Machine Learning

B. Supervised Learning: Nonlinear Models
B.3. Decision Trees

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

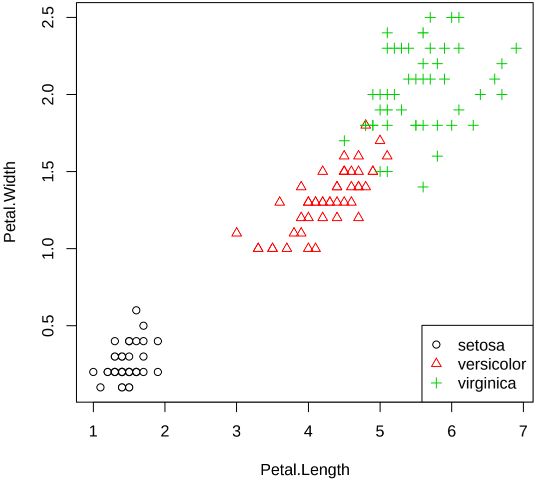
Syllabus

Fri. 25.10.	(1)	0. Introduction
		A. Supervised Learning: Linear Models & Fundamentals
Fri. 1.11.	(2)	A.1 Linear Regression
Fri. 8.11.	(3)	A.2 Linear Classification
Fri. 15.11.	(4)	A.3 Regularization
Fri. 22.11.	(5)	A.4 High-dimensional Data
		B. Supervised Learning: Nonlinear Models
Fri. 29.11.	(6)	B.1 Nearest-Neighbor Models
Fri. 6.12.	(7)	B.2 Neural Networks
Fri. 13.12.	(8)	B.3 Decision Trees
Fri. 20.12.	(9)	B.4 Support Vector Machines
		— *Christmas Break* —
Fri. 10.1.	(10)	B.5 A First Look at Bayesian and Markov Networks
		C. Unsupervised Learning
Fri. 17.1.	(11)	C.1 Clustering
Fri. 24.1.	(12)	C.2 Dimensionality Reduction
Fri. 31.1.	(13)	C.3 Frequent Pattern Mining
Fri. 7.2.	(14)	Q&A

Outline

Outline

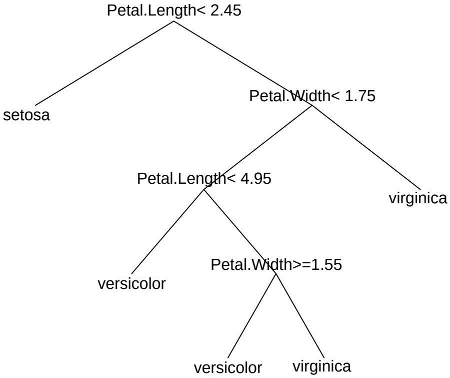
Decision Tree



Decision Tree

A **decision tree** is a tree that

1. at each **inner node** has a **splitting rule** that assigns instances uniquely to child nodes of the actual node, and
2. at each **leaf node** has a **prediction** (class label).

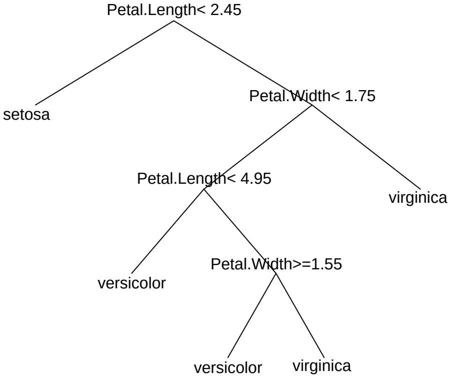


Note: The splitting rule is also called **decision rule**, the prediction the **decision**.

Using a Decision Tree

The class of a given case $x \in \mathcal{X}$ is predicted by

1. starting at the root node,
2. at each interior node
   - evaluate the splitting rule for x and
   - branch to the child node picked by the splitting rule, (default: left = "true", right = "false")
3. once a leaf node is reached,
   - predict the class assigned to that node as class of the case x.



