Algorithmisches Lernen/Machine Learning

Part 1: Stefan Wermter

- Introduction
- Connectionist Learning (e.g. Neural Networks)
- Decision-Trees, Genetic Algorithms

Part 2: Norman Hendrich

- Support-Vector Machines
- Learning of Symbolic Structures
- Bayesian Learning
- Dimensionality Reduction

Part 3: Jianwei Zhang

- Function approximation
- Reinforcement Learning
- Applications in Robotics

Introduction

- Machine Learning
- Designing a ML System
- Issues in Machine Learning
- Learning Paradigms
- Development of a Learning System

Machine Learning

A system is said to *learn* ...

- ... from experience E
- ... with respect to some class of tasks T ...
- ... and a performance measure P, ...

... if ...

- ... its performance at tasks in T, ...
- ... as measured by P, ...
- ... improves with experience E.

Machine Learning

e.g. learning to play checkers

- task T: playing checkers
- performance measure P: share of games won
- experience E: playing practice games against itself or others

e.g. handwriting recognition

- task T: locating and identifying handwritten words/characters in images
- performance measure P: share of correctly identified words/characters
- experience E: a database of handwritten words/characters with given classifications

Machine Learning

e.g. robot driving control

- task T: driving in a public traffic space
- performance measure P: average distance travelled without a driving error
- experience E: human samples of driving actions in various situations

Designing a learning system

- target function: produces the desired behaviour according to the given task
- learning: find a hypothesis for a target function that fits the observed data best
- hypotheses are taken from among a class of hypothesis representations
- hypothesis representations: decision trees, neural networks, Bayesian networks ...

1. chosing the training experience
   - direct vs. indirect information (credit assignment problem)
   - who selects the samples? (active learning)
   - representative vs. spurious samples

Designing a learning system

2. choosing the target function
   - what to do next / what category to choose
     $choose\_move : b_i \rightarrow b_{i+1}$
     $b_i \in B$: possible board states
   - usually mapped to a numerical evaluation $V : B \rightarrow \mathcal{R}$
   - decision rule: maximize/minimize the target function

$$b' = \arg\max_{b_i} V(b_i) \qquad \text{with } b_i = succ(b_{i-1})$$

   - $\rightarrow$ learning as function approximation
     from examples $\{\langle b, V_{train(b)} \rangle\}$

Designing a learning system

2. choosing the target function (cont.)
   - e.g.
     - winning state: $V(b) = 100$
     - loosing state: $V(b) = -100$
     - start state: $V(b) = 0$
     - non-final state:

$$V(b) = V(b') \text{ with } b' = \arg\max_{final(b_i)} V(b_i)$$

Designing a learning system

3. choosing a representation of the target function
   - e.g. a linear function $\widehat{V} = w_0 + \sum_{i=1}^{n} w_i x_i$
   - x_i: features of the problem space, observations
     e.g.
     - x_1/x_2: number of black/white pieces
     - x_3/x_4: number of black/white kings
     - x_5/x_6: number of black/white pieces
       threatened by white/black
   - $\rightarrow$ learning as parameter estimation

Designing a learning system

4. if only indirect information available:
   - estimating training values for V
   - credit assignment: which board state is good/bad
     $\rightarrow$ reinforcement learning
   - e.g.
     - $V(b)$ is known only for the final states
       $\rightarrow$ propagating values backwards
     - $V_{train}(b) \leftarrow \widehat{V}(succ(succ(b)))$
       with $\widehat{V}$: current approximation of V

Designing a learning system

5. choosing a function approximation algorithm
   - e.g. minimize the error between the training examples and the predicted value of the current target function

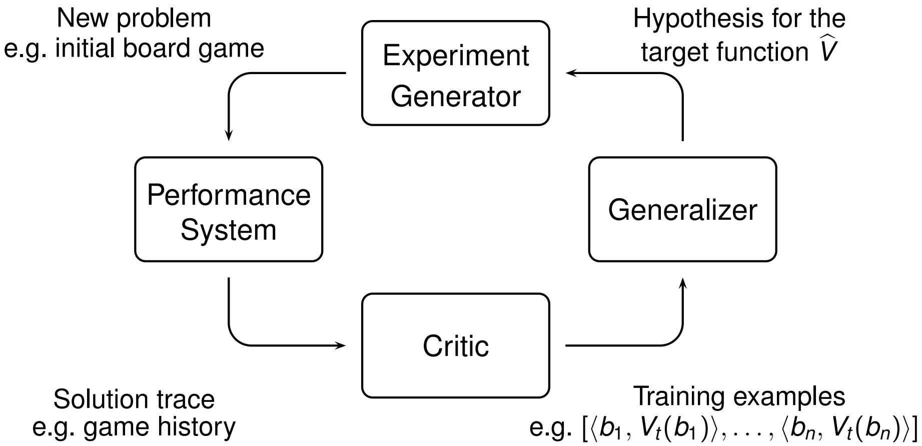
$$E = \sum_{\forall b \in \{\langle b, V_{train(b)} \rangle\}} (V_{train(b)} - \widehat{V}(b))^2$$

$\rightarrow$ linear regression

   - alternatives: neural networks, large margin learning, expectation maximization, ...

Designing a learning system

6. producing an overall design



Designing a learning system

- practical applications are complex
- usually include many embedded learning tasks with mutual dependencies
- e.g. computer vision
    - recognition of visual primitives: lines, points, gradients, ...
    - segmenting into regions
    - recognition of objects
    - recognition of composites
    - ...

Designing a learning system

- e.g. speech recognition: sample sequences $\mapsto$ word sequences
    - choosing the basic acoustic units
    - clustering acoustic models
    - selecting the model topology
    - training of acoustic parameters
    - selection of pronunciation variants
    - training a language model
    - clustering of words
    - ...

Issues in Machine Learning

- algorithms for learning the target function
  - Which different kinds of approaches are there?
  - Under which conditions do they converge?
  - Which is best under which conditions?
- training data
  - How much is sufficient?
  - Are there general bounds for the confidence, depending on the learning task and the training data available?
  - In which order to present the training examples best?
  - Does this order affect the complexity of the learning task?

Issues in Machine Learning

- prior knowledge
  - Can prior knowledge help to guide the learning process? How?
  - Is it useful even if only approximative?
- target function representation
  - What's the best way to reduce the learning task to a function approximation problem?
  - Can this process be automated?
  - Can the representation be changed by learning system itself?

Learning Paradigms

- the value of the target function is given for each individual training sample

  $\longrightarrow$ supervised learning
- no value of the target function is given

  $\longrightarrow$ unsupervised learning
- the value of the target function is given only for a complete *sequence* of elementary decision problems

  $\longrightarrow$ reinforcement learning

Target Functions

- discrete and has (a finite number of) a priori given values
  $\rightarrow$ classification
- the target function is discrete but the values are not given in advance
  $\rightarrow$ clustering
- the target function is continuous
  $\rightarrow$ function approximation

Target Functions

- learning creates a symbolic structure (decision tree, rule set, etc.)
  $\rightarrow$ symbolic learning
- learning estimates the parameters of a stochastic model
  $\rightarrow$ probabilistic learning
- learning adjusts the parameters of a system of (linear) functions
  $\rightarrow$ discriminative learning
- learning adjusts the weights of a system of nonlinear decision units
  $\rightarrow$ connectionist learning

Learning System Development

- available data separated into
  - training data for learning
  - validation/development data for parameter tuning
  - test data for evaluation
- data preprocessing
  - number of values reduction
  - dimensionality reduction
  - orthogonalization

Learning System Development

- development cycle
  1. select a class of target functions
  2. adjust possible free parameters
  3. train the model
  4. evaluate the model
  5. continue with (2) until no further improvement can be achieved
  6. continue with (1) until no further improvement can be achieved

Machine Learning as Optimization

- selecting the optimal class of target functions ...
- adjusting the parameters ...
- determining the best target function ...
- ... are optimization problems

- no analytical solution available
- full search infeasible
- heuristic procedures and intuition required

Learning Symbolic Structures

- Concept Learning
- Decision Trees
- Ensemble-based Methods
- Rule Induction
- Inductive Logic Programming
- Association Rules
- Transformation Rules

Concept Learning

- inferring a boolean function from consistent categorical data
  - features (attributes, predictors) with discrete values
  - boolean target function

No	Sky	Air	Humidity	Wind	Water	Forecast	Enjoy sports
1	sunny	warm	normal	strong	warm	no change	yes
2	sunny	warm	high	strong	warm	no change	yes
3	rainy	cold	high	strong	warm	change	no
4	sunny	warm	high	strong	cool	change	yes

- target representation: underspecified descriptions
  - ?: any value acceptable for the attribute
  - single value specified (e.g. ”warm”)
  - $\emptyset$: no value acceptable

e.g. (?, cold, high, ?, ?, ?)

Concept Learning

Given:

- observations X: possible days described by the attributes
  - sky(X) $\in$ {sunny, cloudy, rainy}
  - air(X) $\in$ {warm, cold}
  - humidity(X) $\in$ {normal, high}
  - wind(X) $\in$ {strong, weak}
  - water(X) $\in$ {warm, cold}
  - forecast(X) $\in$ {change, no change}
- hypotheses H: conjunction of constraints on the attributes: ?, value, $\emptyset$
- target concept c: enjoy sport: $X \mapsto \{0, 1\}$
- training data D: positive and negative examples of the target function

Determine:

- hypothesis $h \in H$ such that $h(x) = c(x)\ .\ \forall x \in X$

Concept Learning

Inductive learning hypothesis: Any hypothesis that approximates the target function well enough over a sufficiently large set of training examples will also approximate the target function well over unseen data.

Concept Learning

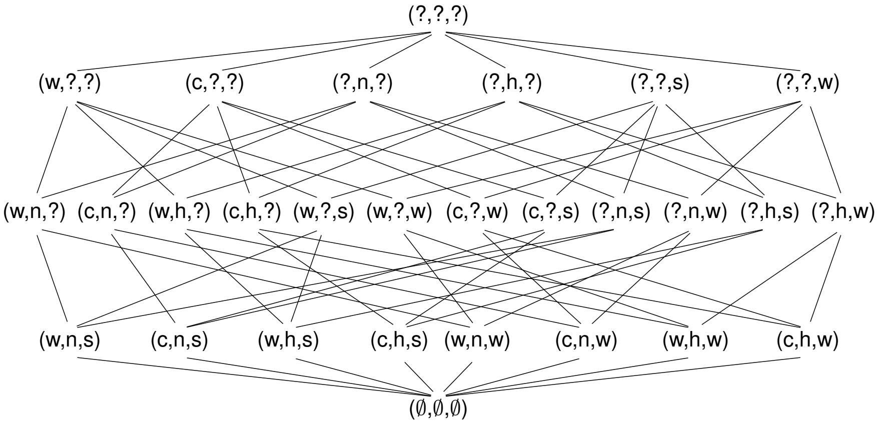
- any underspecified representation is a generalized description of a set of instances

description	stands for examples
(sunny, warm, ?, strong, warm, no change)	$\{x_1, x_2\}$
(?, ?, high, strong, ?, change)	$\{x_3, x_4\}$
(sunny, warm, $\emptyset$, strong, warm, no change)	$\emptyset$

Example	Sky	Air	Humidity	Wind	Water	Forecast	Enjoy sports
x_1	sunny	warm	normal	strong	warm	no change	yes
x_2	sunny	warm	high	strong	warm	no change	yes
x_3	rainy	cold	high	strong	warm	change	no
x_4	sunny	warm	high	strong	cool	change	yes

Concept Learning

- descriptions can be ordered according to their specificity
- subsumtion: h_i subsumes / is more general than h_j ($h_i \geq_g h_j$) iff $\forall x \in X \,.\, h_j(x) = 1 \rightarrow h_i(x) = 1$
- descriptions form a partial order



FIND-S

- finding a maximally specific hypothesis

1. Intitialize h to the most specific hypothesis in H
2. For each positive training example x
    - For each attribute constraint in a_i in h

      If the constraint a_i is satisfied by h
      Then do nothing
      Else replace a_i by the next more general constraint that is satisfied by x
3. Output hypothesis h

FIND-S

	h	$(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)$
x_1		(sunny, warm, normal, strong, warm, no change)
	h	(sunny, warm, normal, strong, warm, no change)
x_2		(sunny, warm, high, strong, warm, no change)
	h	(sunny, warm, ?, strong, warm, no change)
x_3		negative example
x_4		(sunny, warm, high, strong, cool, change)
	h	(sunny, warm, ?, strong, ?, ?)

