Knowledge-Based Systems (KBS)

Machine Learning and Knowledge Engineering

Winter semester 2021/2022

Prof. Dr. Martin Atzmüller

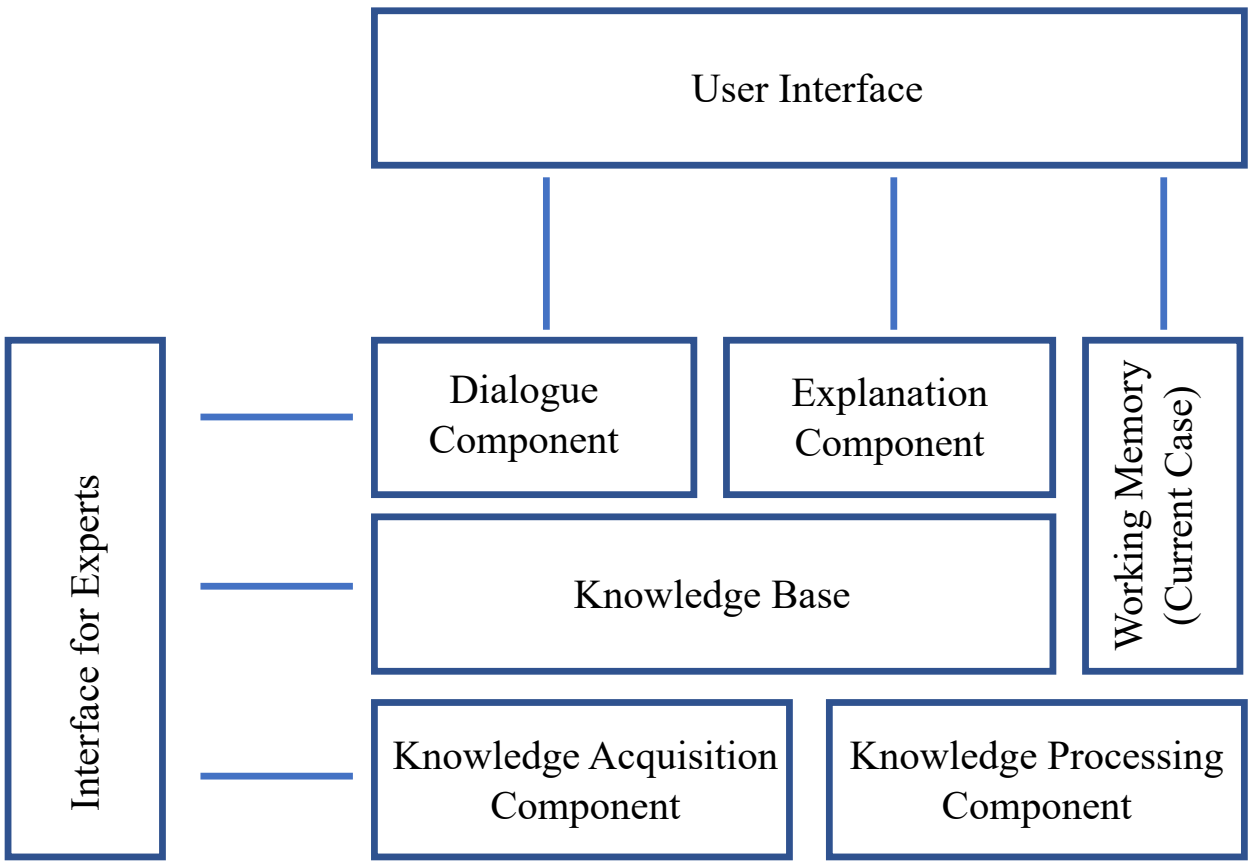
Osnabrück University & DFKI
Semantic Information Systems Group
https://sis.cs.uos.de/

