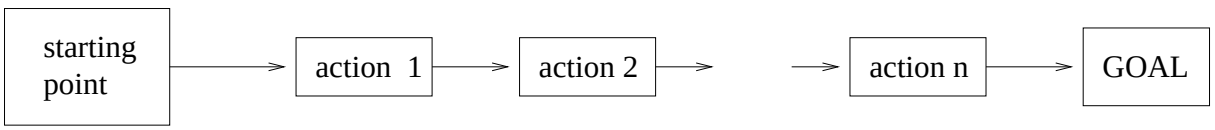
Summary

The main tasks of expert systems can be reduced to two main types:

1. Classification or regression: Given attributes $X_1, ..., X_k$, what is Y?
   (a) Classification: Y is class variable (discrete valued)
   (b) Regression: Y is numeric function value $Y = f(X_1, ..., X_k)$
2. Action selection: given situation and goal(s), what is the optimal action or sequence of actions?



Task: Invent examples of expert systems, which should learn a sequence of optimal actions!

Direction of inference:

- Prediction: Y is effect of X_i's (in future)
- Diagnosis: Y is cause of X_i's (in past)

Modelling paradigms

Three main techniques:

1. Logical rules
2. Numeric functions
3. Probabilistic reasoning

Rule-based systems

ES=Inference engine + knowledge base

Knowledge base

- =rules + most of facts
- Rules are of form antecedent $\Rightarrow$ consequent or "if condition then action"
- condition part is expressed as a logical formula with operators AND, OR, NOT
- Usually both factual and heuristic knowledge
- Heuristics=rules of good judgement, rules of good guessing, rules of thumb

An example rule from MYCIN:
"if the stain of the organism is gram negative AND the morphlogy of the orgasm is rod AND the aerobicity of the organism is anaerobic THEN there is a strongly suggestive evidence (0.8) that the class of the organism is Enterobacter iaceae"

Inference engine

- Searches through knowledge base and tries to match facts to the antecedents of rules.
- If the rule matches, the consequent can be executed.
- Several if–then rules are chained:

1. *Forward chaining*: starts from the data give and tries to find rules, whose premises can be satisfied. When a rule is satisfied, the conclusions are added to the knowledge base, and new rules are searched until all new information has been added.
2. *Backward chaining* (goal-oriented system): tries to prove a goal or rule conclusion by confirming truth of all its premises. The premise can be a conclusion of another rule, which has to be satified, or data supplied by the user from observations.

Advantages:

- easy to construct
- experts can easily interpret the answers
- can explain the reasoning process (gives confidence, easier to debug)

Restrictions:

- cannot represent *uncertainty*
  - real world phenomena are often complex and incompletely known
  - the knowledge can be imprecise or probabilistic
  - or expressed vaguely (”most”, ”several”, ”few”, ”often”, ”unlikely”,...)
- how rank alternative solutions?
- how evaluate confidence of conclusions?

$\Rightarrow$ **Solution**: uncertain rules, which express uncertainty by

- Fuzzy logic
  - The truth value of A, $T(A)$, is not 0 or 1 but $T(A) \in [0, 1]$
  - $T(A \wedge B) = min(T(A), T(B))$
  - $T(A \vee B) = max(T(A), T(B))$
  - $T(\neg A) = 1 - T(A)$
- Dempster-Shafer theory
  - Instead of a single value, we define an interval $[Bel(A), Pl(A)]$, where $Bel(A)$ tells how necessary A is and $Pl(A) = 1 - Bel(\neg A)$ tells, how possible A is (Notice that generally $Bel(A) + Bel(\neg A) \neq 1$)
  - Beliefs are combined by Dempster's rule
  - Flexible way to represent ignorance (lack of information), but sometimes the results seem counter-intuitive (if you are used to probability theory)
- Probability theory (see below)
- other measures

Truth-maintenance systems

- mathematically sound methods
- not in fashion nowadays, but some uncertain extensions have been introduced
- Machine learning algorithms for model construction do not exist? (yet!)
- the most natural implementation language is Prolog, which is not in fashion...

