Chapter ML:III

III. Decision Trees

Decision Trees Basics

Specification of Classification Problems [ML Introduction]

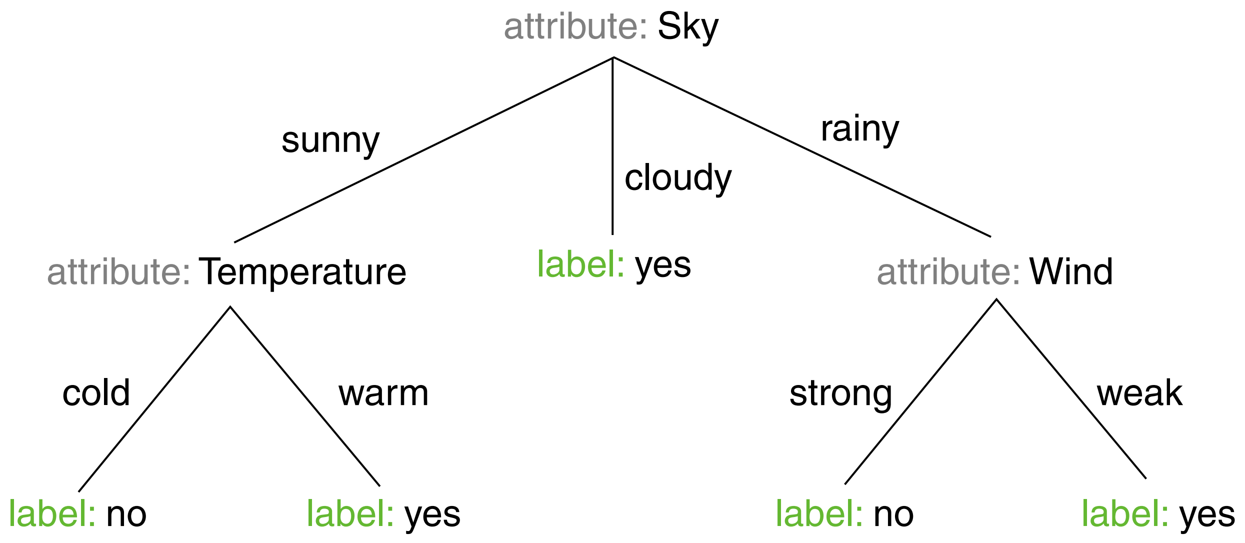
Characterization of the model (model world):

- X is a set of feature vectors, also called feature space.
- C is a set of classes.
- $c : X \to C$ is the ideal classifier for X.
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$ is a set of examples.

Decision Trees Basics

Decision Tree for the Concept "EnjoySport"

Example	Sky	Temperature	Humidity	Wind	Water	Forecast	EnjoySport
1	sunny	warm	normal	strong	warm	same	yes
2	sunny	warm	high	strong	warm	same	yes
3	rainy	cold	high	strong	warm	change	no
4	sunny	warm	high	strong	cool	change	yes



Partitioning of X at the root node:

$$X = \{\mathbf{x} \in X : \mathbf{x}|_{\text{Sky}} = \text{sunny}\} \cup \{\mathbf{x} \in X : \mathbf{x}|_{\text{Sky}} = \text{cloudy}\} \cup \{\mathbf{x} \in X : \mathbf{x}|_{\text{Sky}} = \text{rainy}\}$$

Decision Trees Basics

Definition 1 (Splitting)

Let X be feature space and let D be a set of examples. A splitting of X is a partitioning of X into mutually exclusive subsets $X_1, \ldots, X_s$. I.e., $X = X_1 \cup \ldots \cup X_s$ with $X_j \neq \emptyset$ and $X_j \cap X_{j'} = \emptyset$, where $j, j' \in \{1, \ldots, s\}, j \neq j'$.

A splitting $X_1, \ldots, X_s$ of X induces a splitting $D_1, \ldots, D_s$ of D, where D_j, $j = 1, \ldots, s$, is defined as $\{(\mathbf{x}, c(\mathbf{x})) \in D \mid \mathbf{x} \in X_j\}$.

Decision Trees Basics

Definition 1 (Splitting)

Let X be feature space and let D be a set of examples. A splitting of X is a partitioning of X into mutually exclusive subsets $X_1, \ldots, X_s$. I.e., $X = X_1 \cup \ldots \cup X_s$ with $X_j \neq \emptyset$ and $X_j \cap X_{j'} = \emptyset$, where $j, j' \in \{1, \ldots, s\}, j \neq j'$.

A splitting $X_1, \ldots, X_s$ of X induces a splitting $D_1, \ldots, D_s$ of D, where D_j, $j = 1, \ldots, s$, is defined as $\{(\mathbf{x}, c(\mathbf{x})) \in D \mid \mathbf{x} \in X_j\}$.

A splitting depends on the measurement scale of a feature:

1. Splitting induced by a (nominal) feature A with finite domain:

2. Binary splitting induced by a (nominal) feature A:

3. Binary splitting induced by an ordinal feature A:

Decision Trees Basics

Definition 1 (Splitting)

Let X be feature space and let D be a set of examples. A splitting of X is a partitioning of X into mutually exclusive subsets $X_1, \ldots, X_s$. I.e., $X = X_1 \cup \ldots \cup X_s$ with $X_j \neq \emptyset$ and $X_j \cap X_{j'} = \emptyset$, where $j, j' \in \{1, \ldots, s\}, j \neq j'$.

A splitting $X_1, \ldots, X_s$ of X induces a splitting $D_1, \ldots, D_s$ of D, where D_j, $j = 1, \ldots, s$, is defined as $\{(\mathbf{x}, c(\mathbf{x})) \in D \mid \mathbf{x} \in X_j\}$.

A splitting depends on the measurement scale of a feature:

1. Splitting induced by a (nominal) feature A with finite domain:

   $A = \{a_1, \ldots, a_m\}: \quad X = \{\mathbf{x} \in X : \mathbf{x}|_A = a_1\} \cup \ldots \cup \{\mathbf{x} \in X : \mathbf{x}|_A = a_m\}$

2. Binary splitting induced by a (nominal) feature A:

   $A' \subset A: \quad X = \{\mathbf{x} \in X : \mathbf{x}|_A \in A'\} \cup \{\mathbf{x} \in X : \mathbf{x}|_A \notin A'\}$

3. Binary splitting induced by an ordinal feature A:

   $v \in \mathit{dom}(A): \quad X = \{\mathbf{x} \in X : \mathbf{x}|_A \succeq v\} \cup \{\mathbf{x} \in X : \mathbf{x}|_A \prec v\}$

Remarks:

- The syntax $\mathbf{x}|_A$ denotes the projection operator, which returns that vector component (dimension) of $\mathbf{x} = x_1, \ldots, x_p$ that is associated with A. Without loss of generality this projection can be presumed as being unique.
- A splitting of X into two disjoint, non-empty subsets is called a binary splitting.
- We consider only splittings of X that are induced by a splitting of a single feature A of X.

Decision Trees Basics

Definition 2 (Decision Tree)

Let X be feature space and let C be a set of classes. A decision tree T for X and C is finite tree with a distinguished root node. A non-leaf node t of T has assigned (1) a set $X(t) \subseteq X$, (2) a splitting of $X(t)$, and (3) a one-to-one mapping of the subsets of the splitting to its successors.

$X(t) = X$ iff t is root node. A leaf node of T has assigned a class from C.

Decision Trees Basics

Definition 2 (Decision Tree)

Let X be feature space and let C be a set of classes. A decision tree T for X and C is finite tree with a distinguished root node. A non-leaf node t of T has assigned (1) a set $X(t) \subseteq X$, (2) a splitting of $X(t)$, and (3) a one-to-one mapping of the subsets of the splitting to its successors.

$X(t) = X$ iff t is root node. A leaf node of T has assigned a class from C.

Classification of some $\mathbf{x} \in X$ given a decision tree T:

1. Find the root node of T.
2. If t is a non-leaf node, find among its successors that node whose subset of the splitting of $X(t)$ contains $\mathbf{x}$. Repeat this step.
3. If t is a leaf node, label $\mathbf{x}$ with the respective class.

➜ The set of possible decision trees forms the hypothesis space H.

Remarks:

- ❑ The classification of an $\mathbf{x} \in X$ determines a unique path from the root node of T to some leaf node of T.
- ❑ At each non-leaf node a particular feature of $\mathbf{x}$ is evaluated in order to find the next node and hence a possible next feature to be analyzed.
- ❑ Each path from the root node to some leaf node corresponds to a conjunction of feature values, which are successively tested. This test can be formulated as a decision rule. Example:

  IF Sky=rainy AND Wind=weak THEN EnjoySport=yes

  If all tests in T are of the kind shown in the example, namely, a comparison with a single feature value, all feature domains must be finite.
- ❑ If in all non-leaf nodes of T only one feature is evaluated at a time, T is called a *monothetic* decision tree. Examples for *polythetic* decision trees are the so-called oblique decision trees.
- ❑ Decision trees became popular in 1986, with the introduction of the ID3 Algorithm by J. R. Quinlan.

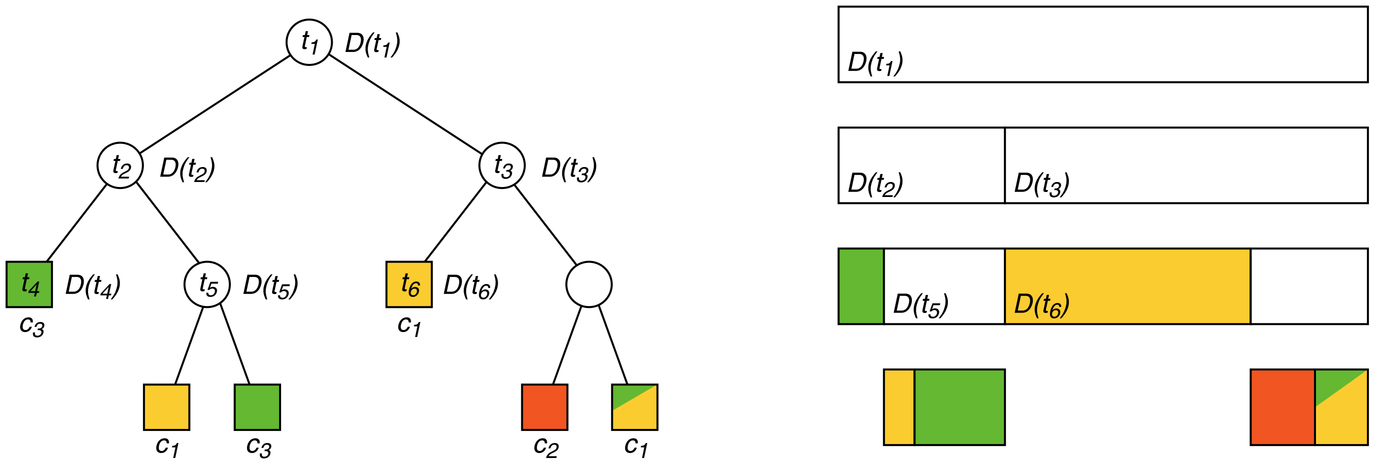
Decision Trees Basics

Notation

Let T be decision tree for X and C, let D be a set of examples, and let t be a node of T. Then we agree on the following notation:

- $X(t)$ denotes the subset of the feature space X that is represented by t.
- $D(t)$ denotes the subset of the example set D that is represented by t, where $D(t) = \{(\mathbf{x}, c(\mathbf{x})) \in D \mid \mathbf{x} \in X(t)\}$

Illustration:



Remarks:

- The set $X(t)$ comprises those members $\mathbf{x}$ of X that are filtered by a path from the root node of T to the node t.
- *leaves*(T) denotes the set of all leaf nodes of T.
- A single node t of a decision tree T, and hence T itself, encode a piecewise constant function. This way, t as well as T can form complex non-linear classifiers. The functions encoded by t and T differ in the number of evaluated features of $\mathbf{x}$, which is one for t and the tree height for T.
- In the following we will use the symbols "t" and "T" to denote also the classifiers that are encoded by a node t and a tree T respectively:

$$t, T : X \to C \quad \text{(instead of } \ y_t, y_T : X \to C\text{)}$$

Decision Trees Basics

Algorithm Template: Construction

Algorithm:	*DT-construct*	Decision Tree Construction
Input:	D	(Sub)set of examples.
Output:	t	Root node of a decision (sub)tree.

```
DT-construct(D)

  1. t = newNode()
     label(t) = representativeClass(D)

  2. IF impure(D)
     THEN criterion = splitCriterion(D)
     ELSE return(t)

  3. {D_1, ..., D_s} = decompose(D, criterion)

  4. FOREACH D' IN {D_1, ..., D_s} DO

       addSuccessor(t, DT-construct(D'))

     ENDDO

  5. return(t)
```