FIND-S

- makes no use of negative examples
- finds a hypothesis which is consistent with the training examples ...
- ... but cannot guarantee that this is the only hypothesis that is consistent with the data
- the most specific hypothesis is not necessarily the most plausible one
- produces implausible results if the training data are inconsistent
- cannot even detect an inconsistency
- if there are several most specific hypotheses (e.g. in case of disjunctive target functions), only one of them is found

CANDIDATE-ELIMINATION

- most specific hypotheses $\rightarrow$ set of all consistent hypotheses
- satisfiability $\rightarrow$ consistency
- a hypothesis h is consistent with a set of training examples D iff $\forall \langle x, c(x) \rangle \in D \,.\, h(x) = c(x)$
- version space: set of all hypotheses from H which consistent with D
- represented by its most (G) and least (S) general members (upper and lower bound in the lattice)

$$\begin{tikzcd}[column sep=small]
 & G\colon \boxed{\{(sunny,?,?,?,?,?),(?,warm,?,?,?,?)\}} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
(sunny,?,?,strong,?,?) \arrow[dr, dash] & (sunny,warm,?,?,?,?) \arrow[d, dash] & (?,warm,?,strong,?,?) \arrow[dl, dash] \\
 & S\colon \boxed{\{(sunny,warm,?,strong,?,?)\}} &
\end{tikzcd}$$

CANDIDATE-ELIMINATION

```
Initialize $G$ with the maximally general hypotheses in $H$
Initialize $S$ with the most specific hypotheses in $H$
For each training example $d$ do
    • If $d$ is a positive example
        • Remove from $G$ any hypothesis which is inconsistent with $d$
        • For each hypothesis $s$ in $S$ that is not consistent with $d$
            • Remove $s$ from $S$
            • Add to $S$ all minimal generalizations $h$ of $s$ such that
              $h$ is consistent with $d$ and
              some member of $G$ is more general than $h$
    • If $d$ is a negative example
        • Remove from $S$ any hypothesis which is inconsistent with $d$
        • For each hypothesis $g$ in $G$ that is not consistent with $d$
            • Remove $g$ from $G$
            • Add to $G$ all minimal specializations $h$ of $g$ such that
              $h$ is consistent with $d$ and
              some member of $S$ is less general than $h$
```

CANDIDATE-ELIMINATION

1. initial state

$$G: \boxed{\{(?,?,?,?,?,?)\}}$$

$$S: \boxed{\{(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)\}}$$

2. training example x_1: $\langle$ (sunny,warm,normal,strong,warm,no_change), yes $\rangle$

$$G: \boxed{\{(?,?,?,?,?,?)\}}$$

$$\begin{tikzcd} S': \boxed{\{(\text{sunny,warm,normal,strong,warm,no\_change})\}} \\ S: \boxed{\{(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)\}} \arrow[u] \end{tikzcd}$$

CANDIDATE-ELIMINATION

3. training example x_2: $\langle$ (sunny,warm,high,strong,warm,no_change), yes $\rangle$

$$\begin{tikzcd}
G\text{:} \ \boxed{\{(?,?,?,?,?,?)\}} \\
\\
S'\text{:} \ \boxed{\{(\text{sunny,warm,?,strong,warm,no\_change})\}} \\
S\text{:} \ \boxed{\{(\text{sunny,warm,normal,strong,warm,no\_change})\}} \arrow[u]
\end{tikzcd}$$

4. training example x_3: $\langle$ (rainy,cold,high,strong,warm,change), no $\rangle$

$$\begin{tikzcd}
G\text{:} \ \boxed{\{(?,?,?,?,?,?)\}} \arrow[d] \arrow[d, shift right=20] \arrow[d, shift left=20] \\
G'\text{:} \ \boxed{\{(\text{sunny,?,?,?,?,?}),(\text{?,warm,?,?,?,?}),\ (\text{?,?,?,?,?,no\_change})\}} \\
\\
S\text{:} \ \boxed{\{(\text{sunny,warm,?,strong,warm,no\_change})\}}
\end{tikzcd}$$

CANDIDATE-ELIMINATION

... finds the target concept

- if it exists in H
- if it can be described by a conjunction of constraints

- independent of the order of training examples
- inconsistencies in the training data can be detected
  - upper and lower bound converge
  - version space becomes empty

Candidate-Elimination

- remaining ambiguity of results can be monitored
- effective queries can be generated (active learning):
  - Try to find a combination of attribute values which satisfies half of the hypotheses in the version space
- training does not need to converge to a unique hypothesis
  - if all hypotheses in the version space agree, classification can be considered reliable
  - degree of agreement can be used as an estimate of confidence

Candidate-Elimination

- application to consistency-based diagnosis
  - attributes: status of the components of a device
  - observations: measurements on the device
  - diagnosis: find a combination of attribute values which is consistent with the observations
  - partial observations can be used
  - follow-up measurements can be suggested

Inductive Bias

- version space learning is biased by only allowing conjunctive representations
- not all possible descriptions can be represented

Example	Sky	Air	Humidity	Wind	Water	Forecast	Enjoy sports
x_5	sunny	warm	normal	strong	cool	change	yes
x_6	cloudy	warm	normal	strong	cool	change	yes
x_7	rainy	warm	normal	strong	cool	change	no

- no consistent conjunctive representation for $x_5 \ldots x_7$
- disjunctive representations allow to express the concept ...
- ... but the CANDIDATE-ELIMINATION algorithm fails to generalize
- result is a simple enumeration of possibilities
  - *S* is a disjunctive combination of positive training examples
  - *G* is a negated disjunction of negative training examples

Inductive Bias

A learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.

- a learner L
- is trained on data $D_c\{\langle x, c(x)\rangle\}$
- and can be applied to unseen data after learning: $L(x_i, D_c)$
- the learner follows inductively from the data:

$$(D_c \wedge x_i) \succ L(x_i, D_c)$$

- the inductive inference is not necessarily correct

Inductive Bias

- to make the inference sound, additional assumptions have to be added
- the inductive bias is a minimal set of additional assumptions necessary to make this inference deductively sound

$$(B \wedge D_c \wedge x_i \;\vdash\; L(x_i, D_c)).\forall x_i \in X$$

- inductive bias of the CANDIDATE-ELIMINATION algorithm: $c \in H$

Inductive Bias

- Learning algorithms can be ranked according to the strength of their inductive bias
- ROTE-LEARNER
    - no inductive bias
    - simply stores the examples
    - rejection if instance not found
- CANDIDATE-ELIMINATION
    - new instances are classified only if all members of the current version space agree
    - rejection otherwise
    - stronger inductive bias: target concept can be represented in its hypothesis space
    - can classify more instances

Inductive Bias

- FIND-S
  - uses the most specific hypothesis consistent with the training examples
  - can classify all observations
  - additional bias: all instances are classified as negative, if not subsumed by the most specific hypothesis (closed world assumption)
- the stronger the inductive bias ...
  - the more inductive leaps are made
  - the more unseen instances can be classified
  - the more risky the decision becomes

Concept Learning

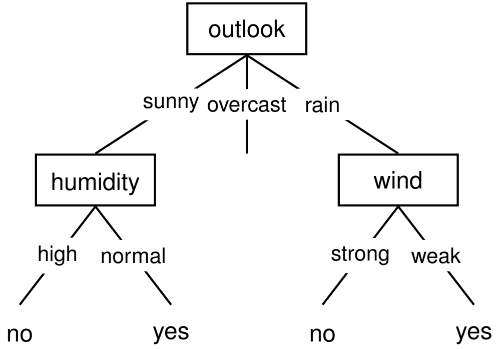
- extensions
  - accomodating noisy data
  - merging of version spaces trained to represent different kinds of constraints
  - learning disjuctive concepts (from noisy data)

Learning Symbolic Structures

- Concept Learning
- Decision Trees
- Ensemble-based Methods
- Rule Induction
- Inductive Logic Programming
- Association Rules
- Transformation rules

Decision Trees

- Representation
  - internal nodes: attributes
  - edges: attribute values
  - leaf nodes: values of the target function



Decision Trees

- disjunction of conjunction of constraints on the attribute values

  (outlook=sunny $\wedge$ humidity=normal)
  $\vee$ (outlook=overcast)
  $\vee$ (outlook=rain $\wedge$ wind=weak)
- well suited for classification problems where
  - instances can be described by attribute-value pairs
  - values are discrete or continuous
  - target function has discrete values
  - disjunctive descriptions may be required
  - training data may be inconsistent
  - training data may contain missing attribute values
- training: Which attribute should be chosen first to split the data?

ID3

- growing the tree top-down
- selecting the next attribute as to maximize the information gain
- greedy algorithm

ID3

- trivial cases (binary classification (yes/no)):
    - all examples are positive: return the single-node tree Root with value=yes
    - all examples are negative: return the single-node tree Root with value=no
    - if the set of attributes / examples is empty return the most frequent value of the target function in the training data

ID3

- select the attribute A e.g. according to maximum information gain
- add a new branch for each value v_i of the selected attribute
- partition the training data according to the value v_i
- call ID3 recursively on each partition

Information Gain

- information gain (HUNT 1966)
- based on the entropy (impurity) of a data collection

$$E = -\sum_{v \in V} p(v) \log_2 p(v)$$

with

V : range of the target function

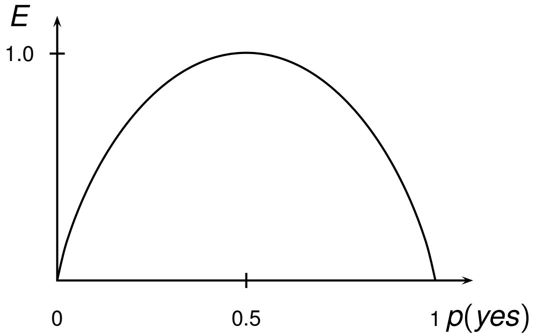
$0 \log_2 0 =_{def} 0$

Information Gain

- binary case

$$E = -p(yes)\log_2 p(yes) - p(no)\log_2 p(no)$$

$$= -p(yes)\log_2 p(yes) - (1 - p(yes))\log_2(1 - p(yes))$$



Information Gain

- Information gain: expected reduction in entropy

$$IG(S,A) =_{def} E(S) - \sum_{v \in V} \frac{|S_v|}{|S|} E(S_v)$$

S : Sample

A : Attribute to be selected

V : Domain of A

S_v : $\{s \in S | \text{value}(A) = v\}$

Information Gain

day	outlook	temperature	humidity	wind	tennis
D_1	sunny	hot	high	weak	no
D_2	sunny	hot	high	strong	no
D_3	overcast	hot	high	weak	yes
D_4	rain	mild	high	weak	yes
D_5	rain	cool	normal	weak	yes
D_6	rain	cool	normal	strong	no
D_7	overcast	cool	normal	strong	yes
D_8	sunny	mild	high	weak	no
D_9	sunny	cool	normal	weak	yes
D_{10}	rain	mild	normal	weak	yes
D_{11}	sunny	mild	normal	strong	yes
D_{12}	overcast	mild	high	strong	yes
D_{13}	overcast	hot	normal	weak	yes
D_{14}	rain	mild	high	strong	no