Main types:

1. Justification-based TMS

- The set of logical rules is represented as a graph.
- The nodes correspond to premises (always true), assumptions (which can be cancelled) or contradictions.
- The justifications are logical rules of form $p_1 \wedge p_2 \wedge ... \wedge p_n \rightarrow p$
- we can calculate a set of nodes which are believed, ask justifications for it, or cancel inconsistent assumptions

2. Assumption-based TMS

- similar graph structure with assumptions and justifications (rules)
- now we calculate for all nodes all minimal assumption sets, under which it is true
- more effective approach, when assumptions change often between queries
- enlargements for uncertain reasoning (with probabilities, Dempster-Shafer beliefs, or possibilistic logic)

Decision trees

- the decision rules are expressed in tree-form
- the whole model can be easily learnt from data
- can deal with both numeric and categorial data

Regression

- For predicting the *numeric* value of dependent variable Y given the independent variables $X_1, ..., X_n$.
- The goal is to learn a function f such that $Y = f(X_1, ..., X_n)$.
- The simplest model is *linear regression* $Y = \alpha + \beta_1 X_1 + ... + \beta_n X_n$
- More complex functions can be learnt by *neural networks*

Probabilistic reasoning

The basic laws of probability calculus

- $0 \leq P(A) \leq 1$
- If A can get k values $a_1, .., a_k$, then $\Sigma_{i=1}{}^k P(A = a_i) = 1$ (and for binary-valued data $P(\neg A) = 1 - P(A)$)
- Probability of A and B, $P(A, B) = P(A)P(B)$, if A is independent of B
- Probability of A or B is $P(A \vee B) = P(A) + P(B) - P(A, B)$
- The conditional probability of B given A is $P(B|A) = \frac{P(A,B)}{P(A)}$
- If we know $P(A)$ and $P(B|A)$, we can update the probability of A given B by *Bayes rule*: $P(A|X) = \frac{P(A)P(B|A)}{P(B)}$

Example

You don't remember, if your friend has a cat, but you are allergic to cats. In addition, you have hay-flu in summer time. In autumn, the probability of hay-flu is only 30%. You know that a cat alone causes you allergic symptoms with 60% probability and hay-flu alone with 70% probability. If there is both a cat and hay-flu, the probability is 90%. What is the probability that you get allergic symptoms, when you visit your friend?

Let's denote A=Your friend has a cat B=You have hay-flu, C=You get allergic symptoms.

$P(A) = 0.5$ (You don't know it)
$P(B) = 0.30$
$P(C|A, B) = 0.90$
$P(C|A, \neg B) = 0.60$
$P(C|\neg A, B) = 0.70$
$P(C|\neg A, \neg B) = 0.10$ (This is rare, but not impossible – other factors can cause allergic symptoms, too!)

Now the probability of allergic

$$P(C) = P(A)P(B)P(C|A,B) + P(A)P(\neg B)P(C|A,\neg B) + P(\neg A)P(B)P(C|\neg A,B) + P(\neg A)P(\neg B)P(C|\neg A,\neg B) =$$

You cannot see a cat – what is the probability of cat, if you get allergic symptoms?

$$P(A|C) = \frac{P(A,C)}{P(C)} = \frac{P(A,B,C)+P(A,\neg B,C}{P(C)} = \frac{P(A)P(B)P(C|A,B)+P(A)P(\neg B)P(C|A,\neg B)}{P(C)} =$$

Task: Calculate the probabilities!

Notice! If we would know the total probability distribution $P(A,B,C)$ (for all value combinations), we could solve everything directly. However, in practice, we can usually estimate only the conditional probabilities.

Bayesian networks

- A nice visual representation for probabilistic relations
- Vertices correspond to variables and edges dependencies between variables (if variables are independent, there is no edge!)
- Each root node A has prior probabilities $P(A)$ (for all values a_i)
- Each non-root node has a table of conditional probabilities $P(B|X)$, where X is the set of antecedent nodes (for all value combinations of X)
- We can predict the probability of future events simply by chaining probabilities
- The probability of a cause A can be updated when events X have been observed by Bayes rule

Neural networks

- Idea is to learn a function, which maps inputs to outputs (general regression).
- In principle, can express any function, but it can be hard to learn!
- The system consists of nodes, which calculate the weighed sum of inputs (from antecedent nodes), apply some function, and output the result to successor nodes

Main types:

- Feed-forward networks: the data flows from inputs to outputs
  - single-layer = perceptron: simple, but restricted
  - multi-layer: very expressive, but hard to learn and interpret
- Recurrent networks: for modelling time series data (now we take into account also the value of the node in the previous time step)

Instance-based methods

- do not built any explicit model
- Assumption: the data point behaves in the similar way than the "neighbouring" points
- How to define neighbours?

k-nearest neighbours

- for numeric data
- A simple approach: if $d_1, .., d_k$ are k nearest neighbours to d according to X_i variables, then Y is weighed average of $Y_1, .., Y_k$
- Problem: how to define neighbours? Several possible distance functions (and Euclidean is not always the best!)

