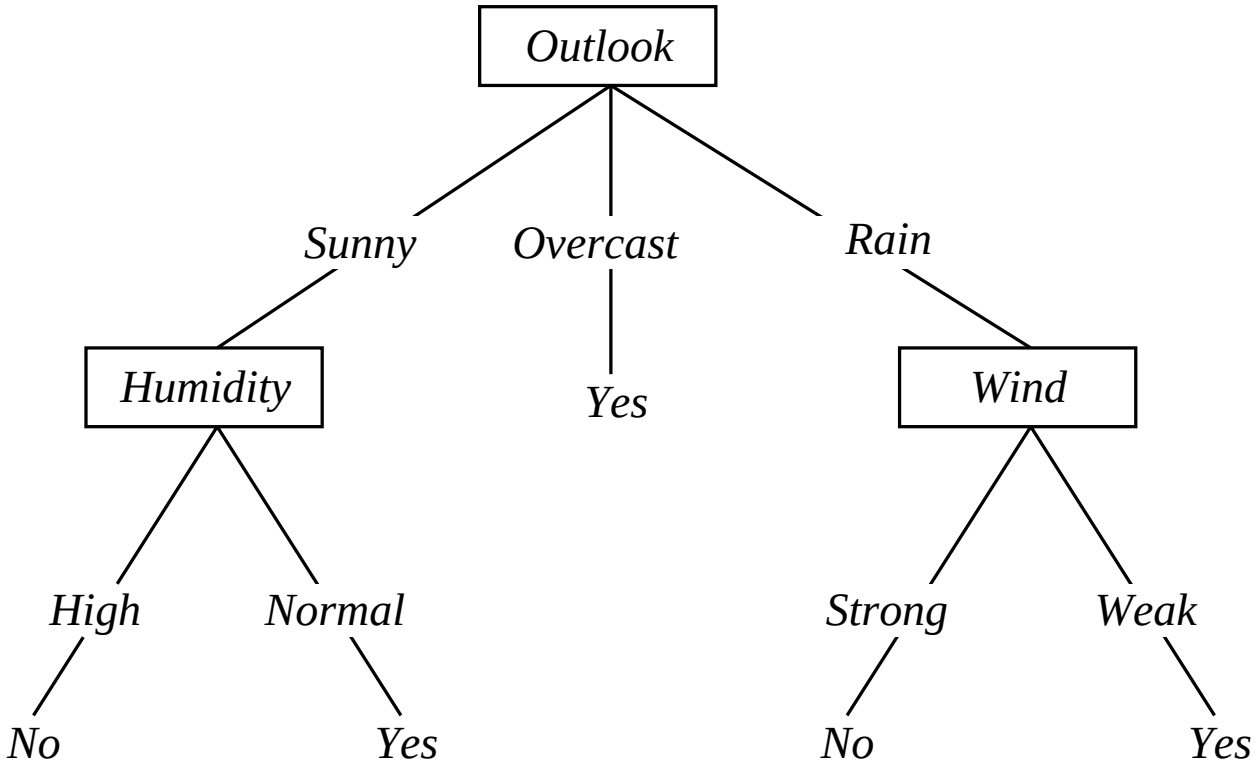
Selecting the Next Attribute

Which attribute is the best classifier?



Humidity provides greater information gain than *Wind*, relative to the target classification. E stands for entropy and S for collection of examples. Figure 3.3 of Mitchell (1997).