3 – Knowledge Acquisition & Machine Learning

1. Knowledge Acquisition & Knowledge-Based Systems
2. Knowledge Acquisition – Models/Frameworks
3. (Semi-)Automatic Knowledge Acquisition
4. Intro/Overview: Machine Learning
5. Machine Learning Basics

Knowledge-Based Systems – Overview/Contents

KBS – Basic Architecture



Steps for Building Knowledge-Based System

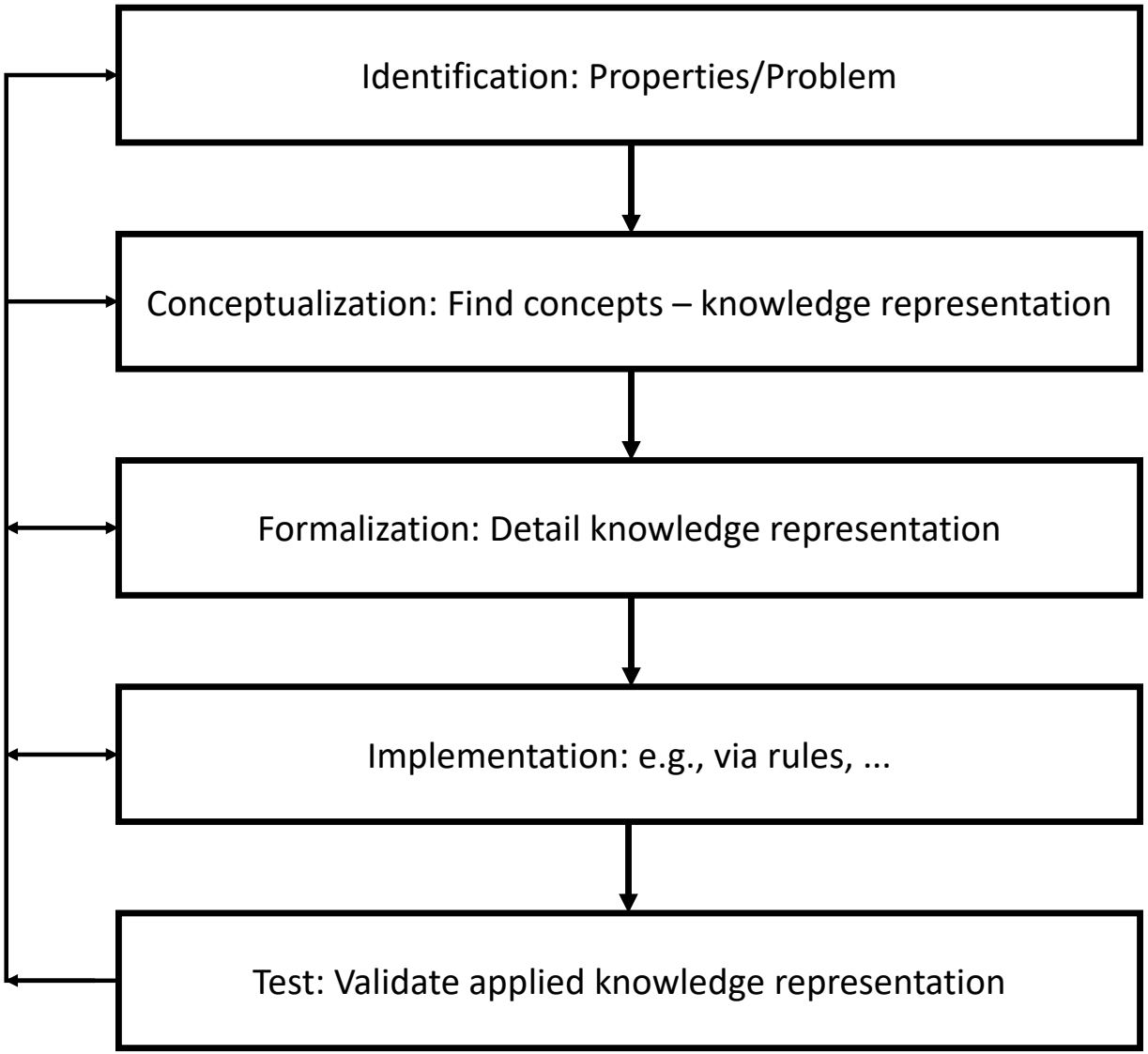
1. Problem description
2. Identification of knowledge sources
3. KBS design (components: inference, explanation, UI, ...)
4. KBS development tool(s)
5. Prototype development (Rapid Prototyping)
6. Test of the prototype
7. Refinement and generalization
8. Maintenance and support