Example:

x: Petal.Length = 6, Petal.Width = 1.6

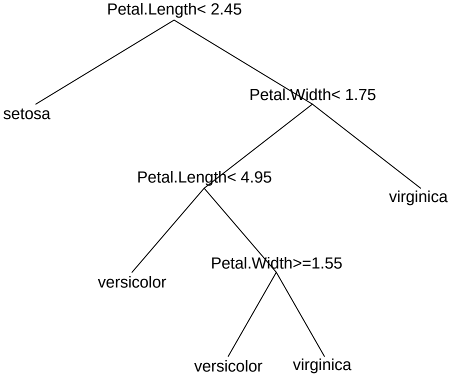
Decision Tree as Set of Rules

Each branch of a decision tree can be formulated as a single conjunctive rule

$$\textit{if } \textit{condition}_1(x) \textit{ and } \textit{condition}_2(x) \textit{ and } \ldots \textit{and } \textit{condition}_k(x),$$
$$\textit{then } y = \textit{class label at the leaf of the branch.}$$

A decision tree is equivalent to a set of such rules,
one for each branch.

Decision Tree as Set of Rules

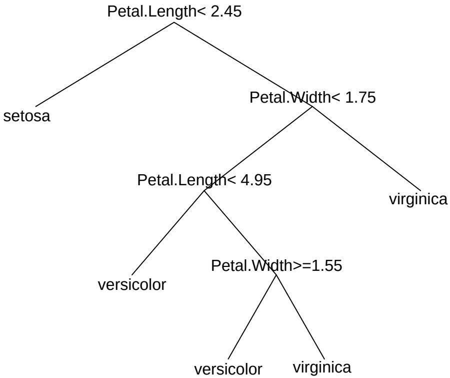


set of rules:

Petal.Length $<$ 2.45 $\rightarrow$ class=setosa
Petal.Length $\geq$ 2.45 and Petal.Width $<$ 1.75 and Petal.Length $<$ 4.95 $\rightarrow$ class=versicolor
Petal.Length $\geq$ 2.45 and Petal.Width $<$ 1.75 and Petal.Length $\geq$ 4.95 and Petal.Width $\geq$ 1.55 $\rightarrow$ class=versicolor
Petal.Length $\geq$ 2.45 and Petal.Width $<$ 1.75 and Petal.Length $\geq$ 4.95 and Petal.Width $<$ 1.55 $\rightarrow$ class=virginica

Petal.Length $\geq$ 2.45 and Petal.Width $\geq$ 1.75 $\rightarrow$ class=virginica

Decision Tree as Set of Rules

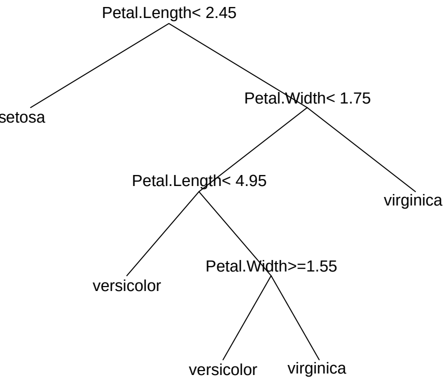


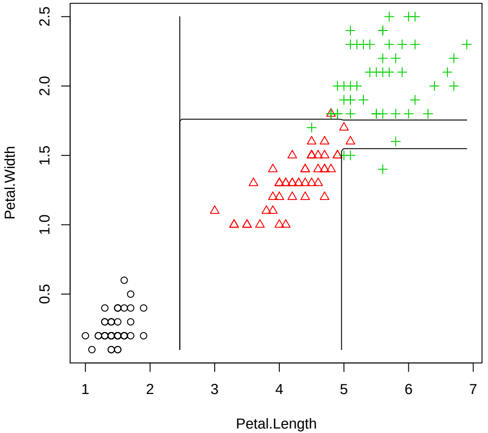
set of rules:

Petal.Length < 2.45		$\rightarrow$ class=setosa
Petal.Length $\in [2.45, 4.95[$	and Petal.Width < 1.75	$\rightarrow$ class=versicolor
Petal.Length ≥ 4.95	and Petal.Width $\in [1.55, 1.75[$	$\rightarrow$ class=versicolor
Petal.Length ≥ 4.95	and Petal.Width < 1.55	$\rightarrow$ class=virginica
Petal.Length ≥ 2.45	and Petal.Width ≥ 1.75	$\rightarrow$ class=virginica

Decision Boundaries

Decision boundaries are rectangular.





Regression Tree

A **regression tree** is a tree that

1. at each **inner node** has a **splitting rule** that assigns instances uniquely to child nodes of the actual node, and
2. at each **leaf node** has a **target value**.