[Illustration]

Decision Trees Basics

Algorithm Template: Classification

Algorithm:	*DT-classify*	Decision Tree Classification
Input:	$\mathbf{x}$	Feature vector.
	t	Root node of a decision (sub)tree.
Output:	$y(\mathbf{x})$	Class of feature vector $\mathbf{x}$ in the decision (sub)tree below t.

```
DT−classify(x, t)

  1. IF   isLeafNode(t)
     THEN return(label(t))
     ELSE return(DT−classify(x, splitSuccessor(t, x)))
```

Remarks:

- ❑ Since *DT-construct* assigns to each node of a decision tree T a class, each subtree of T as well as each pruned version of a subtree of T represents a valid decision tree on its own.
- ❑ Functions of *DT-construct*:
  - *representativeClass*(D)
    Returns a representative class for the example set D. Note that, due to pruning, each node may become a leaf node.
  - *impure*(D)
    Evaluates the (im)purity of a set D of examples.
  - *splitCriterion*(D)
    Returns a split criterion for $X(t)$ based on the examples in $D(t)$.
  - *decompose*$(D,$ *criterion*$)$
    Returns a splitting of D according to *criterion*.
  - *addSuccessor*(t, t')
    Inserts the successor t' for node t.
- ❑ Functions of *DT-classify*:
  - *isLeafNode*(t)
    Tests whether t is a leaf node.
  - *splitSuccessor*$(t, \mathbf{x})$
    Returns the (unique) successor t' of t for which $\mathbf{x} \in X(t')$ holds.

Decision Trees Basics

When to Use Decision Trees

Problem characteristics that may suggest a decision tree classifier:

- the objects can be described by feature-value combinations.
- the domain and range of the target function are discrete
- hypotheses take the form of disjunctions
- the training set contains noise

Selected application areas:

- medical diagnosis
- fault detection in technical systems
- risk analysis for credit approval
- basic scheduling tasks such as calendar management
- classification of design flaws in software engineering

Decision Trees Basics

On the Construction of Decision Trees

- How to exploit an example set both efficiently and effectively?
- According to what rationale a node should become a leaf node?
- How to assign a class for nodes of impure example sets?
- How to evaluate decision tree performance?

Decision Trees Basics

Performance of Decision Trees

1. Size

2. Classification error

Decision Trees Basics

Performance of Decision Trees

1. Size

   Among those theories that can explain an observation, the most simple one is to be preferred *(Occam's Razor)* :

   Entia non sunt multiplicanda praeter necessitatem or *sine necessitate.*

   [Johannes Clauberg 1622-1665]

   Here: among all decision trees of minimum classification error we choose the one of smallest size.

2. Classification error

   Quantifies the rigor according to which a class label is assigned to $\mathbf{x}$ in leaf node of T, based on the examples in D.

   If all leaf nodes of a decision tree T represent a single example of D, the classification error of T with respect to D is zero.

Decision Trees Basics

Performance of Decision Trees: Size

- Leaf node number
- Tree height
- External path length
- Weighted external path length

Decision Trees Basics

Performance of Decision Trees: Size

- Leaf node number

  The leaf node number corresponds to number of rules that are encoded in a decision tree.

- Tree height

  The tree height corresponds to the maximum rule length and bounds the number of premises to be evaluated to reach class decision.

- External path length

  The external path length totals the lengths of all paths from the root of a tree to its leaf nodes. It corresponds to the space to store all rules that are encoded in a decision tree.

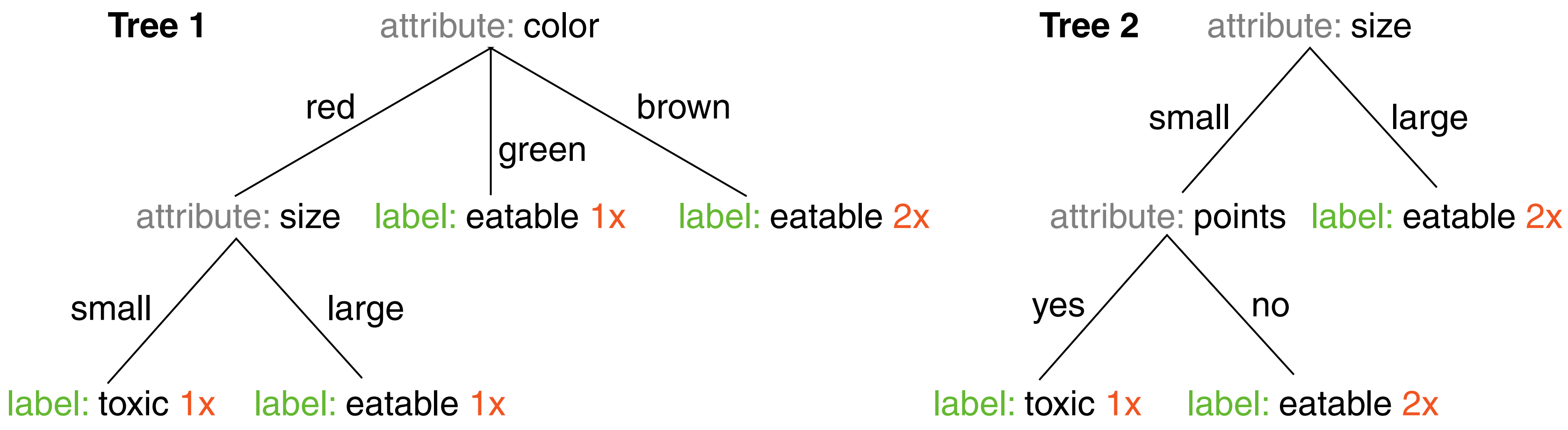
- Weighted external path length

  The weighted external path length is defined as the external path length where each length value is weighted by the number of examples in D that are classified by this path.

Decision Trees Basics

Performance of Decision Trees: Size (continued)

Both trees below correctly classify all examples in D :



Criterion	Tree 1	Tree 2
Leaf node number	4	3
Tree height	2	2
External path length	6	5
Weighted external path length	7	8

Decision Trees Basics

Performance of Decision Trees: Size (continued)

Theorem 1 (External Path Length Bound)

The problem to decide for a set of examples D whether or not a decision tree exists whose external path length is bound by b, is NP complete.

Decision Trees Basics

Performance of Decision Trees: Classification Error

Given a decision tree T, a set of examples D, and a node t of T that represents the example subset $D(t) \subseteq D$. Then, the class that is assigned to t, *label*(t), is defined as follows:

$$\textit{label}(t) = \underset{c \in C}{\operatorname{argmax}} \; \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D(t) : c(\mathbf{x}) = c\}|}{|D(t)|}$$

Decision Trees Basics

Performance of Decision Trees: Classification Error

Given a decision tree T, a set of examples D, and a node t of T that represents the example subset $D(t) \subseteq D$. Then, the class that is assigned to t, *label*(t), is defined as follows:

$$\textit{label}(t) = \underset{c \in C}{\operatorname{argmax}} \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D(t) : c(\mathbf{x}) = c\}|}{|D(t)|}$$

Misclassification rate of node t wrt. $D(t)$:

$$\textit{Err}(t, D(t)) = 1 - \max_{c \in C} \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D(t) : c(\mathbf{x}) = c\}|}{|D(t)|}$$

Misclassification rate of decision tree T wrt. $D(t)$:

$$\textit{Err}(T, D) = \sum_{t \in \textit{leaves}(T)} \textit{Err}(t, D(t)) \cdot \frac{|D(t)|}{|D|}$$

Remarks:

- Observe the difference between max(f) and argmax(f). Both expressions maximize f, whereas the former returns the maximum f-value (the image) while the latter returns the argument (the preimage) for which f becomes maximum:
  - $\max\limits_{c \in C}(f(c)) = \max\{f(c) \mid c \in C\}$
  - $\operatorname*{argmax}\limits_{c \in C}(f(c)) = c^* \;\rightarrow\; f(c^*) = \max\limits_{c \in C}(f(c))$
- The true misclassification rate *Err*$^*(T)$ is based on a probability measure P over $X \times C$ (and not on relative frequencies). For a node t of T this probability becomes minimum iff:

$$\textit{label}(t) = \operatorname*{argmax}_{c \in C} \; P(c \mid X(t))$$

- If D has been used as training set, a reliable interpretation of the (training) error *Err*(T, D) in terms of *Err*$^*(T)$ requires the Inductive Learning Hypothesis to hold. This implies, among others, that the distribution of C over the feature space X corresponds to the distribution of C over the training set D.

Decision Trees Basics

Performance of Decision Trees: Classification Error (continued)

Given a decision tree T, a set of examples D, and a node t of T that represents the example subset $D(t) \subseteq D$. Then, the class that is assigned to t, *label*(t), is defined as follows:

$$\textit{label}(t) = \underset{c \in C}{\text{argmin}} \sum_{c' \in C} \textit{cost}(c' \mid c) \cdot \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D(t) : c(\mathbf{x}) = c\}|}{|D(t)|}$$

Decision Trees Basics

Performance of Decision Trees: Classification Error (continued)

Given a decision tree T, a set of examples D, and a node t of T that represents the example subset $D(t) \subseteq D$. Then, the class that is assigned to t, *label*(t), is defined as follows:

$$\textit{label}(t) = \underset{c \in C}{\text{argmin}} \sum_{c' \in C} \textit{cost}(c' \mid c) \cdot \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D(t) : c(\mathbf{x}) = c\}|}{|D(t)|}$$

Misclassification costs of node t wrt. $D(t)$:

$$\textit{Err}_{\textit{cost}}(t, D(t)) = \min_{c \in C} \sum_{c' \in C} \textit{cost}(c' \mid c) \cdot \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D(t) : c(\mathbf{x}) = c\}|}{|D(t)|}$$

Misclassification costs of decision tree T wrt. $D(t)$:

$$\textit{Err}_{\textit{cost}}(T, D) = \sum_{t \in \textit{leaves}(T)} \textit{Err}_{\textit{cost}}(t, D(t)) \cdot \frac{|D(t)|}{|D|}$$

Remarks:

- Again, observe the difference between min(f) and argmin(f). Both expressions minimize f, whereas the former returns the minimum f-value (the image) while the latter returns the argument (the preimage) for which f becomes minimum.

Chapter ML:III

III. Decision Trees

Impurity Functions

Splitting

Let t be a leaf node of an incomplete decision tree, and let $D(t)$ be the subset of the example set D that is represented by t. [Illustration]

Possible criteria for a splitting of $X(t)$:

1. Size of $D(t)$.

2. Purity of $D(t)$.

3. Occam's Razor.

Impurity Functions

Splitting

Let t be a leaf node of an incomplete decision tree, and let $D(t)$ be the subset of the example set D that is represented by t. [Illustration]

Possible criteria for a splitting of $X(t)$:

1. Size of $D(t)$.

   $D(t)$ will not be partitioned further if the number of examples, $|D(t)|$, is below a certain threshold.

2. Purity of $D(t)$.

   $D(t)$ will not be partitioned further if all examples in D are members of the same class.