(cf. Beierle & Kern-Isberner 2019)

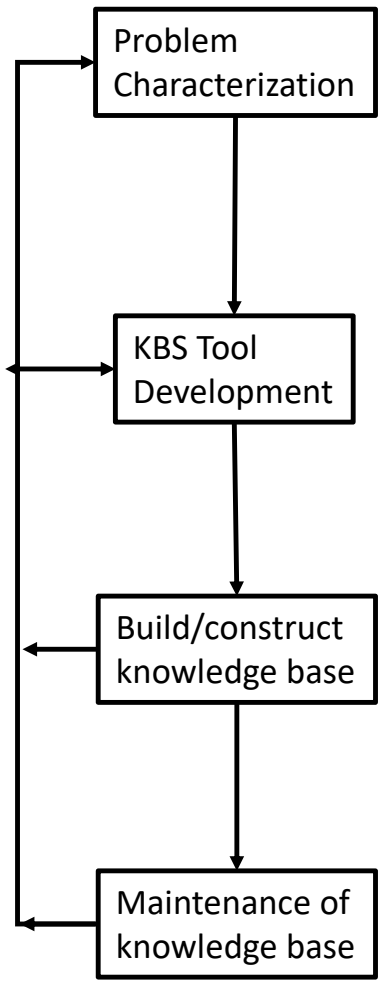
KA & Knowledge-Based Systems

- Can be seen as a (complex) software engineering task
- N.B.: There are various similarities (and relationships) between software engineering and the engineering of a knowledge-based system (e.g., agile development approaches)
- Core problem: Knowledge acquisition (KA)
- The so-called "Knowledge Acquisition Bottleneck": "extracting knowledge from human experts", but also "creating knowledge-based systems"
- KA main options:
  - Indirect KA: Interviews (formalization by knowledge engineer)
  - Direct KA: Experts formalize knowledge by themselves
  - Automatic KA: KBS extracts knowledge itself given suitable data and information, e. g., cases and/or text (literature)

KA Basic Phase Model



KA Concrete Phase Model

Phases	Description	Implementation	Tools
	Identification of problem solving method and knowledge representation	Knowledge Engineer consults expert	Interview methods; protocols
	Provide KBS tool for KBS development (with KA component)	Selection/development by knowledge engineer	General tool: • Knowledge acquisition • Knowledge engineering • Programming Language
	Formalization of (expert) knowledge	Expert, supported by knowledge engineer (if needed); potentially also using automatic KA methods	Specialized KBS Tool
	Tuning, Refinement, Maintenance; Updates (changes in requirements)	Expert; potentially also using automatic KA methods	Database; Case base

KA and Basic Knowledge Elicitation

- Interview: Interview session
  - Structured
  - Unstructured
- Self report/protocol analysis:
  $\rightsquigarrow$ "Thinking aloud" while solving a problem
- Laddering:
  $\rightsquigarrow$ Organize (important) entities in hierarchy (e. g., class, process, relation)
- Concept sorting
  - Set of "concept cards"
  - Find relations
- Repertory grid:
  - Construct theory – find concept relations
  - Consider concept triads (Mercury vs. Venus vs. Jupiter)
  - Find discriminating features: e. g., planet size