Regression Tree & Probability Trees

A **regression tree** is a tree that

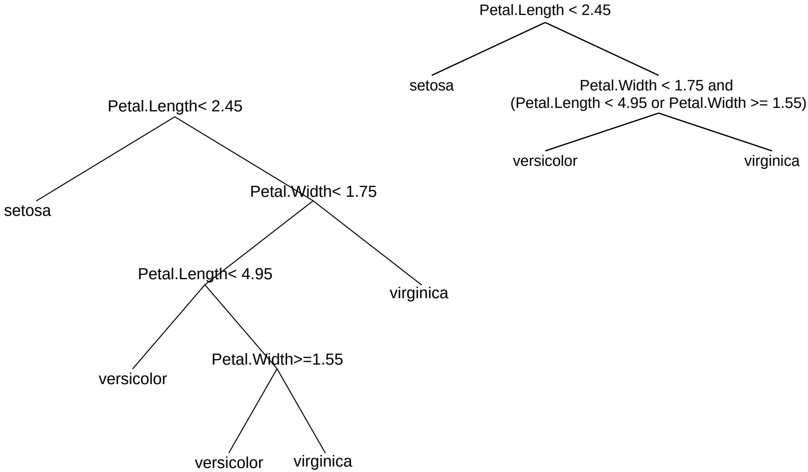
1. at each **inner node** has a **splitting rule** that assigns instances uniquely to child nodes of the actual node, and
2. at each **leaf node** has a **target value**.

A **probability tree** is a tree that

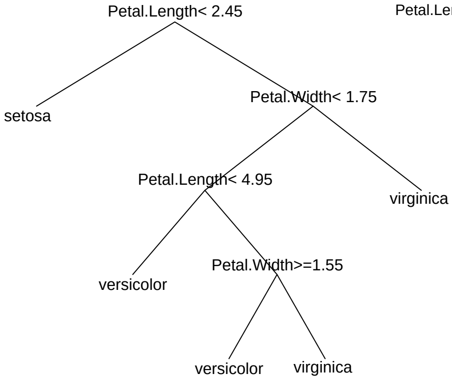
1. at each **inner node** has a **splitting rule** that assigns instances uniquely to child nodes of the actual node, and
2. at each **leaf node** has a **class probability distribution**.

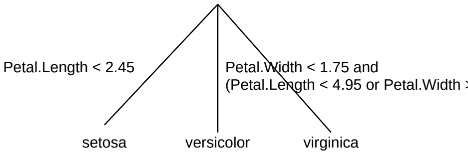
Outline

Which Decision Tree is Better?



Which Decision Tree is Better?





Simple Splits

To allow all kinds of splitting rules at the interior nodes (also called **splits**) does not make much sense. The very idea of decision trees is that

- the **splits** at each node are rather **simple** and
- more complex structures are captured by **chaining several simple decisions** in a tree structure.

Therefore, the set of possible splits is kept small by opposing several types of restrictions on possible splits:

- by restricting the **number of variables** used per split (univariate vs. multivariate decision tree),
- by restricting the **number of children** per node (binary vs. n-ary decision tree),
- by allowing only some **special types** of splits (e.g., complete splits, interval splits, etc.).

Types of Splits: Univarite vs. Multivariate

A split is called **univariate** if it uses only a single variable, otherwise **multivariate**.

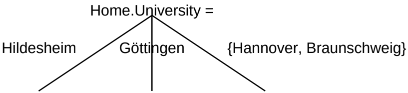
Example:
"Petal.Width < 1.75" is univariate,
"Petal.Width < 1.75 and Petal.Length < 4.95" is bivariate.

- Multivariate splits that are mere conjunctions of univariate splits better would be represented in the tree structure.
- But there are also multivariate splits than cannot be represented by a conjunction of univariate splits, e.g.,
  "Petal.Width / Petal.Length < 1"
  - can be represented by a univariate split on an additional predictor "Petal.Width / Petal.Length"

Types of Splits: *n*-ary

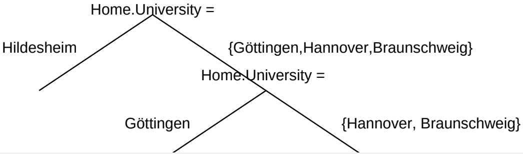
A split is called ***n*-ary** if it has n children.
(**Binary** is used for 2-ary, **ternary** for 3-ary.)

Example:
"Petal.Length < 1.75" is binary,



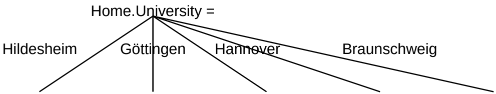
is ternary.

- All *n*-ary splits can be also represented as a tree of binary splits, e.g.,



Types of Splits: Complete Splits

A univariate split on a nominal variable is called **complete**
if each value is mapped to a child of its own,
i.e., the mapping between values and children is bijective.