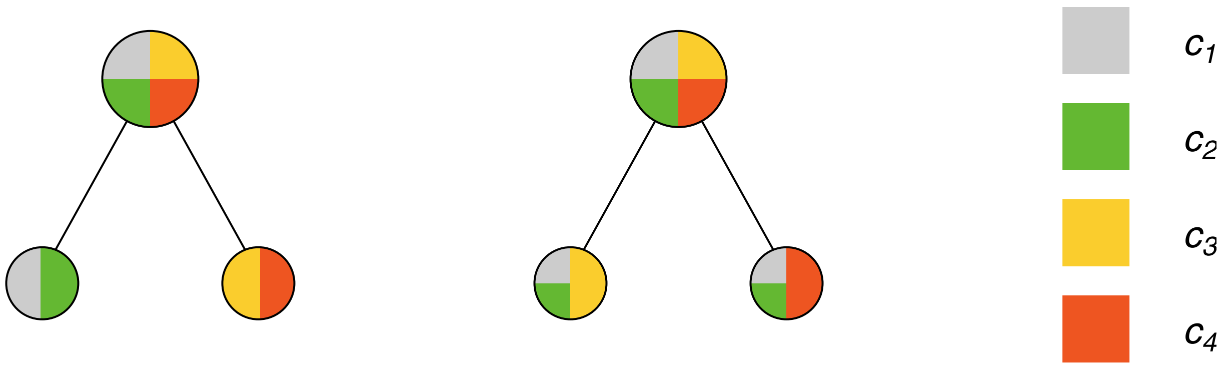
3. Occam's Razor.

   $D(t)$ will not be partitioned further if the resulting decision tree is not improved significantly by the splitting.

Impurity Functions

Splitting (continued)

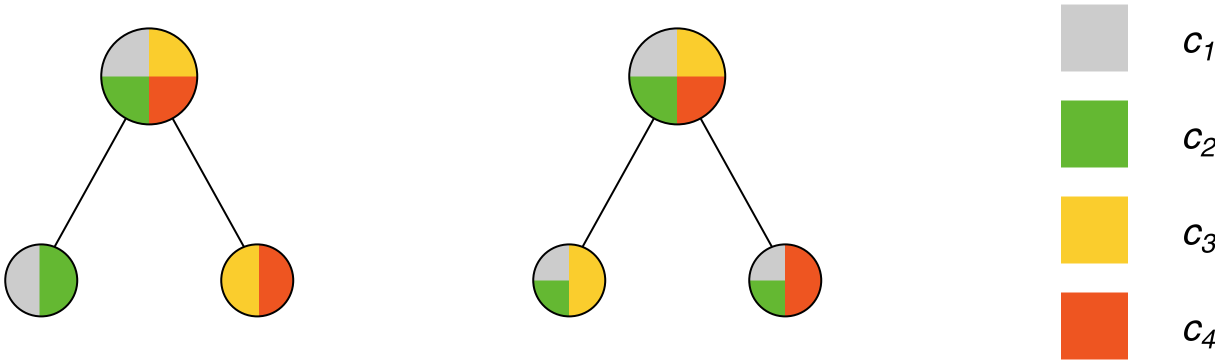
Let D be a set of examples over a feature space X and a set of classes $C = \{c_1, c_2, c_3, c_4\}$. Distribution of D for two possible splittings of X :



Impurity Functions

Splitting (continued)

Let D be a set of examples over a feature space X and a set of classes $C = \{c_1, c_2, c_3, c_4\}$. Distribution of D for two possible splittings of X :



- The left splitting should be preferred, since it minimizes the *impurity* of the subsets of D in the leaf nodes. The argumentation presumes that the misclassification costs are independent of the classes in C.
- The impurity is a function defined on $\mathcal{P}(D)$, the set of all subsets of an example set D.

Impurity Functions

Definition 3 (Impurity Function ι)

Let $k \in \mathbf{N}$. An impurity function $\iota : [0;1]^k \to \mathbf{R}$ is a partial function which is defined for those tuples $(p_1, \ldots, p_k)$ where $\sum_{i=1}^{k} p_i = 1$, $p_i \geq 0$, and for which the following properties hold:

(a) ι becomes minimum at points $(1, 0, \ldots, 0), (0, 1, \ldots, 0), \ldots, (0, \ldots, 0, 1)$.

(b) ι is symmetric with regard to its arguments, $p_1, \ldots, p_k$.

(c) ι becomes maximum at point $(1/k, \ldots, 1/k)$.

Impurity Functions

Definition 4 (Impurity of an Example Set $\iota(D)$)

Let D be a set of examples, let $C = \{c_1, \ldots, c_k\}$ be set of classes, and let $c : X \rightarrow C$ be the ideal classifier for X. Moreover, let $\iota : [0;1]^k \rightarrow \mathbf{R}$ an impurity function. Then, the impurity of D, denoted as $\iota(D)$, is defined as follows:

$$\iota(D) = \iota\left(\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|}, \ldots, \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_k\}|}{|D|}\right)$$

Impurity Functions

Definition 4 (Impurity of an Example Set $\iota(D)$)

Let D be a set of examples, let $C = \{c_1, \ldots, c_k\}$ be set of classes, and let $c : X \to C$ be the ideal classifier for X. Moreover, let $\iota : [0;1]^k \to \mathbf{R}$ an impurity function. Then, the impurity of D, denoted as $\iota(D)$, is defined as follows:

$$\iota(D) = \iota\left(\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|}, \ldots, \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_k\}|}{|D|}\right)$$

Definition 5 (Impurity Reduction $\Delta\iota$)

Let $D_1, \ldots, D_s$ be a partitioning of an example set D, which is induced by a splitting of a feature space X. Then, the resulting impurity reduction, denoted as $\Delta\iota(D, \{D_1, \ldots, D_s\})$, is defined as follows:

$$\Delta\iota(D, \{D_1, \ldots, D_s\}) = \iota(D) - \sum_{j=1}^{s} \frac{|D_j|}{|D|} \cdot \iota(D_j)$$

Remarks:

- ❑ Observe the different signatures of the impurity function ι within the two definitions.
- ❑ The properties in the definition of ι suggest to minimize the external path length of T with respect to D in order to minimize the overall impurity characteristics of T.
- ❑ Within the *DT-construct* algorithm usually a greedy strategy (local optimization) is employed to minimize the overall impurity characteristics of a decision tree T.

Impurity Functions

Impurity Functions Based on the Misclassification Rate

Definition for two classes:

$$\iota_{misclass}(p_1, p_2) = 1 - \max\{p_1, p_2\} = \begin{cases} p_1 & \text{if } 0 \le p_1 \le 0.5 \\ 1 - p_1 & \text{otherwise} \end{cases}$$

$$\iota_{misclass}(D) = 1 - \max\left\{ \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|}, \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|} \right\}$$

Impurity Functions

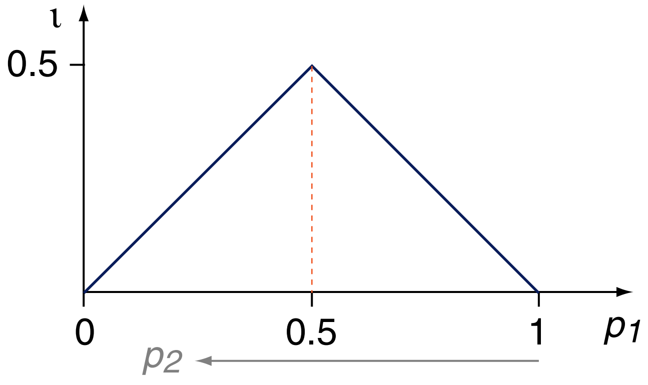
Impurity Functions Based on the Misclassification Rate

Definition for two classes:

$$\iota_{misclass}(p_1, p_2) = 1 - \max\{p_1, p_2\} = \begin{cases} p_1 & \text{if } 0 \leq p_1 \leq 0.5 \\ 1 - p_1 & \text{otherwise} \end{cases}$$

$$\iota_{misclass}(D) = 1 - \max\left\{ \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|}, \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|} \right\}$$

Graph of the function $\iota_{misclass}(p_1, 1 - p_1)$:



Impurity Functions

Impurity Functions Based on the Misclassification Rate (continued)

Definition for k classes:

$$\iota_{\textit{misclass}}(p_1, \ldots, p_k) = 1 - \max_{i=1,\ldots,k} p_i$$

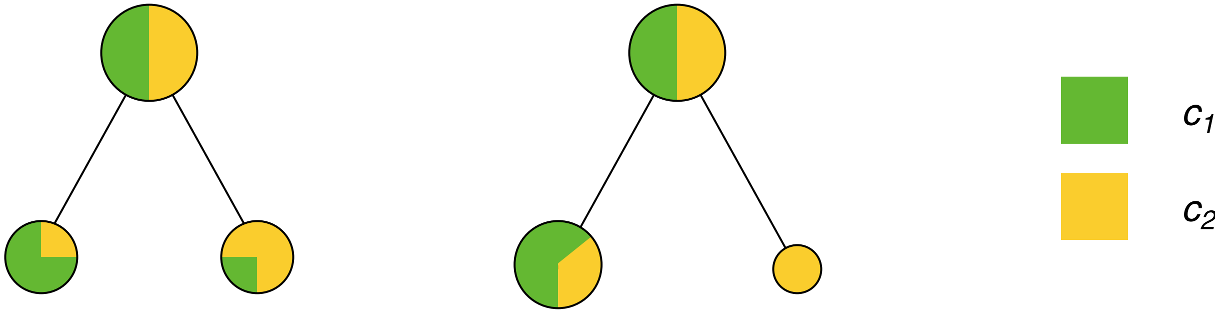
$$\iota_{\textit{misclass}}(D) = 1 - \max_{c \in C} \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c\}|}{|D|}$$

Impurity Functions

Impurity Functions Based on the Misclassification Rate (continued)

Problems:

- $\Delta\iota_{misclass} = 0$ may hold for all possible splittings.
- The impurity function that is induced by the misclassification rate underestimates pure nodes (see splitting on the right-hand side):

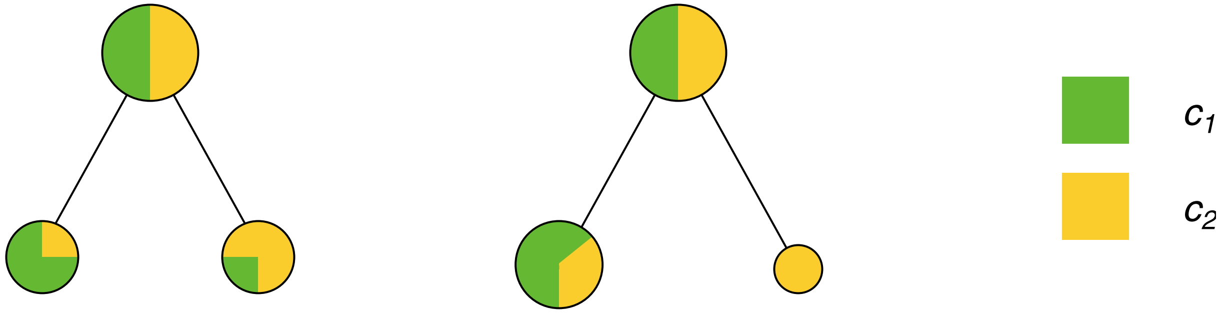


Impurity Functions

Impurity Functions Based on the Misclassification Rate (continued)

Problems:

- $\Delta\iota_{misclass} = 0$ may hold for all possible splittings.
- The impurity function that is induced by the misclassification rate underestimates pure nodes (see splitting on the right-hand side):



$$\Delta\iota_{misclass} = \iota_{misclass}(D) - \left(\frac{|D_1|}{|D|} \cdot \iota_{misclass}(D_1) + \frac{|D_2|}{|D|} \cdot \iota_{misclass}(D_2)\right)$$

left splitting: $\Delta\iota_{misclass} = \frac{1}{2} - (\frac{1}{2} \cdot \frac{1}{4} + \frac{1}{2} \cdot \frac{1}{4}) = \frac{1}{4}$

right splitting: $\Delta\iota_{misclass} = \frac{1}{2} - (\frac{3}{4} \cdot \frac{1}{3} + \frac{1}{4} \cdot 0) = \frac{1}{4}$

Impurity Functions

Definition 6 (Strict Impurity Function)

Let $k \in \mathbf{N}$ and let $\iota : [0;1]^k \to \mathbf{R}$ be an impurity function. ι is called strict, if it is strictly concave:

(c) $\to$ (c') $\quad \iota(\lambda\mathbf{p} + (1-\lambda)\mathbf{p}') \;>\; \lambda\,\iota(\mathbf{p}) + (1-\lambda)\,\iota(\mathbf{p}'), \quad 1 < \lambda < 1,\; \mathbf{p} \neq \mathbf{p}'$

Impurity Functions

Definition 6 (Strict Impurity Function)

Let $k \in \mathbf{N}$ and let $\iota : [0;1]^k \to \mathbf{R}$ be an impurity function. ι is called strict, if it is strictly concave:

(c) $\rightarrow$ (c') $\quad \iota(\lambda\mathbf{p} + (1-\lambda)\mathbf{p}') \; > \; \lambda\,\iota(\mathbf{p}) + (1-\lambda)\,\iota(\mathbf{p}'), \quad 1 < \lambda < 1,\; \mathbf{p} \neq \mathbf{p}'$

Lemma 1

Let ι be a strict impurity function and let $D_1, \ldots, D_s$ be a partitioning of an example set D, which is induced by a splitting of a feature space X. Then the following inequality holds:

$$\Delta\iota(D, \{D_1, \ldots, D_s\}) \geq 0$$

The equality is given iff for all $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, s\}$ holds:

$$\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_i\}|}{|D|} = \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D_j : c(\mathbf{x}) = c\}|}{|D_j|}$$

Remarks:

- Note that $\mathbf{p}$ and $\mathbf{p}'$ in the definition denote vectors from $[0;1]^k$.
- For two classes, strict concavity means $\iota(p_1, 1-p_1) < 0$, where $0 < p_1 < 1$.
- Strict concavity entails Property (c) of the impurity function definition.
- If ι is a twice differentiable function, strict concavity is equivalent with a negative definite Hessian of ι.
- With properly chosen coefficients, polynomials of second degree fulfill the properties (a) and (b) of the impurity function definition as well as strict concavity. See impurity functions based on the Gini index in this regard.
- The impurity function that is induced by the misclassification rate is concave, but it is not strictly concave.
- The proof of the Lemma exploits the strict concavity property of ι.