Information Gain

day	outlook	temperature	humidity	wind	tennis
D_1	sunny	hot	high	weak	no
D_2	sunny	hot	high	strong	no
D_3	overcast	hot	high	weak	yes
D_4	rain	mild	high	weak	yes
D_5	rain	cool	normal	weak	yes
D_6	rain	cool	normal	strong	no
D_7	overcast	cool	normal	strong	yes
D_8	sunny	mild	high	weak	no
D_9	sunny	cool	normal	weak	yes
D_{10}	rain	mild	normal	weak	yes
D_{11}	sunny	mild	normal	strong	yes
D_{12}	overcast	mild	high	strong	yes
D_{13}	overcast	hot	normal	weak	yes
D_{14}	rain	mild	high	strong	no

Information Gain

Example

wind	w	s	w	w	w	s	s	w	w	w	s	s	w	s
target	n	n	y	y	y	n	y	n	y	y	y	y	y	n

	weak	strong	$\sum$
yes	6	3	9
no	2	3	5
$\sum$	8	6	14

Information Gain

Example

wind	w	s	w	w	w	s	s	w	w	w	s	s	w	s
target	n	n	y	y	y	n	y	n	y	y	y	y	y	n

	weak	strong	$\sum$
yes	6	3	9
no	2	3	5
$\sum$	8	6	14

Information Gain

Example

wind	w	s	w	w	w	s	s	w	w	w	s	s	w	s
target	n	n	y	y	y	n	y	n	y	y	y	y	y	n

	weak	strong	$\sum$
yes	6	3	9
no	2	3	5
$\sum$	8	6	14

Information Gain

Example

wind	w	s	w	w	w	s	s	w	w	w	s	s	w	s
target	n	n	y	y	y	n	y	n	y	y	y	y	y	n

	weak	strong	$\sum$
yes	6	3	9
no	2	3	5
$\sum$	8	6	14

Information Gain

Example

wind	w	s	w	w	w	s	s	w	w	w	s	s	w	s
target	n	n	y	y	y	n	y	n	y	y	y	y	y	n

	weak	strong	$\sum$
yes	6	3	9
no	2	3	5
$\sum$	8	6	14

Information Gain

	weak	strong	$\sum$
yes	6	3	9
no	2	3	5
$\sum$	8	6	14

$$E(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940286$$

$$E(S_w) = -\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8} = 0.811278$$

$$E(S_s) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1.0$$

$$G(S, \text{wind}) = E(S) - \frac{1}{|S|}(|S_w|E(S_w) + |S_s|E(S_s)) = 0.0481271$$

Information Gain

$$IG(S, outlook) = 0.246$$

$$IG(S, humidity) = 0.151$$

$$IG(S, wind) = 0.048$$

$$IG(S, temperature) = 0.029$$

Information Gain

- after having chosen *outlook* as the first splitting attribute

value	samples	yes	no
sunny	$\{D_1, D_2, D_8, D_9, D_{11}\}$	2	3
overcast	$\{D_3, D_7, D_{12}, D_{13}\}$	4	0
rain	$\{D_4, D_5, D_6, D_{10}, D_{14}\}$	3	2

Information Gain

- after having chosen *outlook* as the first splitting attribute

value	samples	yes	no	
sunny	$\{D_1, D_2, D_8, D_9, D_{11}\}$	2	3	
overcast	$\{D_3, D_7, D_{12}, D_{13}\}$	4	0	$\rightarrow$ no further splitting
rain	$\{D_4, D_5, D_6, D_{10}, D_{14}\}$	3	2	

Information Gain

- after having chosen *outlook* as the first splitting attribute

value	samples	yes	no
sunny	$\{D_1, D_2, D_8, D_9, D_{11}\}$	2	3
overcast	$\{D_3, D_7, D_{12}, D_{13}\}$	4	0
rain	$\{D_4, D_5, D_6, D_{10}, D_{14}\}$	3	2

- compute the information gain of the remaining attributes on the subsets of the sample as defined by its values, e.g. for value(outlook)=sunny

$$IG(S_{sunny}, humidity) = 0.970$$

$$IG(S_{sunny}, temperature) = 0.570$$

$$IG(S_{sunny}, wind) = 0.019$$

Information Gain

- after having chosen *outlook* as the first splitting attribute

value	samples	yes	no
sunny	$\{D_1, D_2, D_8, D_9, D_{11}\}$	2	3
overcast	$\{D_3, D_7, D_{12}, D_{13}\}$	4	0
rain	$\{D_4, D_5, D_6, D_{10}, D_{14}\}$	3	2

- compute the information gain of the remaining attributes on the subsets of the sample as defined by its values, e.g. for value(outlook)=sunny

$$IG(S_{sunny}, humidity) = 0.970$$

$$IG(S_{sunny}, temperature) = 0.570$$

$$IG(S_{sunny}, wind) = 0.019$$

$\rightarrow$ choose *humidity* as the next splitting attribute for value(outlook)=sunny

Information Gain

- information gain favors attributes with many values
- such attributes rarely contribute to a sensible generalizations
  - e.g. date, address, name, ...
- results in flat trees
  - with a high number of branches
  - which classifies the training data perfectly
  - but poorly predicts the target function for unseen instances

$\rightarrow$ alternative measures

$\rightarrow$ alternative representations

ID3

- hypothesis space: all possible decision trees
- every discrete-valued function can be represented
- simple-to-complex hill climbing search
- guided by the information gain
- maintains only a single hypothesis
  - no awareness of alternative hypotheses
  - no possibility for active learning
- greedy search
  - no possibility to revise decisions
  - optimal tree cannot be guaranteed
- robustness against noisy data

ID3

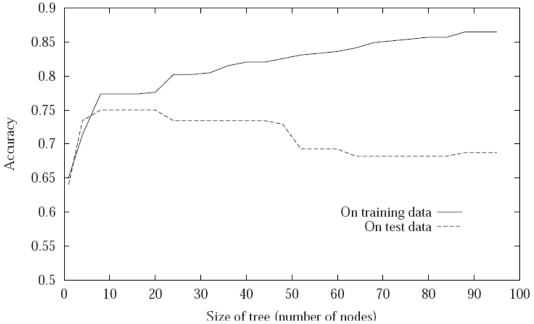
- (approximate) inductive bias:
  - prefer short trees over larger ones
  - prefer to place attributes with large information gain higher up in the tree
- ID3: preference / search bias
  - complete hypothesis space
  - incomplete search
- CANDIDATE-ELIMINATION: restriction / language bias
  - incomplete hypothesis space
  - complete search
- Which one is desirable?

Occam's Razor

- prefer the simplest hypothesis that fits the data
- general scientific principle
- but weakly justified
  - short hypotheses are less likely to be spurious
  - and therefore are more likely to generalize to unseen data
    - but many arbitrary hypothesis sets of small size can be defined which are not preferred at all
  - size is a purely internal measure
    - representations can be changed: different learners might lead to contradicting hypotheses, although both are based on Occam's razor
- better justification: minimal description length
  $\rightarrow$ Bayesian learning

Overfitting

- generating too many nodes for a given learning problem might lead to ”learning off by heart”
  - producing branches for individual training items $\rightarrow$ overfitting
- overfitting: given a hypothesis space H a hypothesis h is said to overfit the training data if there is a hypothesis h' which performs worse on the training data but better on the entire distribution of instances.



Overfitting

- countermeasures
  - stop growing the tree early enough
  - post-prune the tree

  post-pruning is usually more successful, because it has the complete tree available
- criteria for stopping growth
  - monitoring performance on held out data (validation set)
  - application of statistical tests to estimate the likelihood of a further improvement on the test set
  - minimizing an explicit measure of the complexity for encoding the training data and the decision tree
    $\rightarrow$ Minimum Description Length principle
- pruning: two possibilities
  - reduced error pruning
  - rule post-pruning

Reduced Error Pruning

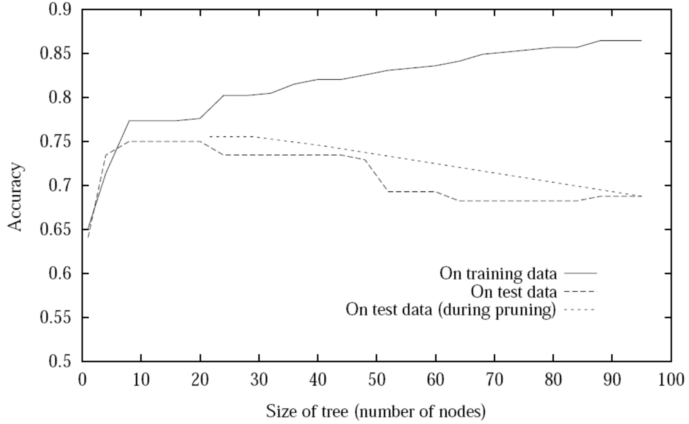
- removal of a node
  - remove a subtree rooted in a node
  - make that node a leaf node
  - assign the new leaf node the most common classification for the examples associated with that node

Algorithm

- repeat as long as this improves the performance on the validation set
  - choose the subtree which improves accuracy most if removed ...
  - ... and remove it

Reduced Error Pruning

- impact of reduced error pruning



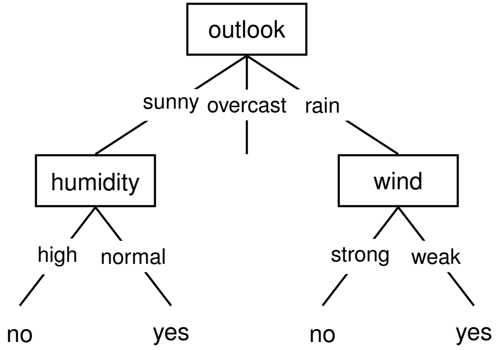
- accuracy on the validation set is not shown

Rule Post-Pruning

- converting an overfitted tree into If-Then-Rules, one for each path from the root node to a leaf
- prune (generalize) the rules by removing preconditions from their antecedents if this improves accuracy on the validation data
- sort the pruned rules by their estimated accuracy
- use them in this sequence when classifying new instances

Rule Post-Pruning

- rule conversion
  - make each attribute test on the path a precondition
  - make the classification at the leaf node the consequence



e.g. If (outlook=sunny $\wedge$ humidity=high)
Then tennis=no

Rule Post-Pruning

- pruning of the rule

  If (outlook=sunny $\wedge$ humidity=high) Then tennis=no
- check whether removing preconditions improves accuracy

  If true Then tennis=no

  If (outlook=sunny) Then tennis=no

  If (humidity=high) Then tennis=no
- do not prune if performance decreases

Continuous-Valued Attributes

- dynamically creating Boolean attributes A_c which are true if $A < c$
- learning task: determine the optimal threshold c
- optimization criterion: information gain
- can be extended to multiple interval splits

Continuous-valued Attributes

temperature	40	48	60	72	80	90
tennis	no	no	yes	yes	yes	no

- sort the examples
- generate threshold candidates
    - where two adjacent examples differ in their target value
    - as the mean of the two attribute values
- evaluate the resulting Boolean attributes as usual, e.g. by comparing their information gain