- A complete split is n-ary
(where n is the number of different values for the nominal variable).

Types of Splits: Interval Splits

A univariate split on an at least ordinal variable is called **interval split** if for each child all the values assigned to that child are an interval.

Examples:
"Petal.Width < 1.75" is an interval split.

"(i) Petal.Width < 1.45,
(ii) Petal.Width ≥ 1.45 and Petal.Width < 1.75,
(iii) Petal.Width ≥ 1.75" also is an interval split.

"Petal.Width < 1.75 or Petal.Width ≥ 2.4" is not an interval split.

Types of Decision Trees

A decision tree is called

univariate,

n-**ary**,

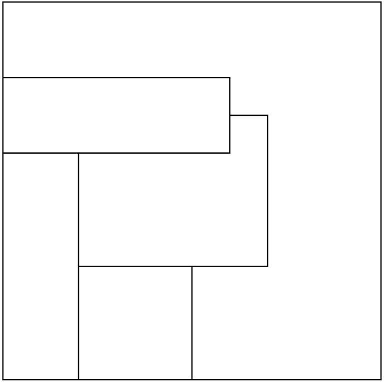
with complete splits or

with interval splits,

if **all** its splits have the corresponding property.

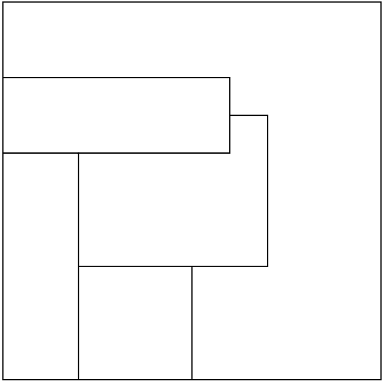
Binary Univariate Interval Splits

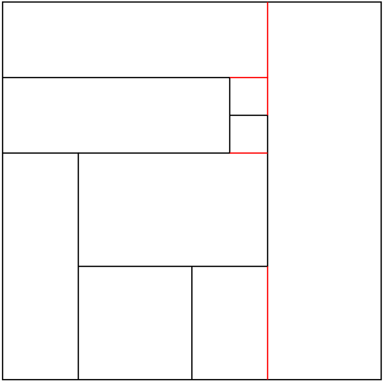
There are partitions (sets of rules)
that cannot be created by binary univariate splits.



Binary Univariate Interval Splits

There are partitions (sets of rules)
that cannot be created by binary univariate splits.





But all partitions can be refined
s.t. they can be created by binary univariate splits.

Outline

Learning Regression Trees (1/2)

Imagine, the **tree structure is already given**,
thus the partition

$$R_k, \quad k = 1, \ldots, K$$

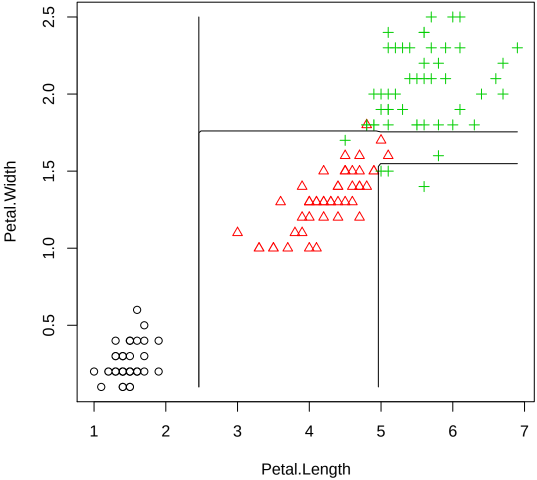
of the predictor space is already given.

Then the remaining problem is to **assign a predicted value**

$$\hat{y}_k, \quad k = 1, \ldots K$$

to each cell.

Learning Regression Trees (1/2) / Example



Learning Regression Trees (2/2)

Fit criteria such as the **smallest residual sum of squares** can be decomposed in partial criteria for cases falling in each cell:

$$\sum_{n=1}^{N}(y_n - \hat{y}(x_n))^2 = \sum_{k=1}^{K} \sum_{n=1, x_n \in R_k}^{N} (y_n - \hat{y}_k)^2$$

and this sum is minimal if the partial sum for each cell is minimal.