Impurity Functions

Impurity Functions Based on Entropy

Definition 7 (Entropy)

Let A denote an event and let $P(A)$ denote the occurrence probability of A. Then the entropy (self-information, information content) of A is defined as $-\log_2(P(A))$.

Let $\mathcal{A}$ be an experiment with the exclusive outcomes (events) $A_1, \ldots, A_k$. Then the mean information content of $\mathcal{A}$, denoted as $H(\mathcal{A})$, is called Shannon entropy or entropy of experiment $\mathcal{A}$ and is defined as follows:

$$H(\mathcal{A}) = -\sum_{i=1}^{k} P(A_i) \log_2(P(A_i))$$

Remarks:

- The smaller the occurrence probability of an event, the larger is its entropy. An event that is certain has zero entropy.
- The Shannon entropy combines the entropies of an experiment's outcomes, using the outcome probabilities as weights.
- We stipulate the identity $0 \cdot \log_2(0) = 0$.

Impurity Functions

Impurity Functions Based on Entropy (continued)

Definition 8 (Conditional Entropy, Information Gain)

Let $\mathcal{A}$ be an experiment with the exclusive outcomes (events) $A_1, \ldots, A_k$, and let $\mathcal{B}$ be another experiment with the outcomes $B_1, \ldots, B_s$. Then the conditional entropy of the combined experiment $(\mathcal{A} \mid \mathcal{B})$ is defined as follows:

$$H(\mathcal{A} \mid \mathcal{B}) = \sum_{j=1}^{s} P(B_j) \cdot H(\mathcal{A} \mid B_j),$$

$$\text{where} \quad H(\mathcal{A} \mid B_j) = -\sum_{i=1}^{k} P(A_i \mid B_j) \log_2(P(A_i \mid B_j))$$

Impurity Functions

Impurity Functions Based on Entropy (continued)

Definition 8 (Conditional Entropy, Information Gain)

Let $\mathcal{A}$ be an experiment with the exclusive outcomes (events) $A_1, \ldots, A_k$, and let $\mathcal{B}$ be another experiment with the outcomes $B_1, \ldots, B_s$. Then the conditional entropy of the combined experiment $(\mathcal{A} \mid \mathcal{B})$ is defined as follows:

$$H(\mathcal{A} \mid \mathcal{B}) = \sum_{j=1}^{s} P(B_j) \cdot H(\mathcal{A} \mid B_j),$$

$$\text{where} \quad H(\mathcal{A} \mid B_j) = -\sum_{i=1}^{k} P(A_i \mid B_j) \log_2(P(A_i \mid B_j))$$

Impurity Functions

Impurity Functions Based on Entropy (continued)

Definition 8 (Conditional Entropy, Information Gain)

Let $\mathcal{A}$ be an experiment with the exclusive outcomes (events) $A_1, \ldots, A_k$, and let $\mathcal{B}$ be another experiment with the outcomes $B_1, \ldots, B_s$. Then the conditional entropy of the combined experiment $(\mathcal{A} \mid \mathcal{B})$ is defined as follows:

$$H(\mathcal{A} \mid \mathcal{B}) = \sum_{j=1}^{s} P(B_j) \cdot H(\mathcal{A} \mid B_j),$$

where $$H(\mathcal{A} \mid B_j) = -\sum_{i=1}^{k} P(A_i \mid B_j) \log_2(P(A_i \mid B_j))$$

The information gain due to experiment $\mathcal{B}$ is defined as follows:

$$H(\mathcal{A}) - H(\mathcal{A} \mid \mathcal{B}) = H(\mathcal{A}) - \sum_{j=1}^{s} P(B_j) \cdot H(\mathcal{A} \mid B_j)$$

Remarks:

- Information gain is defined as reduction in entropy.
- In the context of decision trees, experiment $\mathcal{A}$ corresponds to classifying feature vector $\mathbf{x}$ with regard to the target concept. A possible question, whose answer will inform us about which event $A_i \in \mathcal{A}$ occurred, is the following: “Does $\mathbf{x}$ belong to class c_i?”

  Likewise, experiment $\mathcal{B}$ corresponds to evaluating feature B of feature vector $\mathbf{x}$. A possible question, whose answer will inform us about which event $B_j \in \mathcal{B}$ occurred, is the following: “Does $\mathbf{x}$ have value b_j for feature B?”
- Rationale: Typically, the events “target concept class” and “feature value” are statistically dependent. Hence, the conditional entropy $H(\mathcal{A} \mid \mathcal{B})$ of class $c(\mathbf{x})$ will become smaller if one learns the value of some feature of $\mathbf{x}$ (recall that the class of $\mathbf{x}$ is unknown). We experience an information gain with regard to the outcome of experiment $\mathcal{A}$, which is rooted in our information about the outcome of experiment $\mathcal{B}$. Under no circumstances the information gain will be negative; it is zero if the involved events are statistically independent.
- Since $H(\mathcal{A})$ is constant, the feature that provides the maximum information gain (= the maximally informative feature) is given by the minimization of $H(\mathcal{A} \mid \mathcal{B})$.
- The expanded form of $H(\mathcal{A} \mid \mathcal{B})$ reads as follows:

$$H(\mathcal{A} \mid \mathcal{B}) = -\sum_{j=1}^{s} P(B_j) \cdot \sum_{i=1}^{k} P(A_i \mid B_j) \log_2(P(A_i \mid B_j))$$

Impurity Functions

Impurity Functions Based on Entropy (continued)

Definition for two classes:

$$\iota_{entropy}(p_1, p_2) = -(p_1 \log_2(p_1) + p_2 \log_2(p_2))$$

$$\iota_{entropy}(D) = -\left(\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|} \cdot \log_2 \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|} + \right.$$

$$\left. \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|} \cdot \log_2 \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|}\right)$$

Impurity Functions

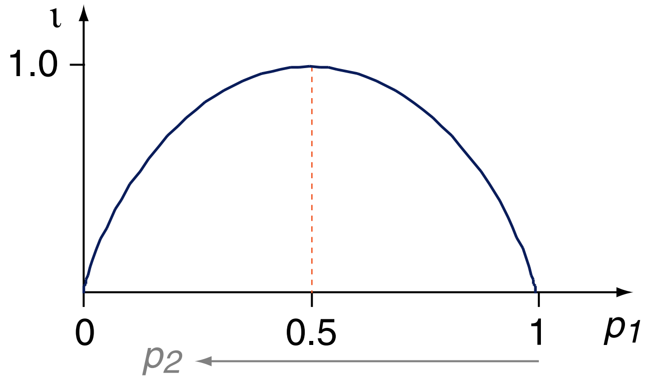
Impurity Functions Based on Entropy (continued)

Definition for two classes:

$$\iota_{entropy}(p_1, p_2) = -(p_1 \log_2(p_1) + p_2 \log_2(p_2))$$

$$\iota_{entropy}(D) = -\left(\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|} \cdot \log_2 \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|} + \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|} \cdot \log_2 \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|} \right)$$

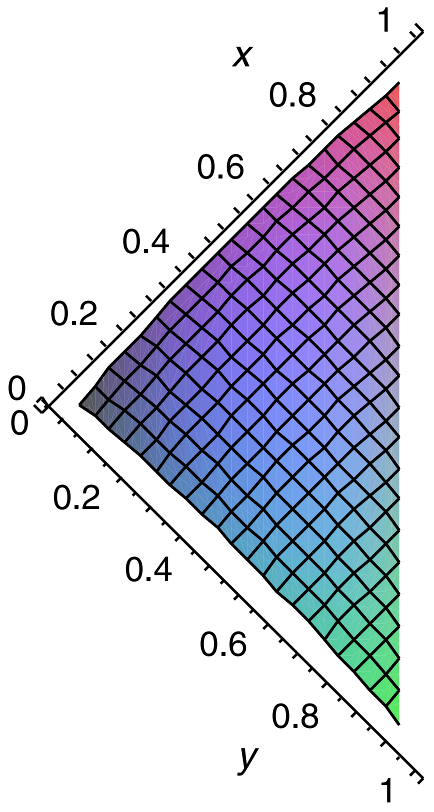
Graph of the function $\iota_{entropy}(p_1, 1 - p_1)$:

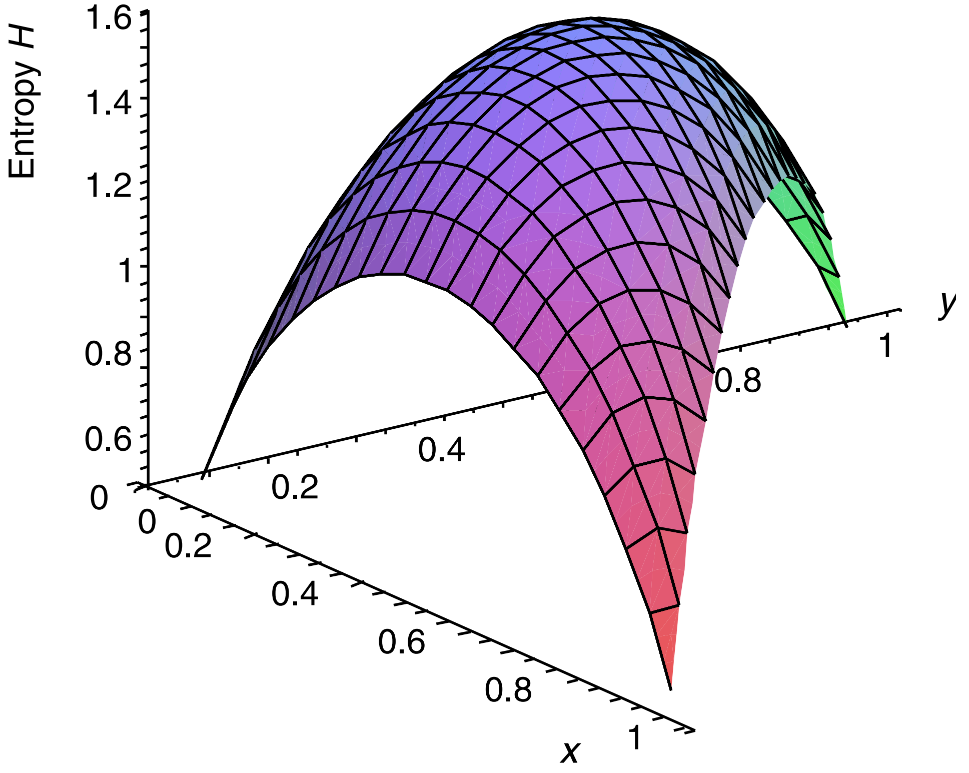


Impurity Functions

Impurity Functions Based on Entropy (continued)