Alternative Gain Measures

- alternative: gain ratio
  - normalizing information gain by the split information

    $$GR(S,A) \;=_{def}\; \frac{G(S,A)}{SI(S,A)}$$

- split information:

  $$SI(S,A) \;=_{def}\; -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

  S_i : partitions of S according to the values of A

  - entropy of *S* with respect to the values of the attribute *A*

Alternative Splitting Criteria

- directly determining the impurity
- choose the attribute and the cut-off point(s) which *minimize impurity* of the resulting daughter nodes
- impurity is lowest if all instances belong to the same class
- entropy:

$$E = -\sum_{v \in V} p(v) \log_2 p(v)$$

$$E = -p(yes) \log_2 p(yes) - (1 - p(yes)) \log_2 (1 - p(yes))$$

Alternative Splitting Criteria

- Gini-coefficient/Gini-index: measures the statistical dispersion or diversity of a population (BREIMAN ET AL. 1984)
- n classes

$$G = 1 - \sum_{i=1}^{n} p(c_i)^2$$

- G is minimal ($G = 0$) if all instances belong to the same class
- G is maximal ($G = 1 - \frac{1}{n}$) if the distribution is even

- also used: Gini gain

Alternative Splitting Criteria

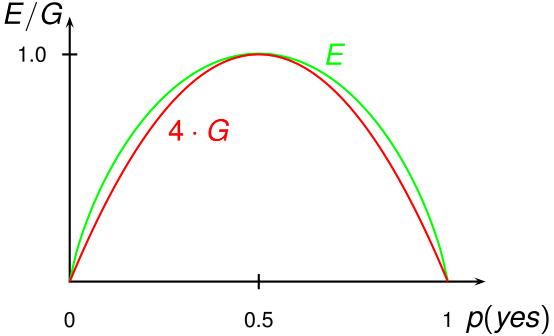
- choose the attribute and the cut-off point(s) which minimize the sum of the Gini-coefficients of the resulting daughter nodes t_j

$$\min \sum_{i=1}^{n} p(c_i|t_j)G(t_j)$$

Alternative Splitting Criteria

- 2 classes $G = p(yes) \cdot (1 - p(yes))$
- Gini coefficients approximate entropy

$$-\log_2 p(c_i) \approx 1 - p(c_i) \text{ for } p(c_i) < 1$$



Alternative Splitting Criteria

- statistical tests (e.g. χ^2)
- discriminative criteria
    - RELIEF-family (KIRA AND RENDELL 1992, KONONENKO 1994)
    - rank attributes according to how well their values distinguish between instances that are near to each other
    - for each instance *i* determine
        - nearest instance with the same class (nearest hit *nh*(*i*))
        - nearest instance with a different class (nearest miss *nm*(*i*))
        - scoring an attribute *a* as

$$w(a) = P(a(i) \neq a(nm(i))) - P(a(i) \neq a(nh(i)))$$

Handling Missing Attribute Values

- assign the most common value among the examples at the node

  or
- determine a probability distribution for the values of the attribute
  - use the probabilities to compute the information gain
  - can be further propagated down the tree

Handling Attributes with Different Costs

- measurement for an attribute can be expensive
- cost: $C(A)$
- cost sensitive selection measures, i.e.

$$\frac{(G(S,A))^2}{C(A)} \quad \text{or} \quad \frac{2^{G(S,A)}-1}{(C(A)+1)^w} \quad w \in [0,1]$$

C4.5

- extensions
  - numerical attributes
  - statistical test for pruning decisions
  - using probability distributions in case of missing values
- fundamental drawback of decision trees
  - no dependencies (e.g. linear ones) between arguments can be represented

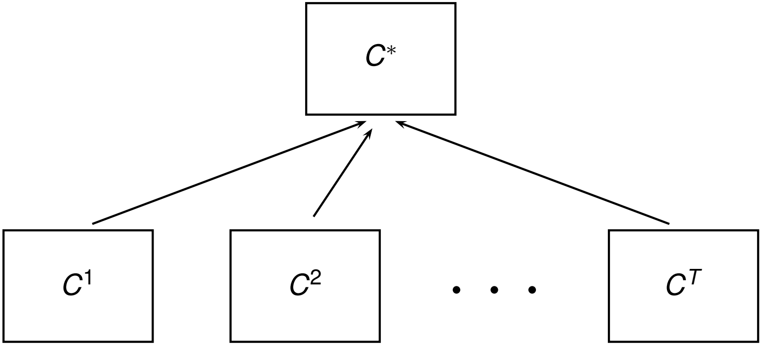
Learning Symbolic Structures

- Concept Learning
- Decision Trees
- Ensemble-based Methods
- Rule Induction
- Inductive Logic Programming
- Association Rules
- Transformation rules

Ensemble-based Methods

- usually
  - alternative approaches for a learning task available
  - training can be done with different parameter settings
- under certain conditions a (partly) complementary behaviour can be expected
  - differences in accuracy
  - different kind of errors
- combining the outcome of differently trained systems migth lead to synergy
  - combination is easiest for simple classifiers

Ensemble-based Methods



Ensemble-based Methods

- different kinds of combination possibilities
- voting
  - e.g. single vote majority voting: decide for the class which was chosen by the majority of classifiers in the ensemble
- weighted voting
  - reliability of the classifier
  - class-specific reliability of the classifier
  - confidence estimation
  - cost-balanced confidence estimation

$$R(i|x) = \sum_{j=1}^{n} P(j|x) c(i,j)^{-1}$$

$c(i,j)$: estimated costs for substituting class i by class j

Ensemble-based Methods

- stacking
  - train a second stage classifier, which learns to decide which first-stage classifier to trust most under certain conditions
- co-training
  - enrich the training data of one classifier with the output of a second one

Ensemble-based Methods

- How to create an ensemble of classifiers?
  - use different classification paradigms:
    - decision trees
    - probabilistic approaches
    - neuronal networks
    - ...
  - bagging
  - boosting

Bagging

- BREIMAN (1996)
- *T* random samplings (”trials”) of the training data with replacement
  - all training sets have the same size as the original one
  - some training instances are not used, some appear multiple times
- *T* different classifiers are trained
- classifier output is combined using a (unit) majority vote

Bagging

- at least ten classifiers used; typically 100 or more
- reduction of the error rate up to a factor of 0.5
- prerequisite: instability of the prediction method
  - small changes of the training set should result in significant changes of a classifiers' output
- increase of the error rate in some cases
  - ”poor predictors can be transformed into worse ones”

Boosting

- AdaBoost Freund and Schapire (1996)
- using all training data but maintaining a weight w_x^t for each training instance x at trial t, initially $w_x^1 = N^{-1}$
- the weight reflects the importance of the training item
- the error of the classifier is estimated

$$e^t = \sum_{x \in M} w_x^t$$

where M is the set of misclassified training instances

Boosting

- each misclassification causes the weight to be increased:
  - reduce weight of the correctly classified items by a factor

$$\beta^t = \frac{e^t}{1 - e^t}$$

  - renormalizing according to

$$\sum_x w_x^{t+1} = 1$$

- different weights cause the learning procedure to emphasize different training instances and therefore create different classifiers
  $\rightarrow$ weight-sensitive splitting criteria

$$p(c_i) = \sum_{x|c(x)=i} w(x)$$

Boosting

- boosting continued until
  - if either $t > T$, $e^t > 0.5$ or $e^t = 0$ (all instances are classified correctly)
- no real boosting but adaptation to the training data
  $\rightarrow$ reduction of the variance
- combination of all T classifiers by a weighted vote based on the accuracy of the individual trial

$$v^t = \log \frac{1}{\beta^T} = \log \frac{1 - e^t}{e^t}$$

Random Forests

- N training instances with M variables
- T random samples (with replacement) with a sample size $S \approx 0.66N$; the remaining instances are used for estimating the prediction error (out-of-bag evaluation)
- for each node of the tree $m \ll M$ variables are selected randomly to take the decision at that node
- no pruning is applied

Random Forests

- highly accurate classifiers without much prior knowledge
- fast training
- highly parallel classification
- can deal with very large problems (training data, number of variables, number of classes)
- estimates the importance of the different variables
- build-in evaluation; no separate test data required

Random Forests

- prone to overfitting, in particular for noisy data sets
- does not handle large amounts of irrelevant features
- sensitive to correlations between trials: ”small heterogeneous is better than large homogeneous” (GASHLER ET AL. 2008)
  - growing too many trees might reduce the accuracy

Learning Symbolic Structures

- Concept Learning
- Decision Trees
- Ensemble-based Methods
- Rule Induction
- Inductive Logic Programming
- Association Rules
- Transformation rules

Rule Induction

- direct approaches
  - rule generation (representational bias)
  - rule assessment
  - rule selection
  - rule reranking

Rule Induction

- direct approaches
  - propositional descriptions:
    $\rightarrow$ sequential covering algorithms
  - first-order descriptions:
    $\rightarrow$ inductive logic programming
  - typical common occurence:
    $\rightarrow$ association rules
  - context sensitive classification:
    $\rightarrow$ transformation rules

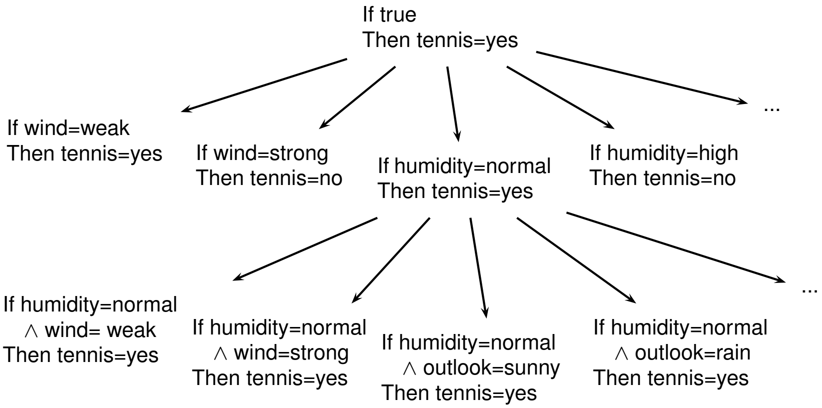
Rule Induction

- indirect approaches
    - transforming a trained model into a rule format
    - e.g. decision trees
    - e.g. neural networks
    - ...

Sequential covering algorithms

- find a rule that covers many positive examples and few negative ones
    - high accuracy
    - not necessarily high coverage
- remove the positive examples from the training data
- sort the rules according to their accuracy
- greedy approach
    - optimal (smallest, most accurate, ...) set of rules cannot be guaranteed

Sequential covering algorithms



Sequential covering algorithms

- general to specific beam search
  - start with the most general precondition (the empty one)
  - add more attribute tests to the precondition
  - sort the attribute tests according to improvement of rule performance
  - select the k-best candidates
  - call recursively as long as performance is still improving
  - (post-prune the rules by removing attribute tests if this improves performance)

Performance measures

- relative frequency: $\frac{n_c}{n}$

  n_c: rule applies correctly; n: rule applies

- m-estimate (for scarce data): $\frac{n_c+mp}{n+m}$

  m: weighting factor

  - biased towards the prior probability p of the outcome of the rule
  - $m = 0 \quad \rightarrow$ relative frequency / $m \rightarrow \infty \quad \rightarrow$ prior probability

- negative entropy: uniformity of the target function for the sample set S covered by the rule

$$NE(S) = \sum_{i=1}^{c} p_i \log_2 p_i$$

Comparison

- SCA chooses attribute-value pairs by comparing the subsets of the training data they cover
  ID3 chooses attribute tests
  $\rightarrow$ ID3 has to make fewer choices
  $\rightarrow$ SCA requires more training data
- general-to-specific search has a unique starting point
  specific-to-general methods have not
- SCA is hypothesis driven
  training data are considered only after rule generation

Learning Symbolic Structures

- Concept Learning
- Decision Trees
- Ensemble-based Methods
- Rule Induction
- Inductive Logic Programming
- Association Rules
- Transformation rules