Example: Final Decision Tree



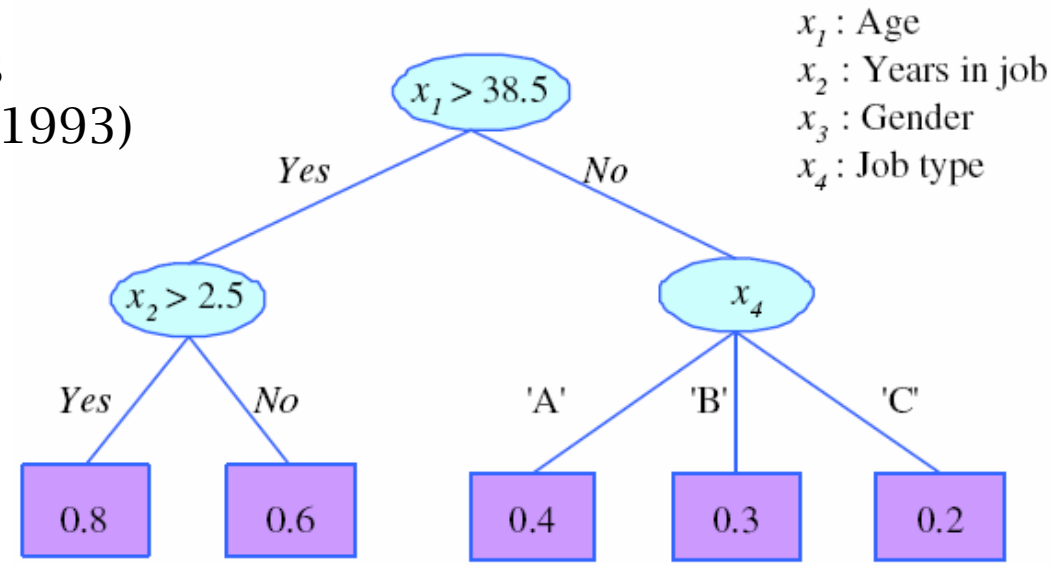
The final decision tree. Figure 3.1 of Mitchell (1997).

Variations of ID3

- Alternative impurity measures:
  - Entropy: $-p_+ \log_2 p_+ - p_- \log_2 p_-$.
  - Gini index: $2p_+p_-$.
  - Misclassification error: $1 - \max(p_+, p_-)$.
  - All vanish for $p_+ \in \{0, 1\}$ and have a maximum at $p_+ = p_- = 1/2$.
- Continuous or ordered variables: sort x_A^t for some attribute A and find the best split $x_A \leq w$ vs. $x_A > w$.

Rule Extraction from Trees

C4.5Rules
(Quinlan, 1993)



R1: IF (age>38.5) AND (years-in-job>2.5) THEN y =0.8
R2: IF (age>38.5) AND (years-in-job≤2.5) THEN y =0.6
R3: IF (age≤38.5) AND (job-type='A') THEN y =0.4
R4: IF (age≤38.5) AND (job-type='B') THEN y =0.3
R5: IF (age≤38.5) AND (job-type='C') THEN y =0.2

Observations of ID3

- Inductive bias:
  - Preference on short trees.
  - Preference on trees with high information gain near root.
- Vanilla ID3 classifies the training data perfectly.
- Hence, in presence of noise, vanilla ID3 overfits.

Pruning

- How to avoid overfitting?
  - Prepruning: stop growing when data split is not statistically significant. For example: stop tree construction when node is smaller than a given limit, or impurity of a node is below a given limit θ_I. (*faster*)
  - Postpruning: grow the whole tree, then prune subtrees which overfit on the pruning (validation) set. (*more accurate*)

Pruning
Postpruning

- Split data into training and pruning (validation) sets.
- Do until further pruning is harmful:
  1. Evaluate impact on *pruning* set of pruning each possible node (plus those below it).
  2. Greedily remove the one that most improves the *pruning* set accuracy.
- Produces smallest version of most accurate subtree.
- Alternative: rule postpruning (commonly used, for example, $C4.5$).

Outline

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 is the most important factor.

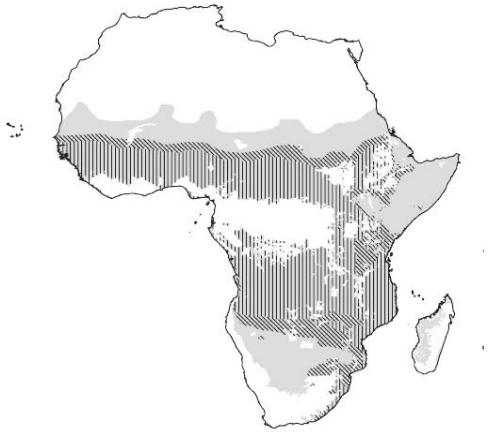


Figure 4 | The distributions of MAP-determined ('stable') and disturbance-determined ('unstable') savannas in Africa. Grey areas represent the existing distribution of savannas in Africa according to ref. 30. Vertically hatched areas show the unstable savannas (>784 mm MAP); cross-hatched areas show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