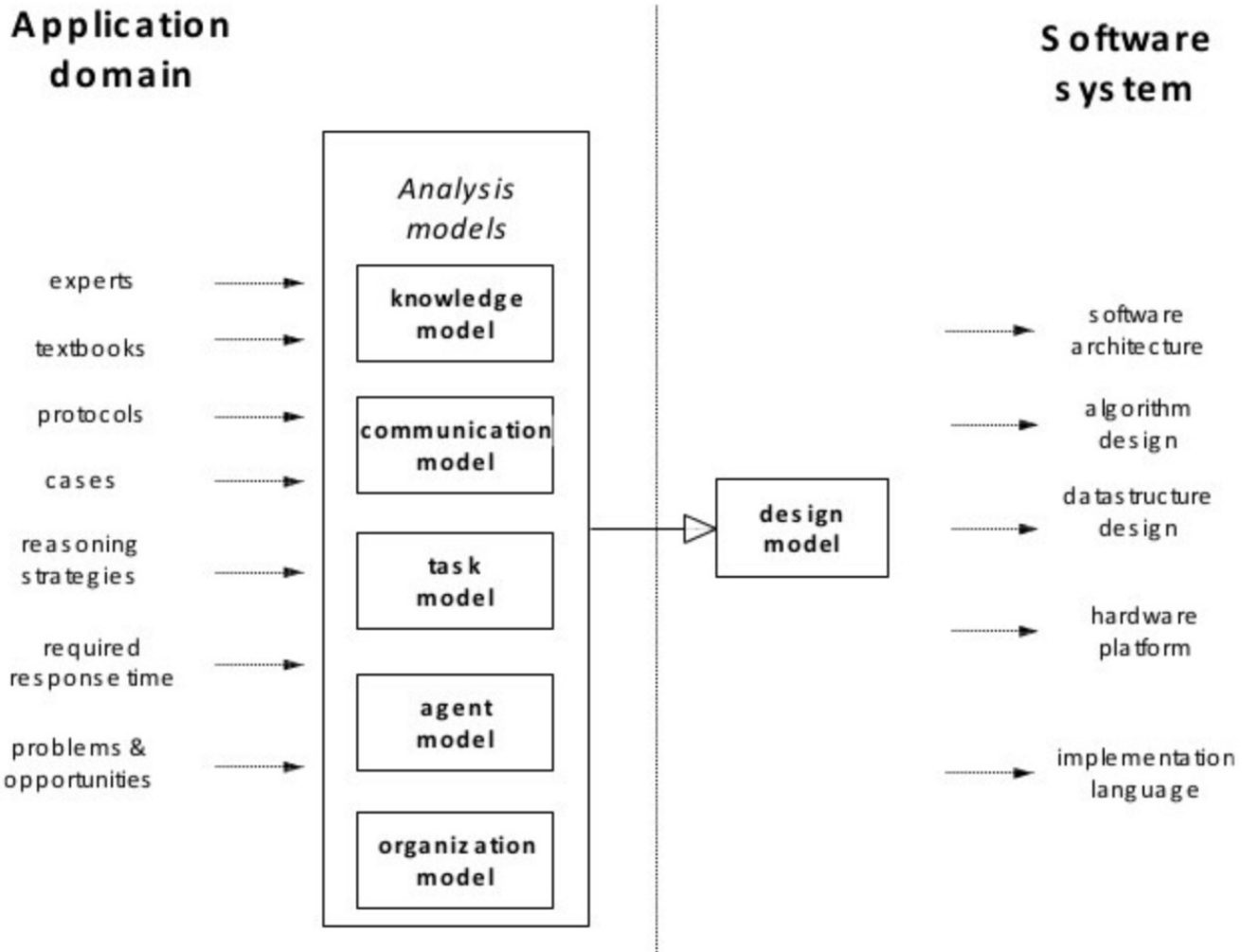
Basic Knowledge Elicitation – KBS Construction