This is the same as fitting a constant model to the points in each cell R_k and thus the $\hat{y}_k$ with smallest RSS are just the **means**:

$$\hat{y}_k := \text{average}\{y_n \mid n = 1, \ldots, N; x_n \in R_k\}$$

Learning Decision Trees

The same argument shows that

- for a probability tree with given structure
  the **class probabilities with maximum likelihood** are just
  the **relative frequencies of the classes** of the points in that region:

$$\hat{p}(Y = y \mid x \in R_k) = \frac{|\{n \mid n = 1, \ldots, N; x_n \in R_k, y_n = y\}|}{|\{n \mid n = 1, \ldots, N; x_n \in R_k\}|}$$

- And for a decision tree with given structure, that
  the **class label with smallest misclassification rate** is just
  the **majority class label** of the points in that region:

$$\hat{y}(x \in R_k) = \arg\max_y |\{n \mid n = 1, \ldots, N; x_n \in R_k, y_n = y\}|$$

Possible Tree Structures

- Even when possible splits are restricted,
  - e.g., only binary univariate interval splits are allowed,

  then tree structures can be built that separate all cases in tiny cells that contain just a single point
  (if there are no points with same predictors).

- For such a very fine-grained partition,
  the fit criteria would be optimal
  (RSS=0, misclassification rate=0, likelihood maximal).

- Thus, decision trees need some sort of regularization to make sense.

Regularization Methods

There are several simple regularization methods:

minimum number of points per cell:

require that each cell (i.e., each leaf node) covers a given minimum number of training points.

maximum number of cells:

limit the maximum number of cells of the partition (i.e., leaf nodes).

maximum depth:

limit the maximum depth of the tree.

The number of points per cell, the number of cells, etc. can be seen as a **hyperparameter** of the decision tree learning method.

Outline

Decision Tree Learning Problem

The decision tree learning problem could be described as follows:
Given a dataset

$$\mathcal{D}^{\text{train}} := \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

find a decision tree $\hat{y} : X \to Y$ that

- is binary, univariate, and with interval splits,
- contains at each leaf a given minimum number m of examples,
- and has minimal misclassification rate

$$\text{mr}(\hat{y}; \mathcal{D}^{\text{train}}) := \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}(y_n \neq \hat{y}(x_n))$$

among all those trees.

Decision Tree Learning Problem

The decision tree learning problem could be described as follows:
Given a dataset

$$\mathcal{D}^{\text{train}} := \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

find a decision tree $\hat{y} : X \to Y$ that

- is binary, univariate, and with interval splits,
- contains at each leaf a given minimum number m of examples,
- and has minimal misclassification rate

$$\text{mr}(\hat{y}; \mathcal{D}^{\text{train}}) := \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}(y_n \neq \hat{y}(x_n))$$

among all those trees.

Unfortunately, this problem is **not feasible** as
there are **too many tree structures** / partitions to check
and no suitable optimization algorithms to sift efficiently through them.

Greedy Search

Therefore, a greedy search is conducted that

- starting from the root
- builds the tree **recursively**
- by selecting the **locally optimal decision** in each step.
  - or alternatively, even just **some locally good** decision.

Greedy Search / Possible Splits (1/2)

At each node one tries **all possible splits**.

For an univariate binary tree with interval splits
at the actual node let there still be the data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

Then check for each predictor variable X with domain $\mathcal{X}$:

1. if X is a **nominal variable**: (with m levels)
   all $2^{m-1} - 1$ possible splits in two subsets $X_1 \dot{\cup} X_2$.

   E.g., for $\mathcal{X} = \{\text{Hi}, \text{Gö}, \text{H}\}$ the splits

$$\begin{array}{lll} \{\text{Hi}\} & \text{vs.} & \{\text{Gö}, \text{H}\} \\ \{\text{Hi}, \text{Gö}\} & \text{vs.} & \{\text{H}\} \\ \{\text{Hi}, \text{H}\} & \text{vs.} & \{\text{Gö}\} \end{array}$$

Greedy Search / Possible Splits (2/2)

2. if X is an **ordinal** or **interval-scaled variable**:

sort the x_n as

$$x'_1 < x'_2 < \ldots < x'_{n'}, \quad N' \leq N$$

and then test all $N' - 1$ possible splits at

$$\frac{x'_n + x'_{n+1}}{2}, \quad n = 1, \ldots, N' - 1$$

E.g.,

$$(x_1, x_2, \ldots, x_8) = (15, 10, 5, 15, 10, 10, 5, 5), \quad N = 8$$

are sorted as

$$x'_1 := 5 < x'_2 := 10 < x'_3 := 15, \quad N' = 3$$

and then split at 7.5 and 12.5.

Greedy Search / Original Fit Criterion

All possible splits – often called **candidate splits** – are assessed by a **quality criterion**.