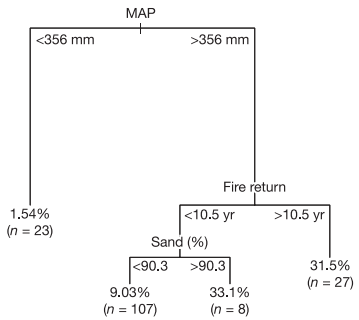


Figure 3 | Regression tree showing generalized relationships between woody cover and MAP, fire-return interval and percentage of sand. The tree is pruned to four terminal nodes and is based on 161 sites for which all data were available. No consistent herbivore effects were detected. Branches are labelled 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.

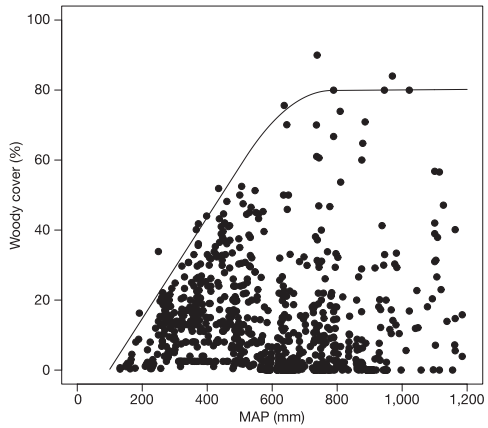


Figure 1 | Change in woody cover of African savannas as a function of MAP. Maximum tree cover is represented by using a 99th quantile piec wise linear regression. The regression analysis identifies the breakpoint (t rainfall at which maximum tree cover is attained) in the interval 650 ± 134 mm MAP (between 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 M et al. (2005) Determinants of woody cover in African savannas. Nature 438: 846–849.

Regression Trees

- Error at node m:

$$b_m(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases}$$

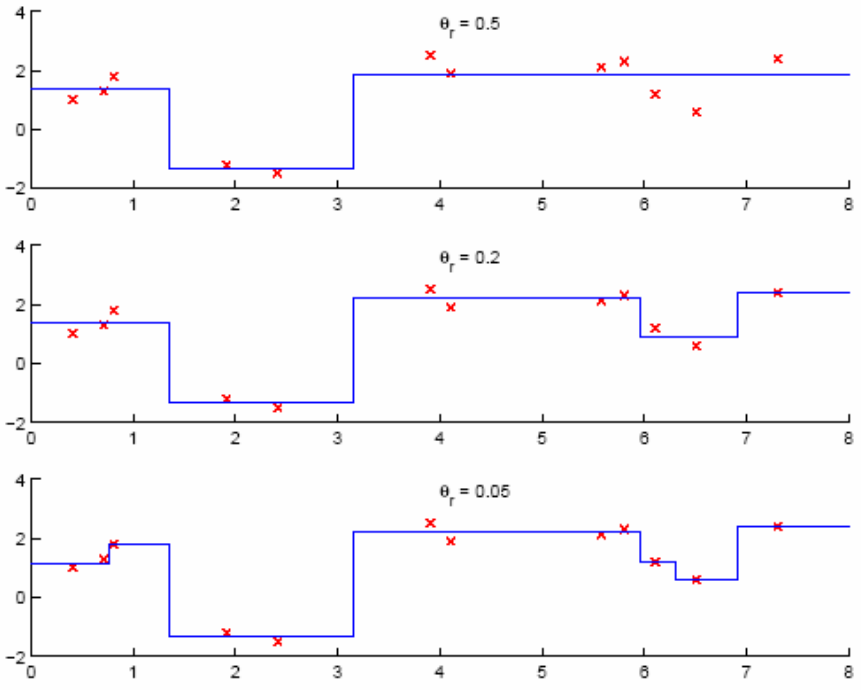
$$\mathcal{E}_m = \frac{1}{N_m} \sum_t \left(r^t - g_m\right)^2 b_m(\mathbf{x}^t) \quad , \quad g_m = \frac{\sum_t b_m(\mathbf{x}^t) r^t}{\sum_t b_m(\mathbf{x}^t)}.$$