Graph of the function $\iota_{entropy}(p_1, p_2, 1 - p_1 - p_2)$:





Impurity Functions

Impurity Functions Based on Entropy (continued)

Definition for k classes:

$$\iota_{\textit{entropy}}(p_1, \ldots, p_k) = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

$$\iota_{\textit{entropy}}(D) = -\sum_{i=1}^{k} \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_i\}|}{|D|} \cdot \log_2 \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_i\}|}{|D|}$$

Impurity Functions

Impurity Functions Based on Entropy (continued)

$\underline{\Delta\iota}_{\textit{entropy}}$ corresponds to the information gain $H(\mathcal{A}) - H(\mathcal{A} \mid \mathcal{B})$:

$$\underline{\Delta\iota}_{\textit{entropy}} = \underbrace{\iota_{\textit{entropy}}(D)}_{H(\mathcal{A})} \quad - \quad \underbrace{\sum_{j=1}^{s} \frac{|D_j|}{|D|} \cdot \iota_{\textit{entropy}}(D_j)}_{H(\mathcal{A}|\mathcal{B})}$$

Impurity Functions

Impurity Functions Based on Entropy (continued)

$\Delta\iota_{\mathit{entropy}}$ corresponds to the information gain $H(\mathcal{A}) - H(\mathcal{A} \mid \mathcal{B})$:

$$\Delta\iota_{\mathit{entropy}} = \underbrace{\iota_{\mathit{entropy}}(D)}_{H(\mathcal{A})} \quad - \quad \underbrace{\sum_{j=1}^{s} \frac{|D_j|}{|D|} \cdot \iota_{\mathit{entropy}}(D_j)}_{H(\mathcal{A}|\mathcal{B})}$$

Legend:

- $\iota_{\mathit{entropy}}(D) = \iota_{\mathit{entropy}}(P(A_1), \ldots, P(A_k))$
- $\iota_{\mathit{entropy}}(D_j) = \iota_{\mathit{entropy}}(P(A_1 \mid B_j), \ldots, P(A_k \mid B_j)),\ j = 1, \ldots, s$
- $\iota_{\mathit{entropy}}(p_1, \ldots, p_k) = -\sum_{i=1}^{k} p_i \cdot \log_2(p_i)$
- $\frac{|D_j|}{|D|} = P(B_j),\ j = 1, \ldots, s$
- $A_i,\ i = 1, \ldots, k$, denotes the event that $\mathbf{x} \in X(t)$ belongs to class c_i. The experiment $\mathcal{A}$ corresponds to the classification $c : X(t) \to C$.
- $B_j,\ j = 1, \ldots, s$, denotes the event that $\mathbf{x} \in X(t)$ has value b_j for feature B. The experiment $\mathcal{B}$ corresponds to evaluating feature B and entails the following splitting:
  $X(t) = X(t_1) \cup \ldots \cup X(t_s) = \{\mathbf{x} \in X(t) : \mathbf{x}|_B = b_1\} \cup \ldots \cup \{\mathbf{x} \in X(t) : \mathbf{x}|_B = b_s\}$
- $P(A_i), P(B_j), P(A_i \mid B_j)$ are estimated as relative frequencies based on D.

Impurity Functions

Impurity Functions Based on the Gini Index

Definition for two classes:

$$\iota_{Gini}(p_1, p_2) = p_1 \cdot p_2$$

$$\iota_{Gini}(D) = \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|} \cdot \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|}$$

Impurity Functions

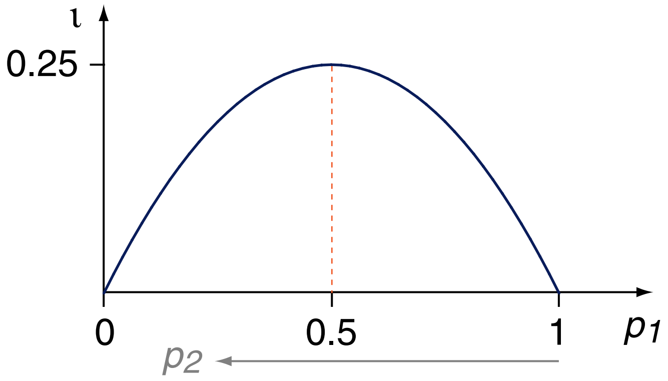
Impurity Functions Based on the Gini Index

Definition for two classes:

$$\iota_{Gini}(p_1, p_2) = p_1 \cdot p_2$$

$$\iota_{Gini}(D) = \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|} \cdot \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|}$$

Graph of the function $\iota_{Gini}(p_1, 1 - p_1)$:



Impurity Functions

Impurity Functions Based on the Gini Index (continued)

Definition for k classes:

$$\iota_{\mathit{Gini}}(p_1, \ldots, p_k) = 1 - \sum_{i=1}^{k} (p_i)^2$$

$$\iota_{\mathit{Gini}}(D) = \left(\sum_{i=1}^{k} \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_i\}|}{|D|} \right)^2 - \sum_{i=1}^{k} \left(\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_i\}|}{|D|} \right)^2$$

$$= 1 - \sum_{i=1}^{k} \left(\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_i\}|}{|D|} \right)^2$$

Chapter ML:III

III. Decision Trees

- ❑ Decision Trees Basics
- ❑ Impurity Functions
- ❑ Decision Tree Algorithms
- ❑ Decision Tree Pruning

Decision Tree Algorithms

ID3 Algorithm [Quinlan 1986] [CART Algorithm]

Characterization of the model (model world) [ML Introduction] :

- X is a set of feature vectors, also called feature space.
- C is a set of classes.
- $c : X \to C$ is the ideal classifier for X.
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$ is a set of examples.

Task: Based on D, construction of a decision tree T to approximate c.

Decision Tree Algorithms

ID3 Algorithm [Quinlan 1986] [CART Algorithm]

Characterization of the model (model world) [ML Introduction] :

- X is a set of feature vectors, also called feature space.
- C is a set of classes.
- $c : X \to C$ is the ideal classifier for X.
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$ is a set of examples.

Task: Based on D, construction of a decision tree T to approximate c.

Characteristics of the ID3 algorithm:

1. Each splitting is based on one nominal feature and considers its complete domain. Splitting based on feature A with domain $\{a_1, \ldots, a_k\}$:

$$X = \{\mathbf{x} \in X : \mathbf{x}|_A = a_1\} \cup \ldots \cup \{\mathbf{x} \in X : \mathbf{x}|_A = a_k\}$$

2. Splitting criterion is the information gain.

Decision Tree Algorithms

ID3 Algorithm [Mitchell 1997] [algorithm template]

ID3(D, Attributes, Target)

- Create a node t for the tree.
- If all examples in D are positive, return the single-node tree t with label “+”.
- If all examples in D are negative, return the single-node tree t, with label “–”.
- Label t with the most common value of Target in D.
- If Attributes is empty, return the single-node tree t.
- Otherwise:
  - Let A* be the attribute from Attributes that best classifies examples in D.
  - The decision attribute of t is A*.
  - For each possible value “a” in A*:
    - Add a new tree branch below t, corresponding to the test A* = “a”.
    - Let D_a be the subset of D that has value “a” for A*.
    - If D_a is empty:
      Then below this new branch add a leaf node with label of the most common value of Target in D.
      Else below this new branch add the subtree ID3(D_a, Attributes \ {A*}, Target).
- Return t.

Decision Tree Algorithms

ID3 Algorithm (pseudo code) [algorithm template]

ID3$(D,$ *Attributes*, *Target*$)$

```
1.  $t =$ createNode()

2.  IF $\forall\langle \mathbf{x}, c(\mathbf{x})\rangle \in D : c(\mathbf{x}) = 1$ THEN label($t$) = '+', return($t$) ENDIF
3.  IF $\forall\langle \mathbf{x}, c(\mathbf{x})\rangle \in D : c(\mathbf{x}) = 0$ THEN label($t$) = '−', return($t$) ENDIF
4.  label($t$) = mostCommonClass($D$, Target)
5.  IF Attributes $= \emptyset$ THEN return($t$) ENDIF

6.

7.

8.
```

Decision Tree Algorithms

ID3 Algorithm (pseudo code) [algorithm template]

ID3$(D$, *Attributes*, *Target*$)$

```
1.  $t$ = createNode()

2.  IF $\forall\langle\mathbf{x}, c(\mathbf{x})\rangle \in D : c(\mathbf{x}) = 1$ THEN label($t$) = '+', return($t$) ENDIF
3.  IF $\forall\langle\mathbf{x}, c(\mathbf{x})\rangle \in D : c(\mathbf{x}) = 0$ THEN label($t$) = '−', return($t$) ENDIF
4.  label($t$) = mostCommonClass($D$, Target)
5.  IF Attributes $= \emptyset$ THEN return($t$) ENDIF

6.  $A^* = \text{argmax}_{A \in Attributes}(informationGain(D, A))$

7.

8.
```

Decision Tree Algorithms

ID3 Algorithm (pseudo code) [algorithm template]

```
ID3($D$, Attributes, Target)

  1.  $t$ = createNode()

  2.  IF $\forall \langle \mathbf{x}, c(\mathbf{x}) \rangle \in D : c(\mathbf{x}) = 1$ THEN label($t$) = '+', return($t$) ENDIF
  3.  IF $\forall \langle \mathbf{x}, c(\mathbf{x}) \rangle \in D : c(\mathbf{x}) = 0$ THEN label($t$) = '−', return($t$) ENDIF
  4.  label($t$) = mostCommonClass($D$, Target)
  5.  IF Attributes $= \emptyset$ THEN return($t$) ENDIF

  6.  $A^* = \text{argmax}_{A \in \textit{Attributes}}(\textit{informationGain}(D, A))$

  7.  FOREACH $a \in A^*$ DO
        $D_a = \{(\mathbf{x}, c(\mathbf{x})) \in D : \mathbf{x}|_{A^*} = a\}$
        IF $D_a = \emptyset$ THEN



        ELSE
          createEdge($t, a$, ID3($D_a$, Attributes $\setminus \{A^*\}$, Target))
        ENDIF

      ENDDO
  8.  return($t$)
```

Decision Tree Algorithms

ID3 Algorithm (pseudo code) [algorithm template]

```
ID3($D$, Attributes, Target)

1.  $t$ = createNode()

2.  IF $\forall\langle\mathbf{x}, c(\mathbf{x})\rangle \in D : c(\mathbf{x}) = 1$ THEN label($t$) = '+', return($t$) ENDIF
3.  IF $\forall\langle\mathbf{x}, c(\mathbf{x})\rangle \in D : c(\mathbf{x}) = 0$ THEN label($t$) = '−', return($t$) ENDIF
4.  label($t$) = mostCommonClass($D$, Target)
5.  IF Attributes $= \emptyset$ THEN return($t$) ENDIF

6.  $A^* = \text{argmax}_{A \in \text{Attributes}}(\text{informationGain}(D, A))$

7.  FOREACH $a \in A^*$ DO

      $D_a = \{(\mathbf{x}, c(\mathbf{x})) \in D : \mathbf{x}|_{A^*} = a\}$
      IF $D_a = \emptyset$ THEN
         $t'$ = createNode()
         label($t'$) = mostCommonClass($D$, Target)
         createEdge($t, a, t'$)
      ELSE
         createEdge($t, a$, ID3($D_a$, Attributes $\setminus \{A^*\}$, Target))
      ENDIF

    ENDDO
8.  return($t$)
```

Decision Tree Algorithms

ID3 Algorithm: Example

Example set D for mushrooms, implicitly defining a feature space X over the three dimensions color, size, and points:

	Color	Size	Points	Eatability
1	red	small	yes	toxic
2	brown	small	no	eatable
3	brown	large	yes	eatable
4	green	small	no	eatable
5	red	large	no	eatable

Decision Tree Algorithms

ID3 Algorithm: Example (continued)

First recursion step, splitting with regard to the feature “color” :

$D|_{\text{color}} =$

	toxic	eatable
red	1	1
brown	0	2
green	0	1

➜ $|D_{\text{red}}| = 2,\ |D_{\text{brown}}| = 2,\ |D_{\text{green}}| = 1$

Decision Tree Algorithms

ID3 Algorithm: Example (continued)