- Knowledge identification:
  - Unstructured interview
  - Laddering
- Knowledge specification:
  - Domain schema: Concept sorting, repertory grid
  - Template selection: Self report
  - Knowledge about inference process: Self report
- Knowledge refinement:
  - Structured interview
  - Specific case(s) (reasoning prototypes)

CommonKADS

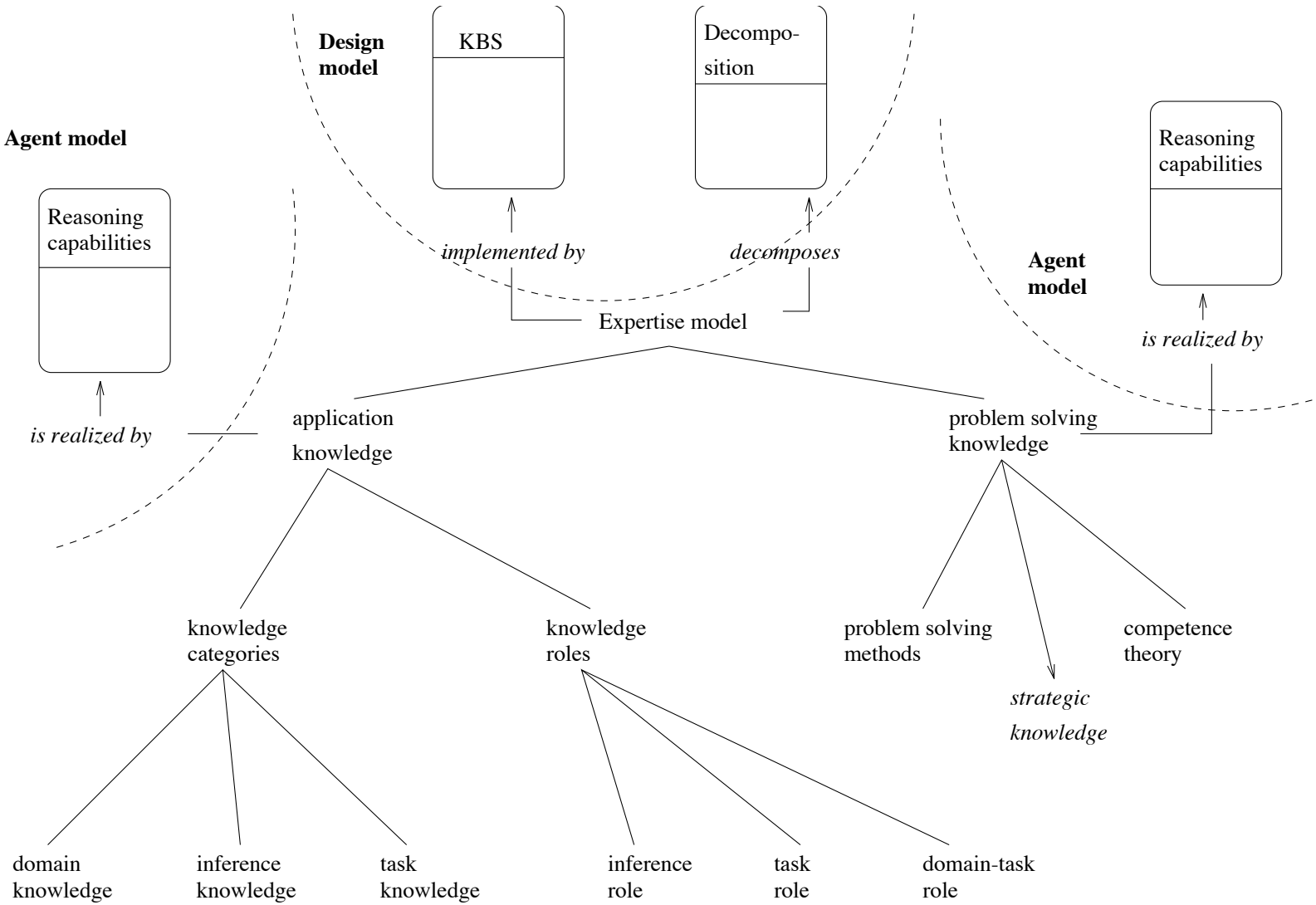
- Approach for structured knowledge engineering
- Developed in the European ESPRIT IT Programme (developed over some 15 years ...)
- Textbook: "Knowledge Engineering and Management: The CommonKADS Methodology" Schreiber et al. 1999, MIT Press
- Similar to object-oriented modeling methodology
- Distinguishes between different models (analysis models) which are then used for implementation (design model) of the respective software system
  - Knowledge model
  - Communication model
  - Task model
  - Agent model
  - Organization model