Inductive Logic Programming

- learning first-order rules (Horn clauses)
- general knowledge cannot be expressed solely by attribute-value assignments
- e.g. family relationships
  - $\rightarrow$ relationships *between* data items
    If father(x,y) $\wedge$ female(y) Then daughter(y,x)
  - $\rightarrow$ dependencies between antecedent and consequence
    If father(x,y) $\wedge$ mother(y,z) $\wedge$ female(z) Then granddaughter(z,x)
  - $\rightarrow$ recursive rules, e.g. ancestor(x,y)

Inductive Logic Programming

- different approaches
  e.g.
  - extended sequential covering
  - inverted deduction

Sequential Covering for First-Order Rules

- simplified Horn clauses
    - no function symbols (deductive databases)
- extended Horn clauses
    - literal in the body can be negative
- uses only positive data

Sequential Covering for First-Order Rules

- extension of rule generation to accomodate variables (FOIL)
- given:
  - Q set of predicate names which can be used
  - training data
- start with the most general clause, i.e. with an empty antecedent $p(x_1, ..., x_k) \leftarrow true$
- generate new literals L_i to be added to the preconditions of a clause $p(x_1, ..., x_k) \leftarrow L_1, ..., L_n$
  - $q(v_1, ..., v_r)$ with $q \in Q$ and v_i new or already existing variables; at least one v_i must already exist
  - $equal(x_j, x_k)$ with x_j and x_k are variables already present in the rule
  - the negation of the above
- terminates if all positive training instances are covered by the rule

Sequential Covering for First-Order Rules

- example
  - training data:

    granddaughter(sharon,victor), father(bob,sharon), father(victor,bob), father(bob,tom), female(sharon)

  - initial hypothesis:
    - granddaughter(x,y) $\leftarrow$ true
  - candidate literals for specialization

    equal(x,y), female(x), female(y), father(x,y), father(y,x), father(x,z), father(z,x), father(y,z), father(z,y), and their negations

  - most promising candidate: father(z,x)
  - specialized hypothesis:

    granddaughter(x,y) $\leftarrow$ father(z,x)

Sequential Covering for First-Order Rules

- example (cont.)
  - hypothesis

    granddaughter(x,y) $\leftarrow$ father(z,x)

  - additional new candidate literals for specialization
    - female(z), equal(x,z), equal(y,z), father(z,w), father(w,z) and their negations
  - most promising candidate: father(y,z)
  - specialized hypothesis:

    granddaughter(x,y) $\leftarrow$ father(z,x) $\wedge$ father(y,z)

  - ...
  - specialized hypothesis:

    granddaughter(x,y)
    $\leftarrow$ father(z,x) $\wedge$ father(y,z) $\wedge$ female(x)

Sequential Covering for First-Order Rules

- large branching factor of the search space
  $\rightarrow$ effective heuristic guide needed
- performance measure needs to distinguish between different variable bindings
- Foil_Gain:
  - evaluating all possible variable bindings
  - estimating the utility of adding a new literal L to a rule R to obtain a new rule R'
  - preference for literals which results in more positive than negative variable bindings for the rule
- e.g. 16 bindings for the literal granddaughter(x,y)

  4 constants (persons) mentioned in the training data (bob,sharon,victor,tom)
  - 1 binding covering positive information: (x/sharon,y/victor)
  - 15 bindings covering negative information (not mentioned in the training data, e.g. x/bob, y/tom)

Sequential Covering for First-Order Rules

- Foil-Gain

$$FG(L, R) =_{def} t(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0})$$

  p_0/n_0: number of positive/negative bindings of R
  p_1/n_1: number of positive/negative bindings of the new rule R'
  t: number of positive bindings of R that remain covered after adding literal L to R

- reduction of the number of bits needed to encode the positive bindings caused by the introduction of L

Sequential Covering for First-Order Rules

- if the target predicate is included into the list of possible literals, recursive clauses can be learned
- successful sample applications:
  - recursive definition of Quicksort
  - rules to discriminate legal from illegal chess positions
- learning with noisy data:
  - trade-off between accuracy, coverage and complexity
  - new literals are added only, if their description length is shorter than the description length of the training data they explain
  - post-pruning of rules to avoid overfitting

Inverse Deduction

- deduction: deriving theorems (conclusions) from axioms (general knowledge)
- deduction rules for first order logic: resolution, entailment
- Can resolution/entailment be reversed?

- find a hypothesis h such that the target function $f(x_i)$ follows deductively from h, the observation x_i and any available background knowledge B

$$\forall \langle x_i, f(x_i) \rangle \in D \,.\, B \wedge x_i \wedge h \vdash f(x_i)$$

Inductive Logic Programming

- example
  - x_i: male(bob), female(sharon), father(bob,sharon)
  - $f(x_i)$: child(sharon,bob)
  - B: parent(x,y) $\leftarrow$ father(x,y)
  - h_1: child(x,y) $\leftarrow$ father(y,x)
    $$h_1 \wedge x_i \vdash \text{child(sharon,bob)}$$
  - h_2: child(x,y) $\leftarrow$ parent(y,x)
    $$h_2 \wedge x_i \wedge B \vdash \text{child(sharon,bob)}$$

  constructive induction: introducing new predicates (based on background knowledge) which have not been present in the original description

Inductive Logic Programming

- description of learning in a framework for logic deduction
- background knowledge can be used to guide the search
- problem: highly sensitive to noisy data: arbitrary theorems can be derived from inconsistent axioms
- search space is intractable in the general case
- often complexity increases if background knowledge is introduced

Inductive Logic Programming

- approach: inverting resolution
- resolution

$$\frac{\begin{array}{c} P \lor L \\ \neg L \lor R \end{array}}{P \lor R} \qquad\qquad \frac{\begin{array}{c} P \lor L \\ L \rightarrow R \end{array}}{P \lor R}$$

Inverse Resolution

$$\begin{tikzcd}
c_1\text{: knowmaterial} \rightarrow \text{passexam} \arrow[dr] & & c_2\text{: study} \rightarrow \text{knowmaterial} \arrow[dl] \\
& \boxed{c\text{: study} \rightarrow \text{passexam}} &
\end{tikzcd}$$

$$\begin{tikzcd}
& & \boxed{c_2\text{: study} \rightarrow \text{knowmaterial}} \\
c_1\text{: knowmaterial} \rightarrow \text{passexam} \arrow[dr, dash] & & \\
& c\text{: study} \rightarrow \text{passexam} \arrow[uur] &
\end{tikzcd}$$

Inverse Resolution

$$\begin{tikzcd}
 & & c_2\text{: } \neg\ \text{study} \lor \text{knowmaterial} \arrow[dl] \\
c_1\text{: } \neg\ \text{knowmaterial} \lor \text{passexam} \arrow[r] & \boxed{c\text{: } \neg\ \text{study} \lor \text{passexam}} &
\end{tikzcd}$$

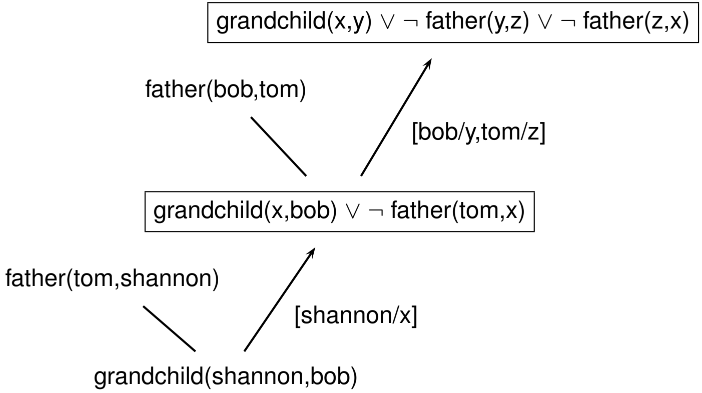
$$\begin{tikzcd}
 & & \boxed{c_2\text{: } \neg\ \text{study} \lor \text{knowmaterial}} \\
c_1\text{: } \neg\ \text{knowmaterial} \lor \text{passexam} \arrow[r, dash] & c\text{: } \neg\ \text{study} \lor \text{passexam} \arrow[ur] &
\end{tikzcd}$$

Inverse Resolution

- inverse resolution (propositional case)
    - given clauses c_1 and c
    - find a literal $L \in c_1 \wedge L \notin c$
    - $c_2 = (c - c_3) \cup \{\neg L\}$ with $c_3 \subseteq c_1 - \{L\}$
- non-deterministic choice:
    - multiple solutions by adding literals from c_1
- heuristics:
    - do not allow c and c_1 to share common literals
      i.e. prefer shorter descriptions over longer ones
    - $c_2 = (c - (c_1 - \{L\})) \cup \{\neg L\}$

Inverse Resolution

- extension to the first-order case
- inverse resolution under unification $\rightarrow$ inverse substitutions


grandchild(x,y) ∨ ¬ father(y,z) ∨ ¬ father(z,x)
father(bob,tom)
[bob/y,tom/z]
grandchild(x,bob) ∨ ¬ father(tom,x)
father(tom,shannon)
[shannon/x]
grandchild(shannon,bob)

- alternative substitutions possible!

Learning Symbolic Structures

- Concept Learning
- Decision Trees
- Ensemble-based Methods
- Rule Induction
- Inductive Logic Programming
- Association Rules
- Transformation rules

Association Rules

- prediction of events commonly occurring together

Association Rules

- prediction of events commonly occurring together
- market basket analysis: which items are often purchased together
  - placement of items in a store
  - layout of mail-order catalogues
  - targeted marketing campaigns
- no complete description is aimed at
- coverage of the most typical cases is sufficient
- association rules: rules of the form

$$a \land b \land \ldots \land c \rightarrow d \land e$$

Association Rules

- prediction of events commonly occurring together
- market basket analysis: which items are often purchased together
  - placement of items in a store
  - layout of mail-order catalogues
  - targeted marketing campaigns
- no complete description is aimed at
- coverage of the most typical cases is sufficient
- association rules: rules of the form

$$a \wedge b \wedge \ldots \wedge c \rightarrow d \wedge e$$

- finding good combinations of premises is a combinatorial problem

Association Rules

- example data base:

trans-action	item
001	cola
001	chips
001	peanuts
002	beer
002	chips
002	cigarettes
. . .	. . .

trans-action	items
001	{chips, cola, peanuts}
002	{beer, chips, cigarettes}
003	{beer, chips, cigarettes, cola}
004	{beer, cigarettes}

Association Rules

- set of n different items $I = \{x_j | j = 1, \ldots, n\}$

Association Rules

- set of n different items $I = \{x_j | j = 1, \ldots, n\}$
- itemset: $I_k \subseteq I$

Association Rules

- set of n different items $I = \{x_j | j = 1, \ldots, n\}$
- itemset: $I_k \subseteq I$
- i-itemset: $I_k^i \subseteq I, \ \ |I_k^i| = i$

Association Rules

- set of n different items $I = \{x_j | j = 1, \ldots, n\}$
- itemset: $I_k \subseteq I$
- i-itemset: $I_k^i \subseteq I, \;\; |I_k^i| = i$
- transaction $T_k \subseteq I$

Association Rules

- set of n different items $I = \{x_j | j = 1, \ldots, n\}$
- itemset: $I_k \subseteq I$
- i-itemset: $I_k^i \subseteq I, \;\; |I_k^i| = i$
- transaction $T_k \subseteq I$
- data base: $D = \{(k, T_k) | k = 1, \ldots, m\}$