First recursion step, splitting with regard to the feature “color” :

$D|_{\text{color}} =$

	toxic	eatable
red	1	1
brown	0	2
green	0	1

➜ $|D_{\text{red}}| = 2,\ |D_{\text{brown}}| = 2,\ |D_{\text{green}}| = 1$

Estimated a-priori probabilities:

$$p_{\text{red}} = \frac{2}{5} = 0.4, \quad p_{\text{brown}} = \frac{2}{5} = 0.4, \quad p_{\text{green}} = \frac{1}{5} = 0.2$$

Conditional entropy:

$$\begin{aligned} H(C \mid \text{color}) &= -(0.4(\tfrac{1}{2}\log_2\tfrac{1}{2} + \tfrac{1}{2}\log_2\tfrac{1}{2}) + \\ &\quad 0.4(\tfrac{0}{2}\log_2\tfrac{0}{2} + \tfrac{2}{2}\log_2\tfrac{2}{2}) + \\ &\quad 0.2(\tfrac{0}{1}\log_2\tfrac{0}{1} + \tfrac{1}{1}\log_2\tfrac{1}{1})) = 0.4 \end{aligned}$$

$$\begin{aligned} H(C \mid \text{size}) &\approx 0.55 \\ H(C \mid \text{points}) &= 0.4 \end{aligned}$$

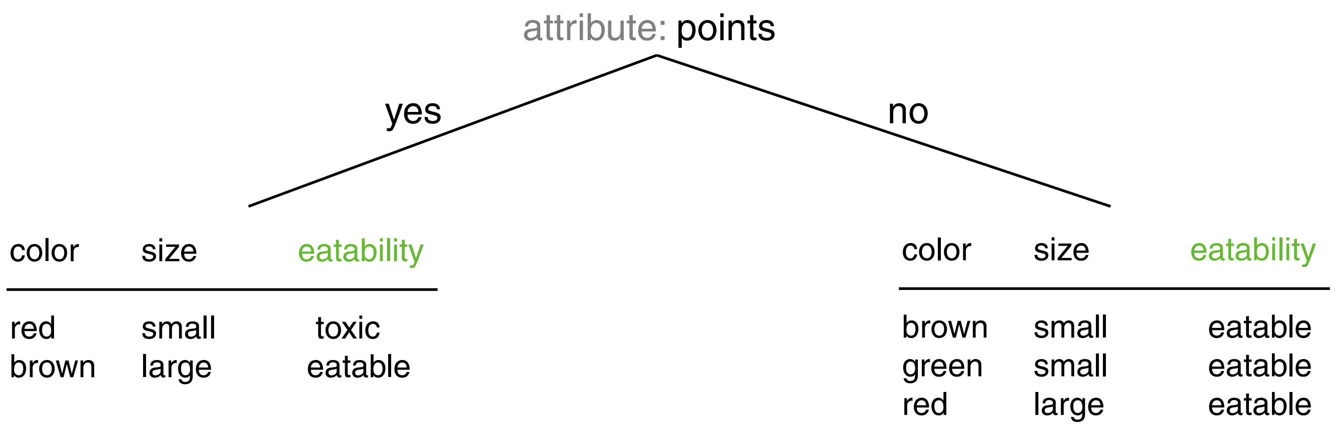
Remarks:

- The smaller $H(C \mid \text{feature})$ is, the larger becomes the information gain. Hence, the difference $H(C) - H(C \mid \text{feature})$ needs not to be computed since $H(C)$ is constant within each recursion step.
- In the example, the information gain in the first recursion step is maximum for the two features "color" and "points".

Decision Tree Algorithms

ID3 Algorithm: Example (continued)

Decision tree after first recursion step:



color	size	eatability
red	small	toxic
brown	large	eatable

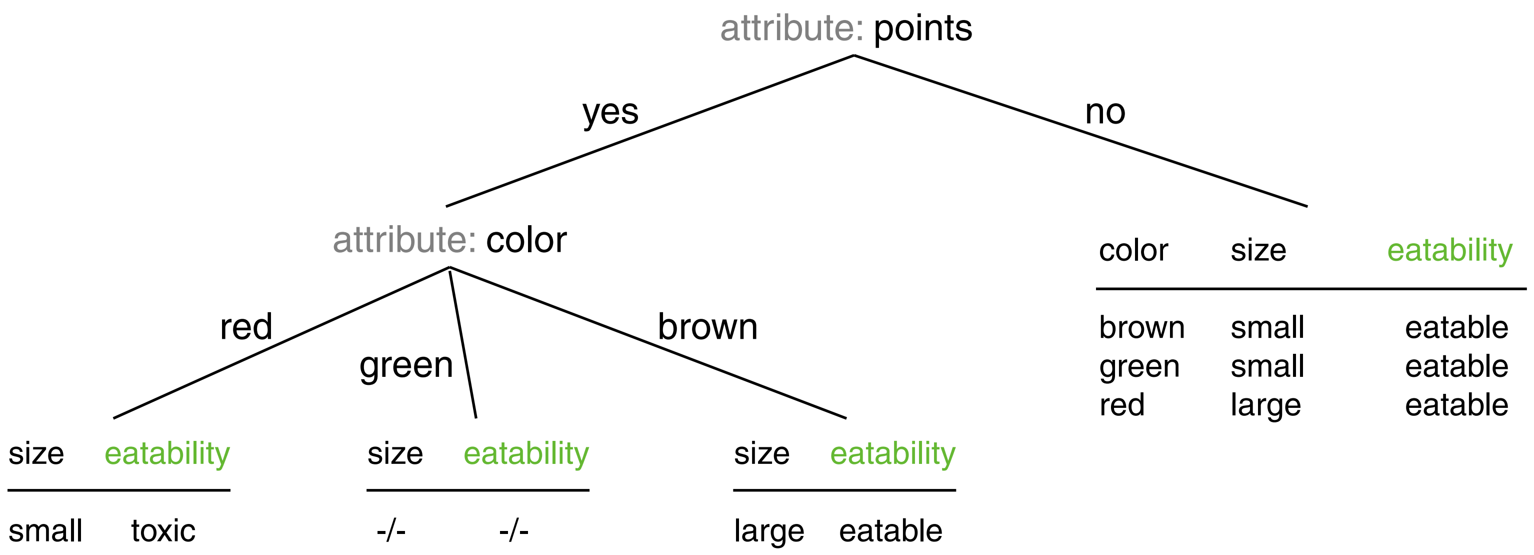
color	size	eatability
brown	small	eatable
green	small	eatable
red	large	eatable

The feature “points” is chosen in Step 5 of the ID3 algorithm.

Decision Tree Algorithms

ID3 Algorithm: Example (continued)

Decision tree after second recursion step:

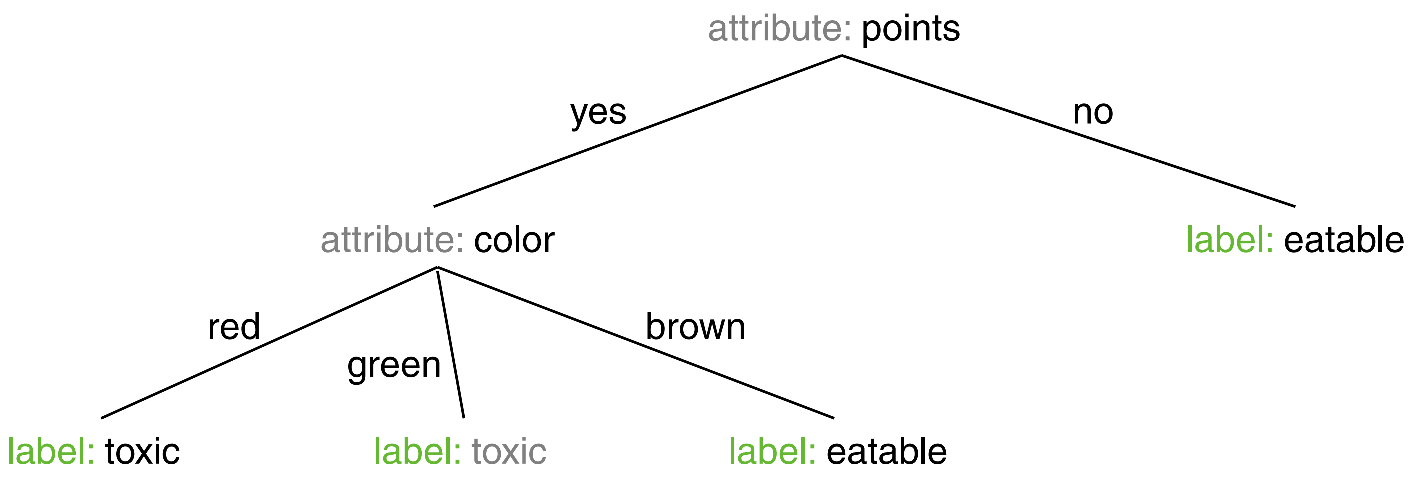


The feature “color” is chosen in Step 5 of the ID3 algorithm.

Decision Tree Algorithms

ID3 Algorithm: Example (continued)

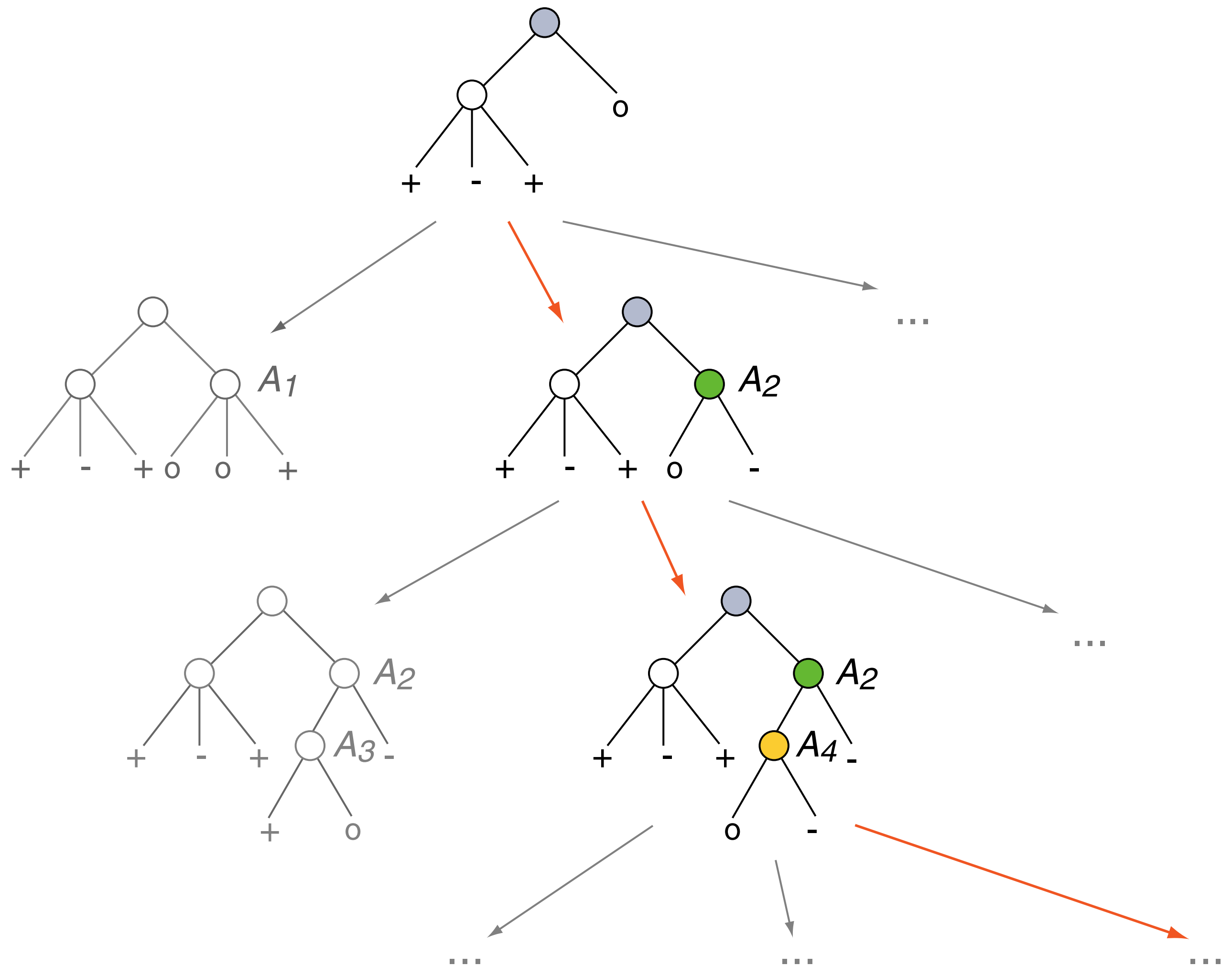
Final decision tree after second recursion step:



Break of a tie by choosing the class “toxic” in Step 7 of the ID3 algorithm.