CommonKADS – Overview/Modeling



(CommonKADS, Schreiber et al. 1999)

CommonKADS – Components

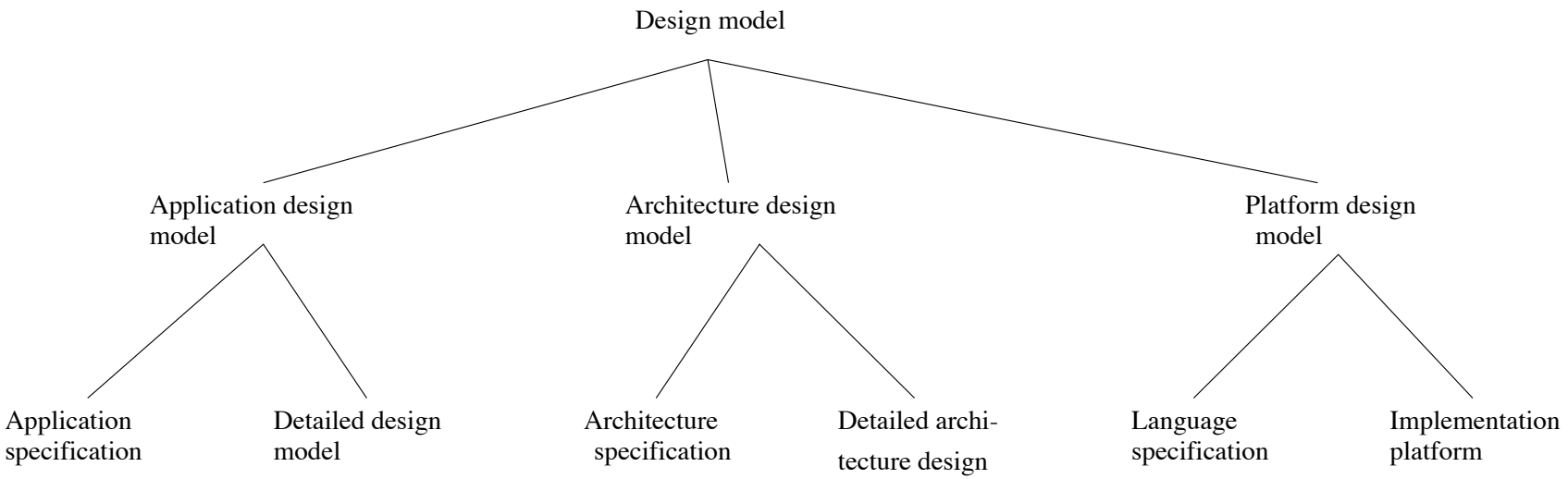


(CommonKADS, Schreiber et al. 1999)