For all kinds of trees the **original fit criterion** can be used, i.e.,

for regression trees:

the **residual sum of squares**.

for decision trees:

the **misclassification rate**.

for probability trees:

the **likelihood**.

The split that gives the best improvement is chosen.

Example

Artificial data about visitors of an online shop:

	x_1	x_2	x_3	y
	referrer	num.visits	duration	buyer
1	search engine	several	15	yes
2	search engine	once	10	yes
3	other	several	5	yes
4	ad	once	15	yes
5	ad	once	10	no
6	other	once	10	no
7	other	once	5	no
8	ad	once	5	no

Learn a decision tree that tries to predict if a visitor will buy.

Example / Root Split

Step 1 (root node): The root covers all 8 visitors.
There are the following splits:

		buyer		
variable	values	yes	no	errors
referrer	{s}	2	0	2
	{a, o}	2	4	
referrer	{s, a}	3	2	3
	{o}	1	2	
referrer	{s, o}	3	2	3
	{a}	1	2	
num.visits	once	2	4	2
	several	2	0	
duration	<7.5	1	2	3
	≥ 7.5	3	2	
duration	< 12.5	2	4	2
	≥ 12.5	2	0	

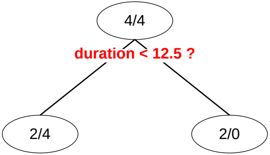
Example / Root Split

The splits

- referrer = search engine ?
- num.visits = once ?
- duration $<$ 12.5 ?

are locally optimal at the root.

We choose "duration $<$ 12.5":



Note: See backup slides after the end for more examples.

Decision Tree Learning Algorithm

expand-decision-tree(node T, training data $\mathcal{D}^{\text{train}}$) :
  if stopping−criterion ($\mathcal{D}^{\text{train}}$):
    $T.\text{class} := \arg\max_{y'} |\{(x,y) \in \mathcal{D}^{\text{train}} \mid y = y'\}|$
    return
  $s := \arg\max_{\text{split } s} \text{quality-criterion}(s)$
  if s does not improve:
    $T.\text{class} = \arg\max_{y'} |\{(x,y) \in \mathcal{D}^{\text{train}} \mid y = y'\}|$
    return
  $T.\text{split} := s$
  for $z \in \text{Im}(s)$:
    create new node T'
    $T.\text{child}[z] := T'$
    expand−decision−tree($T', \{(x,y) \in \mathcal{D}^{\text{train}} \mid s(x) = z\}$)

learn-decision-tree(training data $\mathcal{D}^{\text{train}}$) :
  create new node T
  expand−decision−tree($T, \mathcal{D}^{\text{train}}$)
  return T

Decision Tree Learning Algorithm / Remarks (1/2)

- tree nodes T with 3 attributes:
  - split and child (inner node)
  - class (leaf node)

- stopping-criterion(X):
  e.g.,
  - all cases in X belong to the same class,
  - all cases in X have the same predictor values (for all variables),
  - there are less than the minimum number of cases per node to split.

- split s:
  all possible splits, e.g., all binary univariate interval splits.

- quality-criterion(s):
  e.g., misclassification rate in X after the split (i.e., if in each child node suggested by the split the majority class is predicted).

Decision Tree Learning Algorithm / Remarks (2/2)

- s does not improve:
  e.g., if the misclassification rate is the same as in the actual node (without the split s).

- $\text{Im}(s)$:
  all the possible outcomes of the split,
  e.g., $\{ 0, 1 \}$ for a binary split.

- $T.\text{child}[z] := T'$:
  keep an array that maps all the possible outcomes of the split to the corresponding child node.

Decision Tree Prediction Algorithm

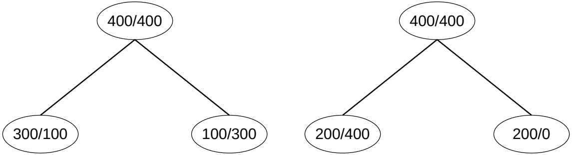
predict-decision-tree(node T, instance $x \in \mathbb{R}^M$) :
    if T.split $\neq \emptyset$:
        $z := T$.split(x)
        $T' := T$.child$[z]$
        return predict-decision-tree(T', x)
    return T.class

Outline

Why Misclassification Rate is a Bad Split Quality Criterion

Although it is possible to use misclassification rate as quality criterion, it usually is not a good idea.

Imagine a dataset with a binary target variable (zero/one) and 400 cases per class (400/400).
Assume there are two splits:



Both have 200 errors / misclassification rate 0.25.