Decision Tree Algorithms

ID3 Algorithm: Hypothesis Space



Decision Tree Algorithms

ID3 Algorithm: Inductive Bias

Inductive bias is the rigidity in applying the (little bit of) knowledge learned from a training set for the classification of unseen feature vectors.

Observations:

- Decision tree search happens in the space of *all* hypotheses.

- The number of decisions of the ID3 algorithm equals the feature number.

Decision Tree Algorithms

ID3 Algorithm: Inductive Bias

Inductive bias is the rigidity in applying the (little bit of) knowledge learned from a training set for the classification of unseen feature vectors.

Observations:

- Decision tree search happens in the space of *all* hypotheses.

  ➜ The target concept is a member of the hypothesis space.

- The number of decisions of the ID3 algorithm equals the feature number.

  ➜ no backtracking takes place

  ➜ *local* optimization of decision trees

Decision Tree Algorithms

ID3 Algorithm: Inductive Bias

Inductive bias is the rigidity in applying the (little bit of) knowledge learned from a training set for the classification of unseen feature vectors.

Observations:

- Decision tree search happens in the space of *all* hypotheses.

  ➜ The target concept is a member of the hypothesis space.

- The number of decisions of the ID3 algorithm equals the feature number.

  ➜ no backtracking takes place
  ➜ *local* optimization of decision trees

Where the inductive bias of the ID3 algorithm becomes obvious:

- Small decision trees are preferred.
- Highly discriminative feature tend to be closer to the root.

Is this justified?

Remarks:

- Let $\mathbf{A}_j$ be the finite domain (the possible values) of feature $A_j,\ j = 1, \ldots, p$, and let C be a set of classes. Then, a hypothesis space H that is comprised of all decision trees corresponds to the set of all functions $h,\ h : \mathbf{A}_1 \times \ldots \times \mathbf{A}_p \rightarrow C$. Typically, $C = \{0, 1\}$.
- The inductive bias of the ID3 algorithm is of a different kind then is the inductive bias of the candidate elimination algorithm (version space algorithm):
  1. The underlying hypothesis space H of the candidate elimination algorithm is incomplete. H corresponds to a coarsened view onto the space of all hypotheses since H contains only conjunctions of attribute-value-pairs as hypotheses. However, this restricted hypothesis space is searched completely by the candidate elimination algorithm.

     Keyword: *restriction bias*
  2. The underlying hypothesis space H of the ID3 algorithm is complete. H corresponds to the set of all discrete functions (from the Cartesian product of the feature domains onto the set of classes) that can be represented in the form of a decision tree. However, this complete hypothesis space is searched incompletely (following a preference).

     Keyword: *preference bias* or *search bias*
- The inductive bias of the ID3 algorithm renders the algorithm robust with respect to noise.

Decision Tree Algorithms

CART Algorithm [Breiman 1984] [ID3 Algorithm]

Characterization of the model (model world) [ML Introduction] :

- X is a set of feature vectors, also called feature space. No restrictions are presumed for the measurement scales of the features.
- C is a set of classes.
- $c : X \to C$ is the ideal classifier for X.
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$ is a set of examples.

Task: Based on D, construction of a decision tree T to approximate c.

Decision Tree Algorithms

CART Algorithm [Breiman 1984] [ID3 Algorithm]

Characterization of the model (model world) [ML Introduction] :

- X is a set of feature vectors, also called feature space. No restrictions are presumed for the measurement scales of the features.
- C is a set of classes.
- $c : X \to C$ is the ideal classifier for X.
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$ is a set of examples.

Task: Based on D, construction of a decision tree T to approximate c.

Characteristics of the CART algorithm:

1. Each splitting is binary and considers one feature at a time.
2. Splitting criterion is the information gain or the Gini index.

Decision Tree Algorithms

CART Algorithm (continued)

1. Let A be a feature with domain $\mathbf{A}$. Ensure a finite number of binary splittings for X by applying the following domain partitioning rules:

   – If A is nominal, choose $\mathbf{A}' \subset \mathbf{A}$ such that $0 < |\mathbf{A}'| \leq |\mathbf{A} \setminus \mathbf{A}'|$.

   – If A is ordinal, choose $a \in \mathbf{A}$ such that $x_{\text{min}} < a < x_{\text{max}}$, where x_{min}, x_{max} are the minimum and maximum values of feature A in D.

   – If A is numeric, choose $a \in \mathbf{A}$ such that $a = (x_k + x_l)/2$, where x_k, x_l are consecutive elements in the ordered value list of feature A in D.

Decision Tree Algorithms

CART Algorithm (continued)

1. Let A be a feature with domain $\mathbf{A}$. Ensure a finite number of binary splittings for X by applying the following domain partitioning rules:

   – If A is nominal, choose $\mathbf{A}' \subset \mathbf{A}$ such that $0 < |\mathbf{A}'| \leq |\mathbf{A} \setminus \mathbf{A}'|$.

   – If A is ordinal, choose $a \in \mathbf{A}$ such that $x_{\text{min}} < a < x_{\text{max}}$, where x_{min}, x_{max} are the minimum and maximum values of feature A in D.

   – If A is numeric, choose $a \in \mathbf{A}$ such that $a = (x_k + x_l)/2$, where x_k, x_l are consecutive elements in the ordered value list of feature A in D.

2. For node t of a decision tree generate all splittings of the above type.

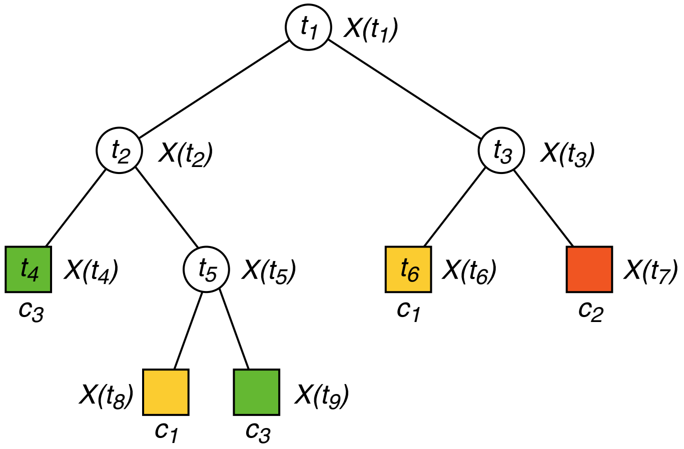
3. Choose a splitting from the set of splittings that maximizes the impurity reduction $\Delta\iota$ (t_L and t_R denote the left and right successor of t) :

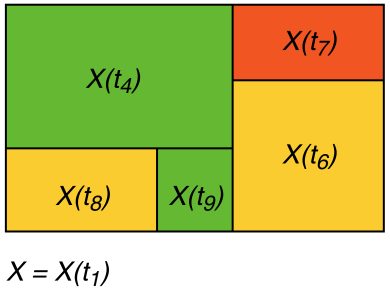
$$\Delta\iota(D(t), \{D(t_L), D(t_R)\}) = \iota(t) - \frac{|D_L)|}{|D|} \cdot \iota(t_L) - \frac{|D_R)|}{|D|} \cdot \iota(t_R)$$

Decision Tree Algorithms

CART Algorithm (continued)

Illustration for two numeric features; i.e., the feature space X corresponds to a two-dimensional plane:





By a sequence of splittings the feature space X is partitioned into rectangles that are parallel to the two axes.

Chapter ML:III

III. Decision Trees

- ❑ Decision Trees Basics
- ❑ Impurity Functions
- ❑ Decision Tree Algorithms
- ❑ Decision Tree Pruning

Decision Tree Pruning

Overfitting

Definition 9 (Overfitting)

Let D be a set of examples and let H be a hypothesis space. The hypothesis $h \in H$ is considered to overfit D if an $h' \in H$ with the following property exists:

$$\mathit{Err}(h, D) < \mathit{Err}(h', D) \quad \text{and} \quad \mathit{Err}^*(h) > \mathit{Err}^*(h')$$

Reasons for overfitting are often rooted in the example set D :

- ❑ D is noisy
- ❑ D is biased (and hence non-representative)
- ❑ D is too small (and hence pretends unrealistic data properties)

Decision Tree Pruning

Overfitting (continued)

Let $D_{tr} \subset D$ be the training set. Then *Err*$^*(h)$ can be estimated with a test set $D_{ts} \subset D$ where $D_{ts} \cap D_{tr} = \emptyset$. The hypothesis $h \in H$ is considered to overfit D if an $h' \in H$ with the following property exists:

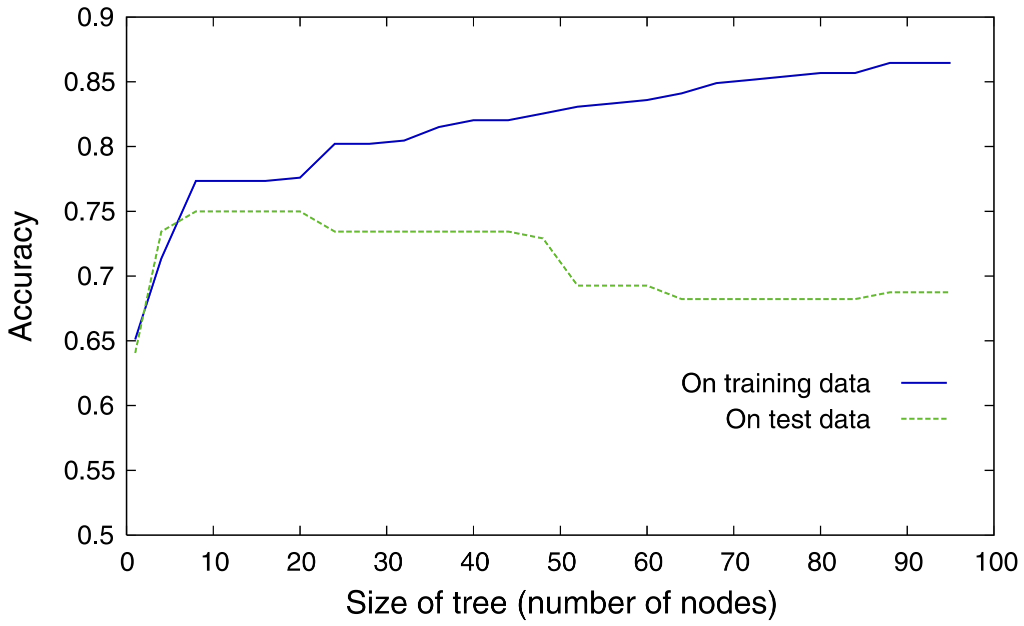
$$\textit{Err}(h, D_{tr}) < \textit{Err}(h', D_{tr}) \quad \text{and} \quad \textit{Err}(h, D_{ts}) > \textit{Err}(h', D_{ts})$$

Decision Tree Pruning

Overfitting (continued)

Let $D_{tr} \subset D$ be the training set. Then *Err*$^*(h)$ can be estimated with a test set $D_{ts} \subset D$ where $D_{ts} \cap D_{tr} = \emptyset$. The hypothesis $h \in H$ is considered to overfit D if an $h' \in H$ with the following property exists:

$$\mathit{Err}(h, D_{tr}) < \mathit{Err}(h', D_{tr}) \quad \text{and} \quad \mathit{Err}(h, D_{ts}) > \mathit{Err}(h', D_{ts})$$



[Mitchell 1997]

Remarks:

- Accuracy is the percentage of correctly classified examples.
- When does *Err*(T, D_{tr}) of a decision tree T become zero?
- The training error *Err*(T, D_{tr}) of a decision tree T is a monotonically decreasing function in the size of T. See Lemma 2.