Several variations for non-numeric data, e.g.

- *Case-based* systems: All solutions are stored in a database. Similar cases are searched from the database and the solutions are applied to new situations or new solutions are generated from the old ones.
- *Social or collaborative filtering* (usually recommending systems, not necessarily expert systems in classical sense)

Genetic algorithms

- simulates evolution process
- we keep a set of solutions (population), which evolves through selection, crossing and mutation operations
- can be slow, but achieve good results for complex problems, which are hard to model otherwise
- e.g. learning the set of optimal action rules for a robot

Which paradigm to select?

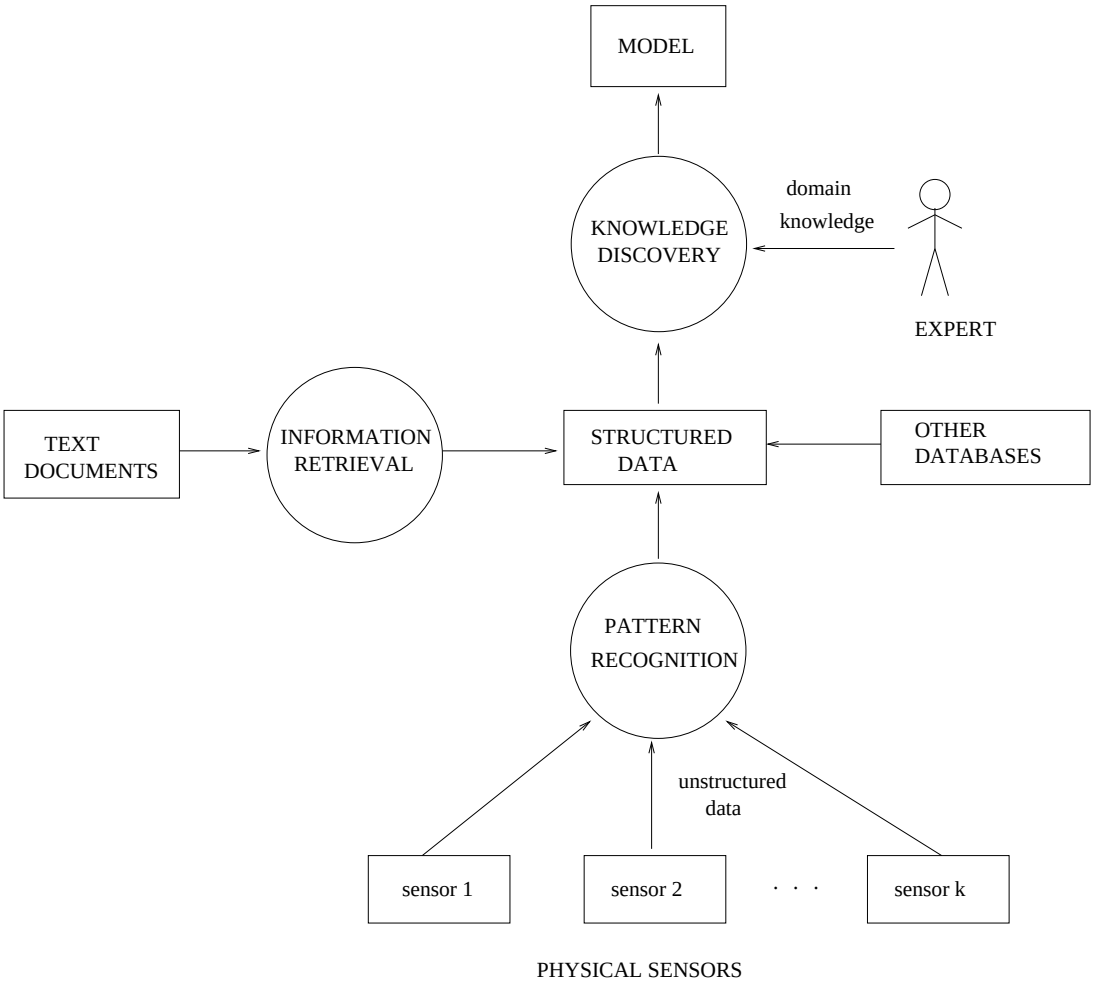
- There is no one super system for all problems!
- The most appropriate system depends on the problem
- The systems come into fashion and go out of fashion

Abstraction level?