Association Rules

- set of n different items $I = \{x_j | j = 1, \ldots, n\}$
- itemset: $I_k \subseteq I$
- i-itemset: $I_k^i \subseteq I, \;\; |I_k^i| = i$
- transaction $T_k \subseteq I$
- data base: $D = \{(k, T_k) | k = 1, \ldots, m\}$
- support of an itemset: share of transactions which contain the itemset

$$s(I_i) = \frac{|\{T_k | I_i \subseteq T_k\}|}{|D|}$$

Association Rules

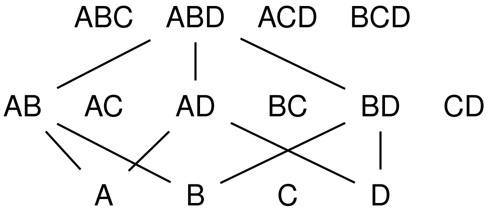
- set of n different items $I = \{x_j | j = 1, \ldots, n\}$
- itemset: $I_k \subseteq I$
- i-itemset: $I_k^i \subseteq I, \quad |I_k^i| = i$
- transaction $T_k \subseteq I$
- data base: $D = \{(k, T_k) | k = 1, \ldots, m\}$
- support of an itemset: share of transactions which contain the itemset

$$s(I_i) = \frac{|\{T_k | I_i \subseteq T_k\}|}{|D|}$$

- frequent (strong, large) itemset: $s(I_i) \geq s_{min}$

Association Rules

- downward closure: every subset of a frequent itemset is also a frequent itemset



- every superset of a not frequent itemset is also a not frequent itemset

Association Rules

- association rule: $X \rightarrow Y, \quad X, Y \subseteq I, Y \cap X = \emptyset$

Association Rules

- association rule: $X \rightarrow Y, \quad X, Y \subseteq I, Y \cap X = \emptyset$
- support of a rule: share of transactions which contain both, premise and conclusion of the rule

$$s(X \rightarrow Y) = s(X \cup Y) = \frac{|\{T_k | X \cup Y \subseteq T_k\}|}{|D|} = p(XY)$$

Association Rules

- association rule: $X \rightarrow Y, \quad X, Y \subseteq I, Y \cap X = \emptyset$
- support of a rule: share of transactions which contain both, premise and conclusion of the rule

$$s(X \rightarrow Y) = s(X \cup Y) = \frac{|\{T_k | X \cup Y \subseteq T_k\}|}{|D|} = p(XY)$$

- confidence of a rule: share of transactions supporting the rule from those supporting the premise

$$c(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)} = \frac{|\{T_k | X \cup Y \subseteq T_k\}|}{|\{T_k | X \subseteq T_k\}|} = p(Y|X)$$

Association Rules

- strong rule: high support + high confidence

Association Rules

- strong rule: high support + high confidence
- detection of strong rules: two pass algorithm

Association Rules

- strong rule: high support + high confidence
- detection of strong rules: two pass algorithm

1. find frequent (strong, large) itemsets (Apriori)
    - necessary to generate rules with strong support
    - uses the downward closure
    - itemsets are ordered

Association Rules

- strong rule: high support + high confidence
- detection of strong rules: two pass algorithm

1. find frequent (strong, large) itemsets (Apriori)
   - necessary to generate rules with strong support
   - uses the downward closure
   - itemsets are ordered
2. use the frequent itemsets to generate association rules
   - find strong correlations in a frequent itemset

Association Rules

- Apriori: finding frequent itemsets of increasing size
  itemsets are ordered!
  - start with all itemsets of size one: I^1
  - select all itemsets with sufficient support
  - from the selected itemsets I^i generate larger itemsets I^{i+1}

$$is(\{i_1,\ldots,i_{n-2},i_{n-1}\}) \wedge is(\{i_1,\ldots,i_{n-2},i_n\}) \rightarrow is(\{i_1,\ldots,i_{n-2},i_{n-1},i_n\})$$

    - already blocks some of the non-frequent itemsets, but not all of them
  - remove those itemsets which still contain a non-frequent immediate subset
    - they cannot have enough support (downward closure)
  - continue until no further frequent itemsets can be generated

Association Rules

- example data base again
- assumption: minimum support $s_{min} = 0.5$

k	T_k
001	{chips, cola, peanuts}
002	{beer, chips, cigarettes}
003	{beer, chips, cigarettes, cola}
004	{beer, cigarettes}

I_k^1	#	$s(I_k^1)$
{chips}		
{cola}		
{peanuts}		
{beer}		
{cigarettes}		

Association Rules

- example data base again
- assumption: minimum support $s_{min} = 0.5$

k	T_k
001	{chips, cola, peanuts}
002	{beer, chips, cigarettes}
003	{beer, chips, cigarettes, cola}
004	{beer, cigarettes}

I_k^1	#	$s(I_k^1)$
{chips}		
{cola}		
{peanuts}		
{beer}		
{cigarettes}		

- no non-empty subsets

Association Rules

- example data base again
- assumption: minimum support $s_{min} = 0.5$

k	T_k
001	{chips, cola, peanuts}
002	{beer, chips, cigarettes}
003	{beer, chips, cigarettes, cola}
004	{beer, cigarettes}

I_k^1	#	$s(I_k^1)$
{chips}	3	0.75
{cola}	2	0.5
{peanuts}	1	0.25
{beer}	3	0.75
{cigarettes}	3	0.75

- no non-empty subsets

Association Rules

- example data base again
- assumption: minimum support $s_{min} = 0.5$

k	T_k
001	{chips, cola, peanuts}
002	{beer, chips, cigarettes}
003	{beer, chips, cigarettes, cola}
004	{beer, cigarettes}

I_k^1	#	$s(I_k^1)$
{chips}	3	0.75
{cola}	2	0.5
{peanuts}	1	0.25
{beer}	3	0.75
{cigarettes}	3	0.75

- no non-empty subsets

Association Rules

- 2-itemsets I_k^2

I_k^1	#	$s(I_k^1)$
{chips}	3	0.75
{cola}	2	0.5
{beer}	3	0.75
{cigarettes}	3	0.75

I_k^2	#	$s(I_k^2)$
{chips, cola}		
{beer, chips}		
{chips, cigarettes}		
{beer, cola}		
{cigarettes, cola}		
{beer, cigarettes}		

Association Rules

- 2-itemsets I_k^2

I_k^1	#	$s(I_k^1)$
{chips}	3	0.75
{cola}	2	0.5
{beer}	3	0.75
{cigarettes}	3	0.75

I_k^2	#	$s(I_k^2)$
{chips, cola}		
{beer, chips}		
{chips, cigarettes}		
{beer, cola}		
{cigarettes, cola}		
{beer, cigarettes}		

- no itemsets to prune

Association Rules

- 2-itemsets I_k^2

I_k^1	#	$s(I_k^1)$
{chips}	3	0.75
{cola}	2	0.5
{beer}	3	0.75
{cigarettes}	3	0.75

I_k^2	#	$s(I_k^2)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigarettes}	2	0.5
{beer, cola}	1	0.25
{cigarettes, cola}	1	0.25
{beer, cigarettes}	3	0.75

Association Rules

- 2-itemsets I_k^2

I_k^1	#	$s(I_k^1)$
{chips}	3	0.75
{cola}	2	0.5
{beer}	3	0.75
{cigarettes}	3	0.75

I_k^2	#	$s(I_k^2)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigarettes}	2	0.5
{beer, cola}	1	0.25
{cigarettes, cola}	1	0.25
{beer, cigarettes}	3	0.75

Association Rules

- 3-itemsets I_k^3

I_k^2	#	$s(I_k^2)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigar.}	2	0.5
{beer, cigar.}	3	0.75

I_k^3	#	$s(I_k^2)$
{beer, chips, cigar.}		
{chips, cigar., cola}		

Association Rules

- 3-itemsets I_k^3

I_k^2	#	$s(I_k^2)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigar.}	2	0.5
{beer, cigar.}	3	0.75

I_k^3	#	$s(I_k^2)$
{beer, chips, cigar.}		
{chips, cigar., cola}		

Association Rules

- 3-itemsets I_k^3

I_k^2	#	$s(I_k^2)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigar.}	2	0.5
{beer, cigar.}	3	0.75

I_k^3	#	$s(I_k^2)$
{beer, chips, cigar.}		
{chips, cigar., cola}		

Association Rules

- 3-itemsets I_k^3

I_k^2	#	$s(I_k^2)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigar.}	2	0.5
{beer, cigar.}	3	0.75

I_k^3	#	$s(I_k^2)$
{beer, chips, cigar.}	2	0.5
{chips, cigar., cola}		

Association Rules

- 3-itemsets I_k^3

I_k^2	#	$s(I_k^2)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigar.}	2	0.5
{beer, cigar.}	3	0.75

I_k^3	#	$s(I_k^2)$
{beer, chips, cigar.}	2	0.5
{chips, cigar., cola}		

Association Rules

- 3-itemsets I_k^3

I_k^2	#	$s(I_k^2)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigar.}	2	0.5
{beer, cigar.}	3	0.75

I_k^3	#	$s(I_k^2)$
{beer, chips, cigar.}	2	0.5
{chips, cigar., cola}	1	0.25

Association Rules

- resulting frequent itemsets:

  {beer, chips, cigarettes}
  {chips, cola}
  {chips, beer}
  {chips, cigar.}
  {beer, cigar.}
  {beer}
  {chips}
  {cigarettes}
  {cola}

Association Rules

- generation of strong association rules:
  - for all frequent itemsets l_j determine all nonempty subsets l_k for which

$$c = \frac{s(l_j)}{s(l_k)} \geq c_{min}$$

  - add a rule $l_k \rightarrow Y, \quad Y = l_j - l_k$ to the rule set

Association Rules

- generation of strong association rules:
  - for all frequent itemsets I_j determine all nonempty subsets I_k for which

$$c = \frac{s(I_j)}{s(I_k)} \geq c_{min}$$

  - add a rule $I_k \rightarrow Y, \quad Y = I_j - I_k$ to the rule set
- e.g. $s(\{chips\}) = 0.75, s(\{cola\}) = 0.5,$ $s(\{chips, cola\}) = 0.5$

rule	confidence
{cola} $\rightarrow$ {chips}	1.00
{chips} $\rightarrow$ {cola}	0.67

Association Rules

- interesting association rules: only those for which the confidence is greater than the support of the conclusion

$$c(X \to Y) > s(Y)$$

Association Rules

- interesting association rules: only those for which the confidence is greater than the support of the conclusion

  $$c(X \rightarrow Y) > s(Y)$$

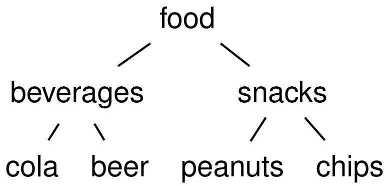
- negative border:

  $$\{I_k \mid s(I_k) < s_{min} \land \forall I_j \subset I_k \,.\, s(I_j) \geq s_{min}\}$$

  used
  - to compute the set of frequent itemsets more efficiently
  - to derive negative association rules

Association Rules

- hierarchical Apriori algorithm
  - in addition to the base level of items, determine also frequent itemsets on a higher level in an is-a hierarchy



  - sometimes regularities can only be found at higher levels of abstraction

Association Rules

- the Apriori algorithm requires several scans of the database

Association Rules

- the Apriori algorithm requires several scans of the database
- goal: reducing the number of scans

Association Rules

- the Apriori algorithm requires several scans of the database
- goal: reducing the number of scans
- partitioned Apriori: two scans

Association Rules

- the Apriori algorithm requires several scans of the database
- goal: reducing the number of scans
- partitioned Apriori: two scans
  - 1st scan: partition the database and compute locally frequent itemsets on the partitions

Association Rules

- the Apriori algorithm requires several scans of the database
- goal: reducing the number of scans
- partitioned Apriori: two scans
    - 1st scan: partition the database and compute locally frequent itemsets on the partitions
    - 2nd scan: determine the support of all locally frequent itemsets

Association Rules

- the Apriori algorithm requires several scans of the database
- goal: reducing the number of scans
- partitioned Apriori: two scans
  - 1st scan: partition the database and compute locally frequent itemsets on the partitions
  - 2nd scan: determine the support of all locally frequent itemsets
  - heuristics: if an itemset is globally frequent it will be so locally in at least one partition
    $\rightarrow$ second scan deals with a superset of possible itemsets

Association Rules

- sampling: multiple scans

Association Rules

- sampling: multiple scans
  - 1st scan: take a sample and compute frequent itemsets

Association Rules

- sampling: multiple scans
  - 1st scan: take a sample and compute frequent itemsets
  - 2nd scan: count their support and the support for their immediate supersets

Association Rules

- sampling: multiple scans
    - 1st scan: take a sample and compute frequent itemsets
    - 2nd scan: count their support and the support for their immediate supersets
    - if the itemset is at the negative border
        - all frequent itemsets have been found
        - else check supersets of the itemsets for being at the negative border in subsequent scans

Association Rules

- incremental update: scan only the added transactions, whether they
  - invalidate a former frequent itemset, or
  - introduce new frequent itemsets

Learning Symbolic Structures

- Concept Learning
- Decision Trees
- Ensemble-based Methods
- Rule Induction
- Inductive Logic Programming
- Association Rules
- Transformation rules

Transformation Rules

- task: context-sensitive classification (sequence labelling)
- e.g. genome analysis, part-of-speech-tagging
- target function with infinitely many values

observation:	*He*	*wants*	*to*	*work*	*on*	*the*	*job*	.
target:	PRP	VBZ	TO	NN	IN	DT	NN	.