But the right split may be preferred as it contains a pure node.

Split Contingency Tables

The effects of a split on training data can be described by a **contingency table** $(C_{j,k})_{j \in J, k \in K}$, i.e., a matrix

- with **rows** indexed by the different child nodes $j \in J$,
- with **columns** indexed by the different target classes $k \in K$,
- and cells $C_{j,k}$ containing the number of points in class k that the split assigns to child j:

$$C_{j,k} := |\{(x, y) \in \mathcal{D}^{\text{train}} \mid s(x) = j \text{ and } y = k\}|$$

Entropy

Let

$$\Delta_N := \{(p_1, p_2, \ldots, p_N) \in [0,1]^N \mid \sum_n p_n = 1\}$$

be the set of multinomial probability distributions on the values $1, \ldots, N$.

An **entropy function** $q : \Delta_N \rightarrow \mathbb{R}_0^+$ has the properties

- q is maximal for uniform $p = (\frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N})$.
- q is 0 iff p is deterministic
  (one of the $p_n = 1$ and all the others equal 0).

An entropy quantifies the unorder of an distribution.

Entropy / Examples

Cross-Entropy / **Deviance**:

$$H(p_1, \ldots, p_N) := -\sum_{n=1}^{N} p_n \log(p_n)$$

Shannons Entropy:

$$H(p_1, \ldots, p_N) := -\sum_{i=1}^{n} p_n \log_2(p_n)$$

Quadratic Entropy:

$$H(p_1, \ldots, p_N) := \sum_{i=1}^{n} p_n(1 - p_n) = 1 - \sum_{n=1}^{N} p_n^2$$

Entropy measures can be extended to $\mathbb{R}_0^+$ via

$$q(x_1, \ldots, x_N) := q\left(\frac{x_1}{\sum_n x_n}, \frac{x_2}{\sum_n x_n}, \ldots, \frac{x_N}{\sum_n x_n}\right)$$

Entropy for Contingency Tables

For a contingency table $C_{j,k}$ we use the following abbreviations:

$$C_{j,.} := \sum_{k \in K} C_{j,k} \qquad \text{sum of row } j$$

$$C_{.,k} := \sum_{j \in J} C_{j,k} \qquad \text{sum of column } k$$

$$C_{.,.} := \sum_{j \in J} \sum_{k \in K} C_{j,k} \qquad \text{sum of matrix}$$

and define the following entropies:

row entropy:

$$H_J(C) := H(C_{j,.} \mid j \in J)$$

column entropy:

$$H_K(C) := H(C_{.,k} \mid k \in K)$$

conditional column entropy:

$$H_{K|J}(C) := \sum_{j \in J} \frac{C_{j,.}}{C_{.,.}} H(C_{j,k} \mid k \in K)$$

Entropy for Contingency Tables

Suitable split quality criteria are

entropy gain:

$$HG(C) := H_K(C) - H_{K|J}(C)$$

entropy gain ratio:

$$HG(C) := \frac{H_K(C) - H_{K|J}(C)}{H_J(C)}$$

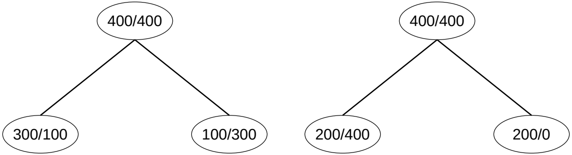
Shannon entropy gain is also called **information gain**:

$$\mathrm{IG}(C) := -\sum_k \frac{C_{.,k}}{C_{.,.}} \log_2 \frac{C_{.,k}}{C_{.,.}} + \sum_j \frac{C_{j,.}}{C_{.,.}} \sum_k \frac{C_{j,k}}{C_{j,.}} \log_2 \frac{C_{j,k}}{C_{j,.}}$$

Quadratic entropy gain is also called **Gini index**:

$$\mathrm{Gini}(C) := -\sum_k \left(\frac{C_{.,k}}{C_{.,.}}\right)^2 + \underbrace{\sum_j \frac{C_{j,.}}{C_{.,.}} \sum_k \left(\frac{C_{j,k}}{C_{j,.}}\right)^2}_{=:\text{gini-impurity}(C)}$$

Entropy Measures as Split Quality Criterion



Both have 200 errors / misclassification rate 0.25.

But the right split may be preferred as it contains a pure node.