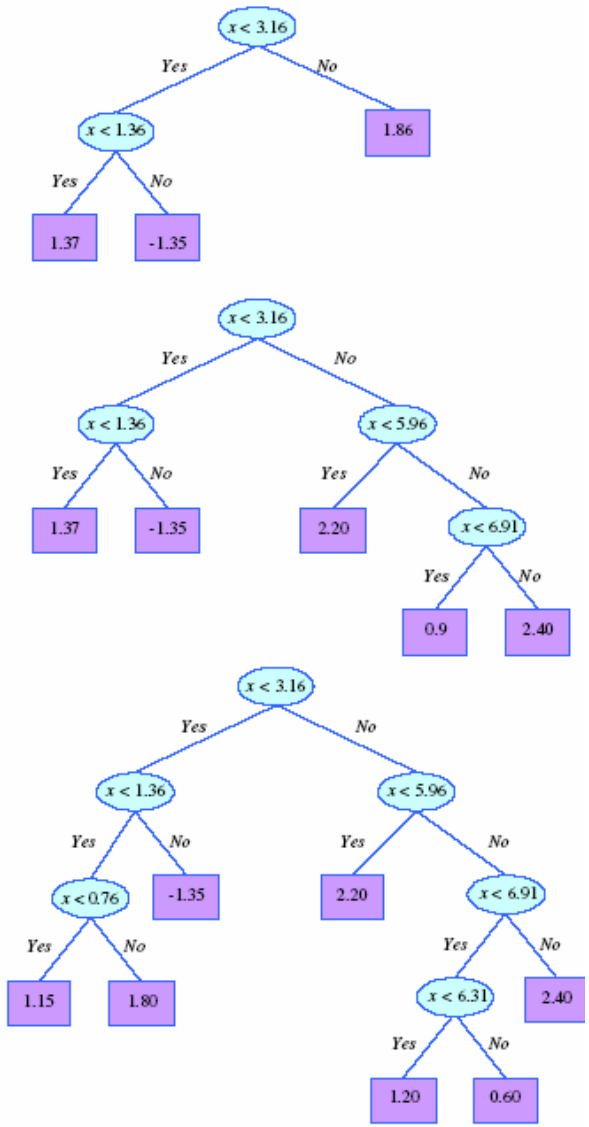
- After splitting:

$$b_{mj}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \text{ reaches node } m \text{ and branch } j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{E}_m = \frac{1}{N_m} \sum_j \sum_t \left(r^t - g_{mj}\right)^2 b_{mj}(\mathbf{x}^t) \quad , \quad g_{mj} = \frac{\sum_t b_{mj}(\mathbf{x}^t) r^t}{\sum_t b_{mj}(\mathbf{x}^t)}.$$

Model Selection in Trees:





Implementations

- There are many implementations, with sophisticated pruning methods.

```
> library(rpart)
> rpart(Hipparion ~ .,DD[,taxa])
n= 124

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 124 32 0 (0.74193548 0.25806452)
   2) Amphimachairodus=0 108 19 0 (0.82407407 0.17592593)
     4) Choerolophodon=0 96 13 0 (0.86458333 0.13541667)
       8) Ursus=0 76  6 0 (0.92105263 0.07894737) *
       9) Ursus=1 20  7 0 (0.65000000 0.35000000)
        18) Cervus=1 13  2 0 (0.84615385 0.15384615) *
        19) Cervus=0 7  2 1 (0.28571429 0.71428571) *
     5) Choerolophodon=1 12  6 0 (0.50000000 0.50000000) *
   3) Amphimachairodus=1 16  3 1 (0.18750000 0.81250000) *
```