- successively correction of erroneous classifications by means of context-sensitive rules, e.g.
  Change NN to VB if the previous tag is TO
- supervised learning of the transformation rules
- rule applications change the preconditions of other rule applications

Transformation Rules

Algorithm

1. start with an educated guess (e.g. highest prior probability)
2. run the current rule set over the training data
3. compare its outcome with the target
4. generate a rule which repairs most of the current errors
5. continue with 2 until no further improvement can be achieved

Transformation Rules

- rule generation restricted by templates
  - Change tag *a* to tag *b* if . . .

    . . . the preceding/following word is tagged *z*.

    . . . the word two before/after is tagged *z*.

    . . . one of the two preceding/following words is tagged *z*.

    . . . one of the three preceding/following words is tagged *z*.

    . . . the preceding word is tagged *z* and the following word is tagged *w*.

    . . . the preceding/following word is tagged *z* and the word two before/after is tagged *w*.

Transformation Rules

- result: ordered list of transformation rules

from	to	condition	example
NN	VB	previous tag is TO	to/TO race/NN $\rightarrow$ VB
VBP	VB	one of the 3 prev. tags is MD	might/MD vanish/VBP $\rightarrow$ VB
NN	VB	one of the 2 prev. tags is MD	might/MD not reply/NN $\rightarrow$ VB
VB	NN	one of the 2 prev. tags is DT	
VBD	VBN	one of the 3 prev. tags is VBZ	

Transformation Rules

- 97.0% accuracy, if only the first 200 rules are used
- 96.8% accuracy, if only the first 100 rules are used
- achieves the quality of a probabilistic tagger on the same data (96.7%) with only 82 rules
- extremly time consuming training
  $\approx 10^6$ times of a probabilistic tagger

Algorithmisches Lernen/Machine Learning

Part 1: Wolfgang Menzel

- Learning of Symbolic Structures
- (Distance-Based Methods)
- Bayesian Learning

Part 2: Peer Stelldinger

- Connectionist Learning (e.g. Neural Networks)
- Dimensionality Reduction

Part 3: Jianwei Zhang

- Function approximation
- Reinforcement Learning
- Support-Vector Machines
- Applications in Robotics

Distance-based approaches

- nearest neighbor classifier
- k-nearest neighbor classifier
- instance-based methods
- memorizing the complete training sample
- more data $\rightarrow$ better results
- but no proper learning/generalization

- clustering
- grouping of observations
  - based on a similarity/dissimilarity measure
  - unsupervised learning (without teacher)
- often used for data compression

Clustering

- hierarchical clustering
- partitioning clustering
- (clustering with neural networks)

Clustering

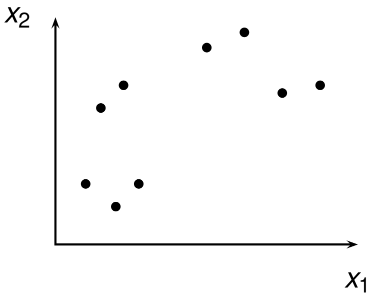
- computing the optimal clustering is computationally infeasible
  $\rightarrow$ greedy, sub-optimal approaches

Clustering

- computing the optimal clustering is computationally infeasible
  $\rightarrow$ greedy, sub-optimal approaches
- different clustering algorithms might lead to different clustering results

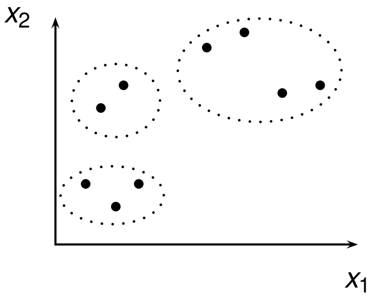
Clustering

- computing the optimal clustering is computationally infeasible $\rightarrow$ greedy, sub-optimal approaches
- different clustering algorithms might lead to different clustering results



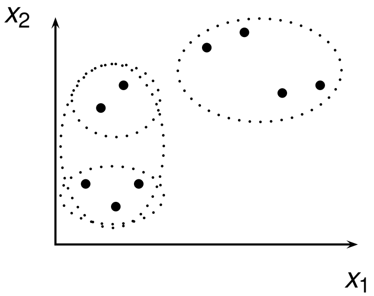
Clustering

- computing the optimal clustering is computationally infeasible
  $\rightarrow$ greedy, sub-optimal approaches
- different clustering algorithms might lead to different clustering results



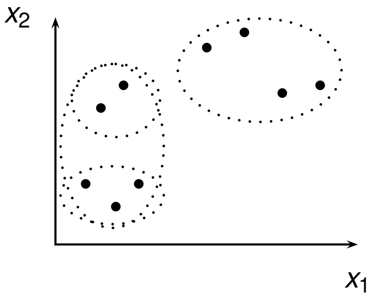
Clustering

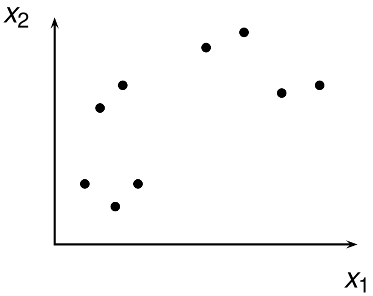
- computing the optimal clustering is computationally infeasible
  $\rightarrow$ greedy, sub-optimal approaches
- different clustering algorithms might lead to different clustering results



Clustering

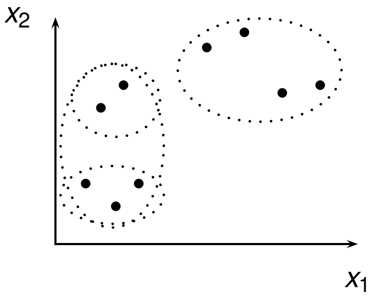
- computing the optimal clustering is computationally infeasible
  $\rightarrow$ greedy, sub-optimal approaches
- different clustering algorithms might lead to different clustering results

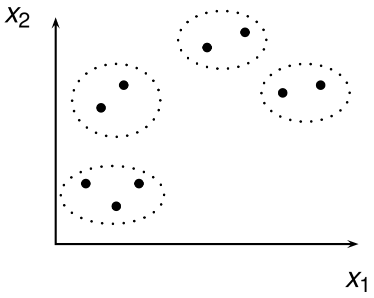




Clustering

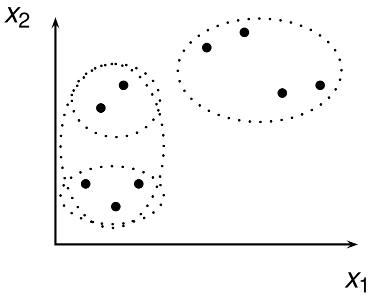
- computing the optimal clustering is computationally infeasible
  $\rightarrow$ greedy, sub-optimal approaches
- different clustering algorithms might lead to different clustering results

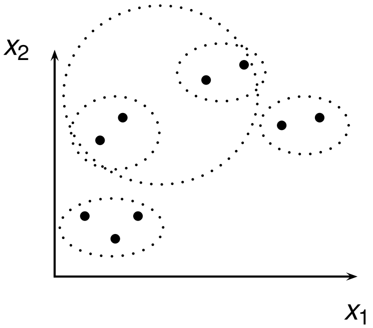




Clustering

- computing the optimal clustering is computationally infeasible
  $\rightarrow$ greedy, sub-optimal approaches
- different clustering algorithms might lead to different clustering results





Hierarchical Clustering

- agglomerative hierarchical clustering

Hierarchical Clustering

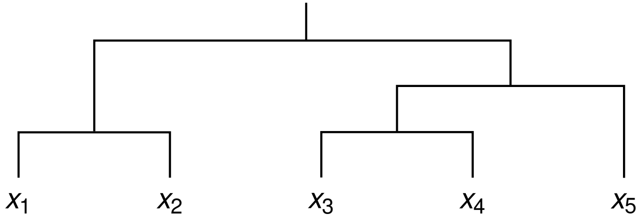
- agglomerative hierarchical clustering
- successively merging data sets

Hierarchical Clustering

- agglomerative hierarchical clustering
- successively merging data sets
- result can be displayed as a dendrogram

Hierarchical Clustering

- agglomerative hierarchical clustering
- successively merging data sets
- result can be displayed as a dendrogram



Hierarchical Clustering

- algorithm
  - initially each cluster consists of a single data point
  - determine all inter-cluster distances
  - merge the least distant clusters into a new one
  - continue until all clusters have been merged

Distance Measures

- distance measure for clusters
  - single link: minimum of distances between all pairs of data points
  - complete link: e.g. mean of distances between all pairs of data points

Distance Measures

- distance measure for clusters
  - single link: minimum of distances between all pairs of data points
  - complete link: e.g. mean of distances between all pairs of data points
- local clustering criterion for data points: minimal mutual neighbor distance (MND)
  - distance depends also on the local context of a data point

$$d_{MND}(\vec{x}_i, \vec{x}_j) = r(\vec{x}_i, \vec{x}_j) + r(\vec{x}_j, \vec{x}_i)$$

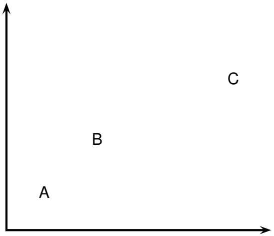
$r(\vec{x}_i, \vec{x}_j)$: rank of x_j according to distance from x_i

Partitioning Clustering

- mutual neighbor distance (MND)

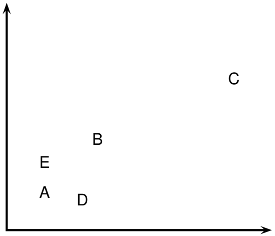
$$\begin{aligned} d_{MND}(A,B) &= r(A,B) + r(B,A) \\ &= 1 + 1 = 2 \end{aligned}$$

$$\begin{aligned} d_{MND}(B,C) &= r(B,C) + r(C,B) \\ &= 2 + 1 = 3 \end{aligned}$$



$$\begin{aligned} d_{MND}(A,B) &= r(A,B) + r(B,A) \\ &= 3 + 3 = 6 \end{aligned}$$

$$\begin{aligned} d_{MND}(B,C) &= r(B,C) + r(C,B) \\ &= 4 + 1 = 5 \end{aligned}$$