Gini-Impurity

$$= \frac{1}{2}((\frac{3}{4})^2 + (\frac{1}{4})^2) + \frac{1}{2}((\frac{3}{4})^2 + (\frac{1}{4})^2) = 0.625$$

Gini-Impurity

$$= \frac{3}{4}((\frac{1}{3})^2 + (\frac{2}{3})^2) + \frac{1}{4}(1^2 + 0^2) \approx 0.667$$

Popular Decision Tree Configurations

name	ChAID	CART	ID3	C4.5
author	Kass 1980	Breiman et al. 1984	Quinlan 1986	Quinlan 1993
selection measure	χ^2	Gini index, twoing index	information gain	information gain ratio
splits	all	binary nominal, binary quantitative, binary bivariate quantitative	complete	complete, binary nominal, binary quantitative
stopping criterion	χ^2 independence test	minimum number of cases/node	χ^2 independence test	lower bound on selection measure
pruning technique	none	error complexity pruning	pessimistic error pruning	pessimistic error pruning, error based pruning

Summary

- Decision trees are trees having
  - **splitting rules** at the **inner nodes** and
  - **predictions** (decisions) at the **leaves**.
- Decision trees use only **simple splits**
  - univariate, binary, **interval splits**.
- Decision trees have to be regularized by **constraining their structure**
  - minimum number of examples at inner nodes, maximum depth, etc.
- Decision trees are learned by greedy **recursive partitioning**.
  - As **split quality criteria** entropy measures are used
    - **Gini index**, **information gain ratio**, etc.
- Outlook (see lecture Machine Learning 2):
  - Sometimes **pruning** is used to make the search less greedy.
  - Decision trees use **surrogate splits** to cope with missing data.
  - Decision trees can be boosted yielding very competitive models (**random forests**, **gradient boosted decision trees**).

Further Readings

- [Hastie et al., 2005, chapter 9.2+6+7], [Murphy, 2012, chapter 16.1–2], [James et al., 2013, chapter 8.1+3].

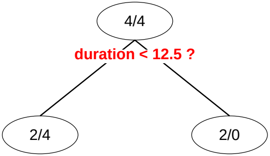
References

Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, volume 27. Springer, 2005.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, 2013.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Example / Node 2 Split



The right node is pure and thus a leaf.

Step 2 (node 2): The left node (called "node 2") covers the following cases:

	referrer	num.visits	duration	buyer
2	search engine	once	10	yes
3	other	several	5	yes
5	ad	once	10	no
6	other	once	10	no
7	other	once	5	no
8	ad	once	5	no

Example / Node 2 Split

At node 2 are the following splits:

variable	values	buyer yes	buyer no	errors
referrer	{s}	1	0	1
	{a, o}	1	4	
referrer	{s, a}	1	2	2
	{o}	1	2	
referrer	{s, o}	2	2	2
	{a}	0	2	
num.visits	once	1	4	1
	several	1	0	
duration	<7.5	1	2	2
	≥ 7.5	1	2	

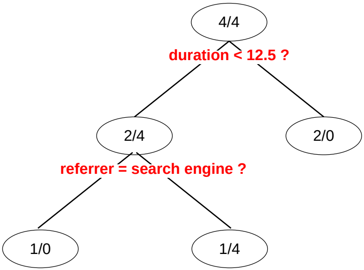
Again, the splits

- referrer = search engine ?
- num.visits = once ?

are locally optimal at node 2.

Example / Node 5 Split

We choose the split "referrer = search engine":



The left node is pure and thus a leaf.

The right node (called "node 5") allows further splits.

Example / Node 5 Split

Step 3 (node 5): The right node (called "node 5") covers the following cases:

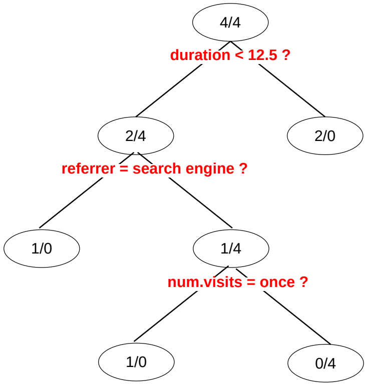
	referrer	num.visits	duration	buyer
3	other	several	5	yes
5	ad	once	10	no
6	other	once	10	no
7	other	once	5	no
8	ad	once	5	no

It allows the following splits:

		buyer		
variable	values	yes	no	errors
referrer	{a}	0	2	1
	{o}	1	2	
num.visits	once	1	0	0
	several	0	4	
duration	<7.5	1	2	1
	≥ 7.5	0	2	

Example / Node 5 Split

The split "num.visits = once" is locally optimal.



Both child nodes are pure thus leaf nodes.
The algorithm stops.