Does the system work with

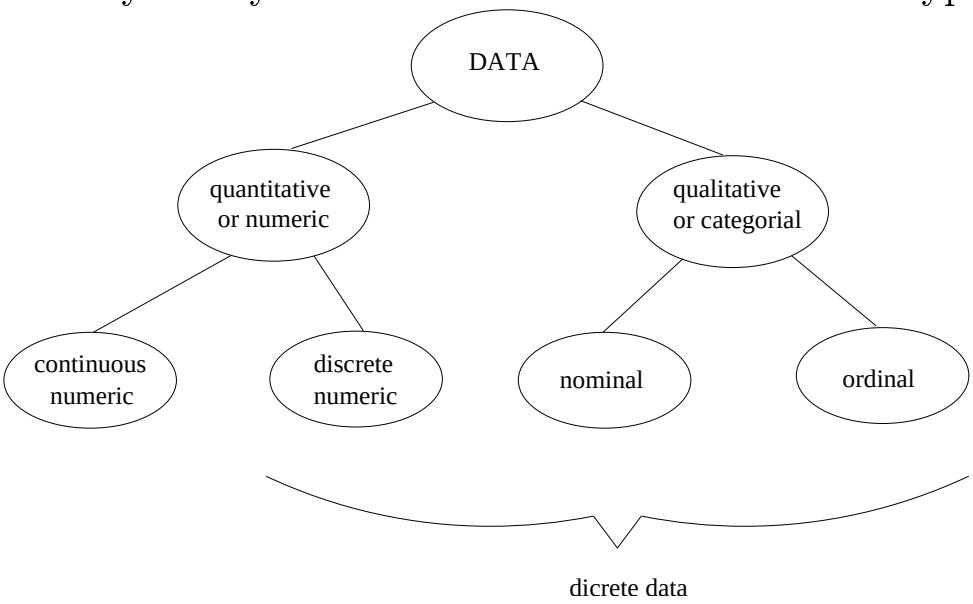
- high-level data (structured, e.g. relational database)
- low-level data (unstructured sensor data, e.g. sounds, images)
- or both?

Traditionally only high-level data $\Rightarrow$ the expert can define the model and understands it!
Task: Invent applications of expert systems, which use only low-level data!


MODEL
KNOWLEDGE DISCOVERY
domain knowledge
EXPERT
TEXT DOCUMENTS
INFORMATION RETRIEVAL
STRUCTURED DATA
OTHER DATABASES
PATTERN RECOGNITION
unstructured data
sensor 1
sensor 2
. . .
sensor k
PHYSICAL SENSORS

Basic datatypes

Usually the system is restricted to certain data types!



The data can be classified according to type of attribute domain:

- *numerical* (*quantitative*) data: has meaning as numbers: we can measure order, distance and equality of two numerical variables.
  - *discrete* data: can get only countably many values (Notice: in computer systems, all the data is represented with some precision and is thus discrete) (e.g. age=$\{1, ..., 120\}$)
  - *continuous* data can in principle get any value in the given interval (e.g. weight ≥ 0)
- *categorial* (*qualitative*) data: has no meaning as numbers; generally we can measure only equality between two categorial variables (e.g. 1=female, 2=male)
  - *nominal* data: the values are only names for categories and have no other meaning as categorizing
  - *ordinal* data: the values have a natural ordering, but the differences have no meaning (e.g. 1=poor, 2=fair, 3=good, 4=excellent)

Temporal or static data?
In temporal data we can find dynamic patterns (how the system evolves in time)
Amount of data:
If we have too little data (relative to number of variables), the parametres cannot be estimated accurately!

Corrupted data:

- The data may contain errorneous or missing values
- The measurements can cause variation (noise)
- Outliers: non-typical data points with extreme values

$\Rightarrow$ Solutions:

- enough data
- robust models (=not sensitive to small variations in data)
- use of expert's knowledge

Task: Find out, what means *overfitting* and how it can be prevented!

Desirable characteristics

Typically, we wish that the model

- can be learnt from data
- can integrate expert's knowledge
- reasons in a consistent way and detects incosistencies (e.g. conflicting rules, circular rule chains, illegal values)
- can be updated in the light of new evidence
- can handle noisy data (missing, spurious or errorneous)
- meets time-requirements with the given data (works in real time, if needed)