Partitioning Clustering

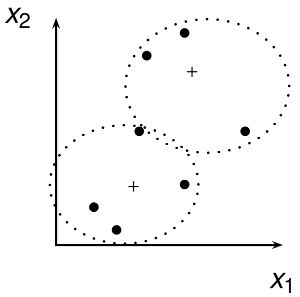
- vector quantization
- often used for data compression
- number of resulting clusters is given in advance

Partitioning Clustering

- vector quantization
- often used for data compression
- number of resulting clusters is given in advance
- each cluster is represented by a centroid

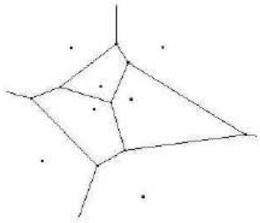
Partitioning Clustering

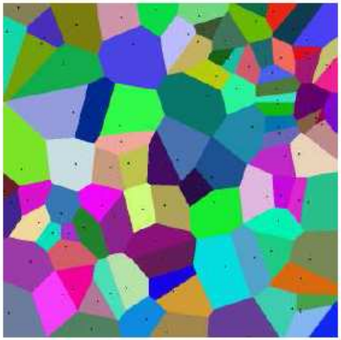
- vector quantization
- often used for data compression
- number of resulting clusters is given in advance
- each cluster is represented by a centroid



Partitioning Clustering

- complete segmentation of the feature space





VORONOI- oder DIRICHLET partitioning

Partitioning Clustering

- global clustering criterion: minimizing the mean square error
  - mean vector as centroid

$$\vec{c_k} = \frac{1}{n_k} \sum_{i=1}^{n_k} \vec{x_{ik}}$$

  - error for one cluster (within-cluster variation)

$$e_k^2 = \sum_{i=1}^{n_k} (\vec{x_{ik}} - \vec{c_k})^2$$

  - global error

$$e = \sum_{k=1}^{K} e_k^2$$

Partitioning Clustering

- algorithm for *k*-means partitioning clustering
  - select a randomly chosen initial partitioning with *k* clusters
  - compute the centroids
  - assign each sample to the nearest centroid
  - compute new centroids
  - continue until the clustering stabilizes (or another termination criterion based on the global error is met)

Partitioning Clustering

- computation of the prototype vectors $\vec{x}_i$

  $$\vec{x}_i = cent(i) = \arg\min_i E(d(\vec{x}, \vec{x}_i) | k(\vec{x}) = i)$$

  use the centroid of each cluster as prototype vector
  estimation of the mean
- vicious circle:
  - to determine the centroids a partitioning is needed
  - to partition the feature space the centroids are needed

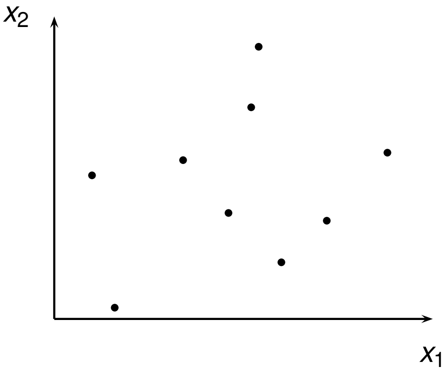
  $\rightarrow$ recursive approximation

Partitioning Clustering

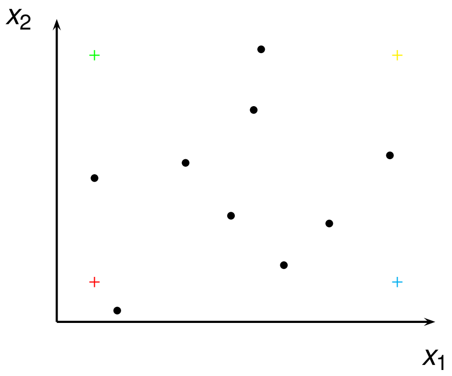
- partitioning of the feature space
  finding the optimal centroids for a given number of partitions:

  1. choose the initial centroids
  2. classify the sample according to the nearest-neighbor rule
  3. compute the similarity measure for each partition
  4. compute the a new centroid for each partition
  5. classify the sample according to the nearest-neighbor rule
  6. compute the similarity measure for each partition
  7. as long as the similarity measure improves, continue with 4

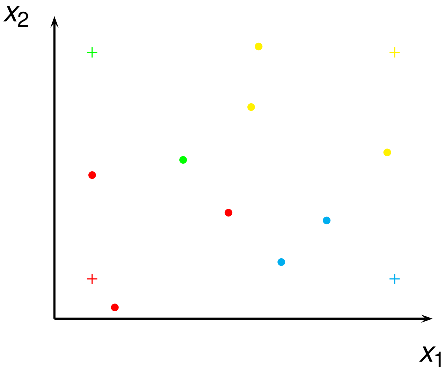
Partitioning Clustering



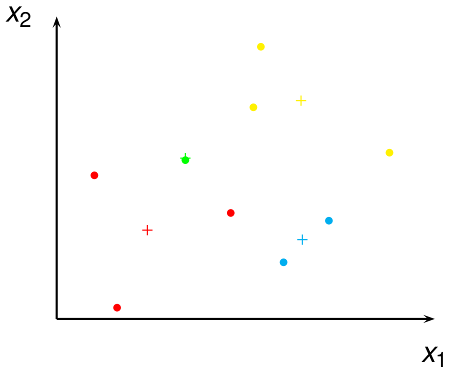
Partitioning Clustering



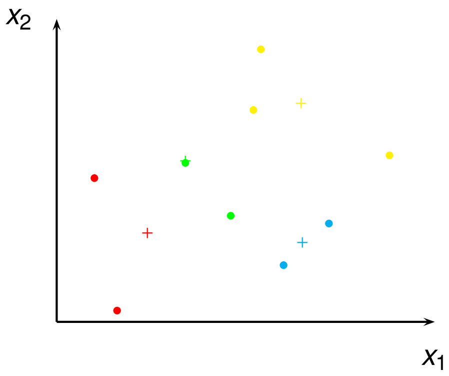
Partitioning Clustering



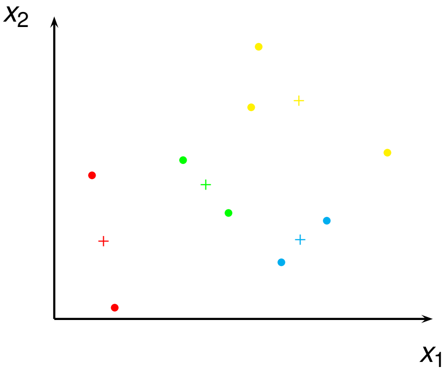
Partitioning Clustering



Partitioning Clustering



Partitioning Clustering

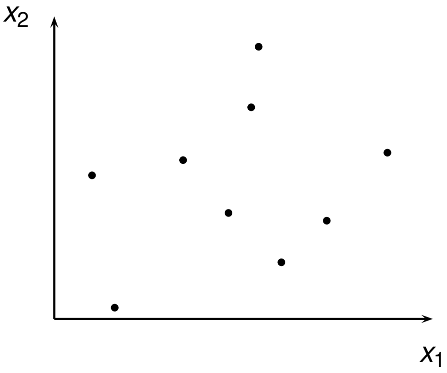


Partitioning Clustering

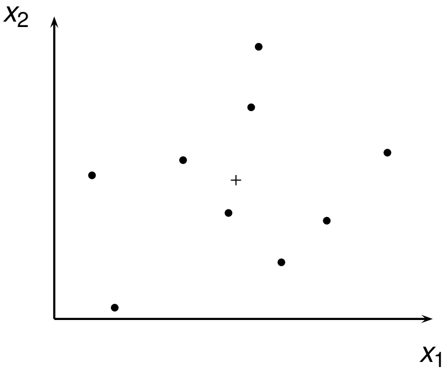
- finding a set of centroids by recursively splitting the partitions:

  1. start with a single centroid
     (e.g. the mean of all observations)
  2. split the centroid(s)
     - by a random distortion or
     - by choosing two points with maximum distance
  3. determine the optimal centroids (s.a.)
  4. required number of partitions not yet reached? continue with 2

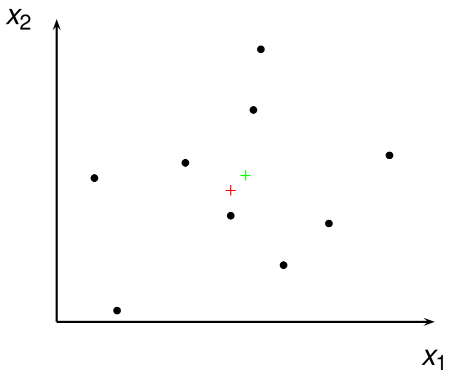
Partitioning Clustering



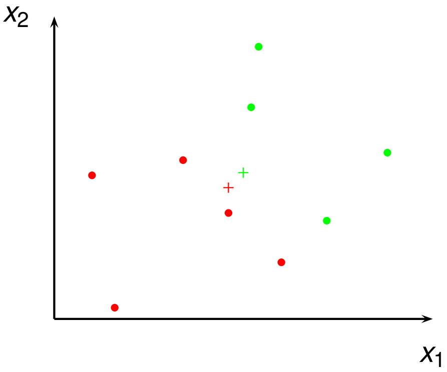
Partitioning Clustering



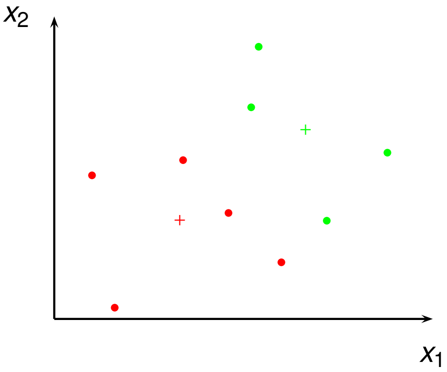
Partitioning Clustering



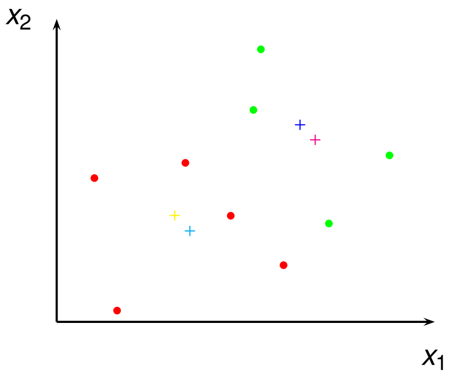
Partitioning Clustering



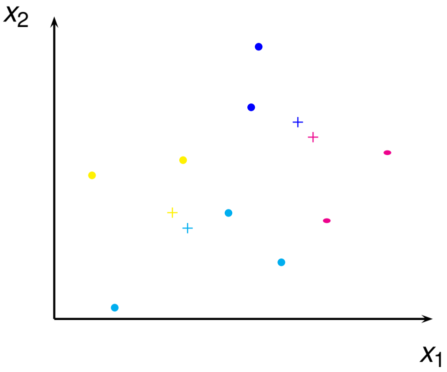
Partitioning Clustering



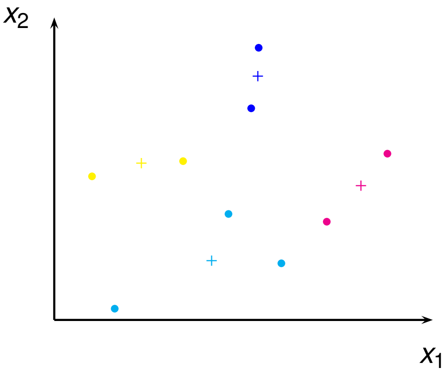
Partitioning Clustering



Partitioning Clustering



Partitioning Clustering



Partitioning Clustering

- additional heuristics for partitioning
  - if a partition has too many samples, divide it.
  - if two partitions are too close to each other, combine them.
- algorithms can also be adapted to incremental clustering