Decision Tree Pruning

Overfitting (continued)

Lemma 2

Let t be a node in a decision tree T. Then, for each induced splitting $D(t_1), \ldots, D(t_s)$ of a set of examples $D(t)$ holds:

$$\mathit{Err}_{\mathit{cost}}(t, D(t)) \geq \sum_{i \in \{1,\ldots,s\}} \mathit{Err}_{\mathit{cost}}(t_i, D(t_i))$$

The equality is given in the case that all nodes $t, t_1, \ldots, t_s$ represent the same class.

Decision Tree Pruning

Overfitting (continued)

Proof Sketch

$$\begin{aligned}
\textit{Err}_{\textit{cost}}(t, D(t)) &= \min_{c' \in C} \sum_{c \in C} \textit{cost}(c' \mid c) \cdot p(c \mid t) \cdot p(t) \\
&= \sum_{c \in C} \textit{cost}(\textit{label}(t) \mid c) \cdot p(c, t) \\
&= \sum_{c \in C} \textit{cost}(\textit{label}(t) \mid c) \cdot (p(c, t_1) + \ldots + p(c, t_{k_s})) \\
&= \sum_{i \in \{1,\ldots,k_s\}} \sum_{c \in C} \textit{cost}(\textit{label}(t) \mid c) \cdot (p(c, t_i)
\end{aligned}$$

$$\textit{Err}_{\textit{cost}}(t, D(t)) - \textstyle\sum_{i \in \{1,\ldots,k_s\}} \textit{Err}_{\textit{cost}}(t_i, D(t_i)) =$$

$$\sum_{i \in \{1,\ldots,k_s\}} \left(\sum_{c \in C} \textit{cost}(\textit{label}(t) \mid c) \cdot (p(c, t_i) \; - \min_{c' \in C} \sum_{c \in C} \textit{cost}(c' \mid c) \cdot p(c, t_i) \right)$$

The summands on the right equation side are greater than or equal to zero.

Remarks:

- Lemma 2 does also hold if the misclassification rate is used as performance measure.
- The algorithm template for the construction of decision trees, *DT-construct*, prefers larger trees, entailing a more fine-grained partitioning of D. A consequence of this behavior is a tendency to overfitting.

Decision Tree Pruning

Overfitting (continued)

Approaches to counter overfitting:

1. Stopping of the decision tree construction process *during* training.
2. Pruning of a decision tree *after* training:
   - ❑ Partitioning of D into three sets for training, validation, and test:
     (a) reduced error pruning
     (b) minimal cost complexity pruning
     (c) rule post pruning
   - ❑ statistical tests such as χ^2 to access generalization capability
   - ❑ heuristic pruning

Decision Tree Pruning

Stopping

Possible criteria for stopping [splitting criteria] :

1. Size of $D(t)$.

   $D(t)$ will not be partitioned further if the number of examples, $|D(t)|$, is below a certain threshold.

2. Purity of $D(t)$.

   $D(t)$ will not be partitioned further if all induced splittings yield no significant impurity reduction $\Delta\iota$.

Problems:

ad 1. A threshold that is too small results in oversized decision trees.

ad 1. A threshold that is too large omits useful splittings.

ad 2. $\Delta\iota$ cannot be extrapolated with regard to the tree height.

Decision Tree Pruning

Pruning

Principle:

1. Construct a sufficiently large decision tree T_{max}.
2. Prune T_{max}, starting from the leaf nodes towards the tree root.

Each leaf node t of T_{max} fulfills one or more of the following conditions:

- $D(t)$ is sufficiently small. Typically, $|D(t)| \leq 5$.
- $D(t)$ is comprised of examples of only one class.
- $D(t)$ is comprised of examples with identical feature vectors.

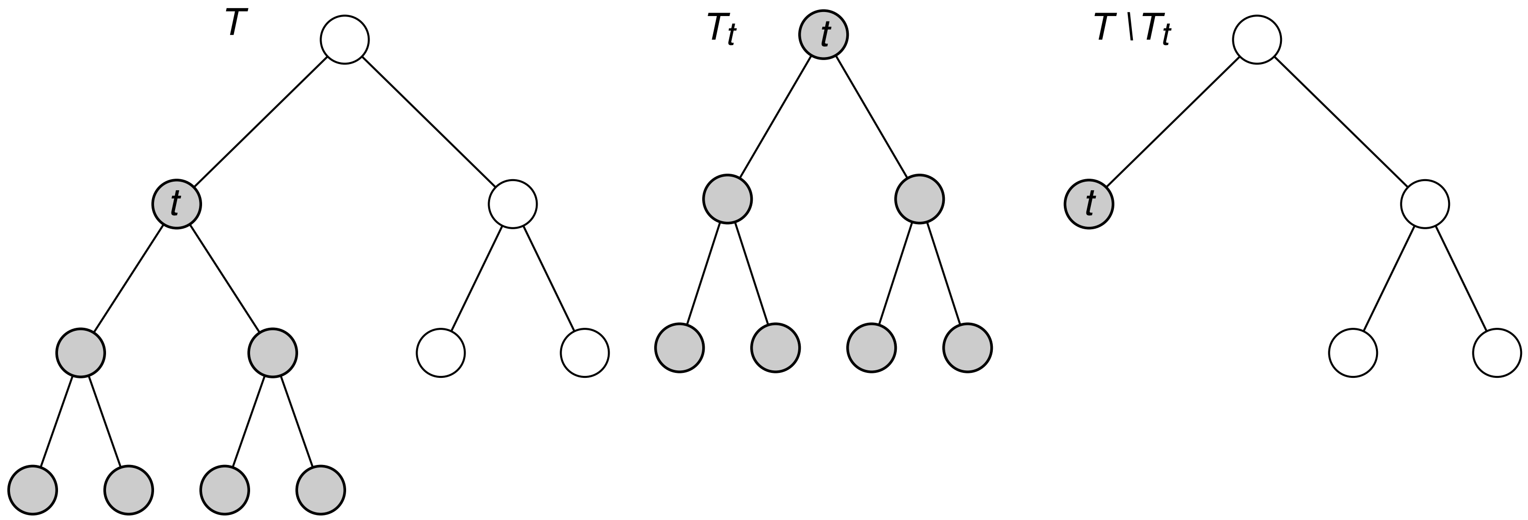
Decision Tree Pruning

Pruning (continued)

Definition 9 (Decision Tree Pruning)

Given a decision tree T and an inner (non-root, non-leaf) node t. Then pruning of T wrt. t is the deletion of all successor nodes of t in T. The pruned tree is denoted as $T \setminus T_t$. The node t becomes a leaf node in $T \setminus T_t$.

Illustration:



Decision Tree Pruning

Pruning (continued)

Definition 10 (Pruning-Induced Ordering)

Let T' and T be two decision trees. Then $T' \preceq T$ denotes the fact that T' is the result of a (possibly repeated) pruning applied to T. The relation $\preceq$ forms a partial ordering on the set of all trees.

Decision Tree Pruning

Pruning (continued)

Definition 10 (Pruning-Induced Ordering)
Let T' and T be two decision trees. Then $T' \preceq T$ denotes the fact that T' is the result of a (possibly repeated) pruning applied to T. The relation $\preceq$ forms a partial ordering on the set of all trees.

Problems when assessing pruning candidates:

- Pruned decision trees may not stand in the $\preceq$-relation.
- Starting with T_{max}, promising candidates may not result from locally optimum pruning decisions (greedy strategy).
- Its monotony disqualifies *Err*(T, D_{tr}) as an estimator for *Err*$^*(T)$. [Lemma 2]

Decision Tree Pruning

Pruning (continued)

Definition 10 (Pruning-Induced Ordering)
Let T' and T be two decision trees. Then $T' \preceq T$ denotes the fact that T' is the result of a (possibly repeated) pruning applied to T. The relation $\preceq$ forms a partial ordering on the set of all trees.

Problems when assessing pruning candidates:

- Pruned decision trees may not stand in the $\preceq$-relation.
- Starting with T_{max}, promising candidates may not result from locally optimum pruning decisions (greedy strategy).
- Its monotony disqualifies *Err*(T, D_{tr}) as an estimator for *Err*$^*(T)$. [Lemma 2]

Control pruning with validation set D_{vd}, where $D_{vd} \cap D_{tr} = \emptyset$, $D_{vd} \cap D_{ts} = \emptyset$:

1. $D_{tr} \subset D$ for decision tree construction.
2. $D_{vd} \subset D$ for overfitting analysis *during* pruning.
3. $D_{ts} \subset D$ for decision tree evaluation *after* pruning.

Decision Tree Pruning

Pruning: Reduced Error Pruning

Reduced error pruning for decision tree T and validation set D_{vd} :

1. Choose inner node t in T.
2. Tentative pruning of T wrt. t. Based on $D(t)$ assign class to t. [*DT-construct*]
3. Retract the pruning of Step 2 if *Err*$(T, D_{vd}) <$ *Err*$(T \setminus T_t, D_{vd})$.
4. Continue with Step 1 until all inner nodes of the *current tree* are tested.

Decision Tree Pruning

Pruning: Reduced Error Pruning

Reduced error pruning for decision tree T and validation set D_{vd} :

1. Choose inner node t in T.
2. Tentative pruning of T wrt. t. Based on $D(t)$ assign class to t. [*DT-construct*]
3. Retract the pruning of Step 2 if *Err*$(T, D_{vd}) <$ *Err*$(T \setminus T_t, D_{vd})$.
4. Continue with Step 1 until all inner nodes of the *current tree* are tested.

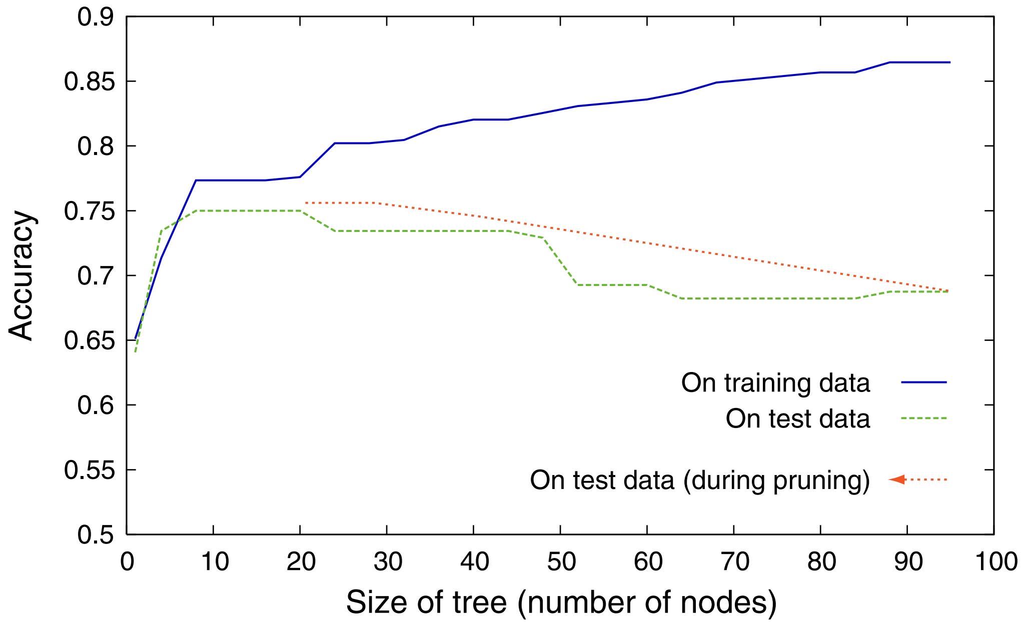
Problem:

If D is small, its partitioning into three sets for training, validation, and test will discard valuable information for decision tree construction.

Improvement: rule post pruning

Decision Tree Pruning

Pruning: Reduced Error Pruning (continued)



[Mitchell 1997]

Decision Tree Pruning

Extensions

- consideration of the misclassification cost introduced by a splitting
- “surrogate splittings” for insufficiently covered feature domains
- (linear) combinations of features
- regression tress