CommonKADS: Expertise Model

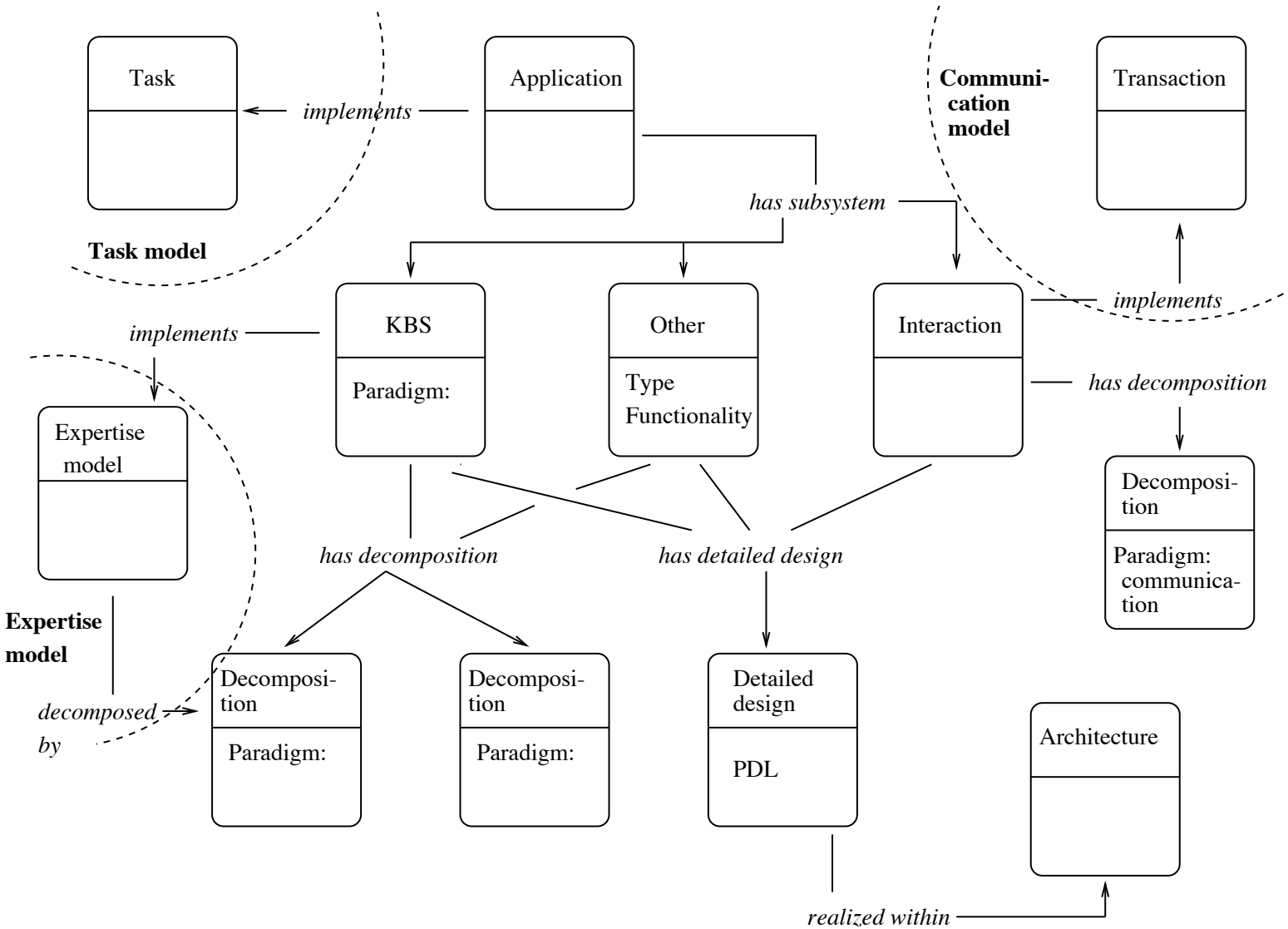
- Application knowledge:
  - Domain knowledge: What is known about the application domain, what a task is about; domain ontology, domain model
  - Task knowledge: Specifies tasks; goal, activities
  - Inference knowledge: Basic inferences using the domain knowledge
- Problem solving knowledge:
  - Problem solving methods
  - Competence theory: required competence of the problem solver
  - Strategic knowledge: inference/task aspects of problem solving knowledge

CommonKADS – Design & Implementation



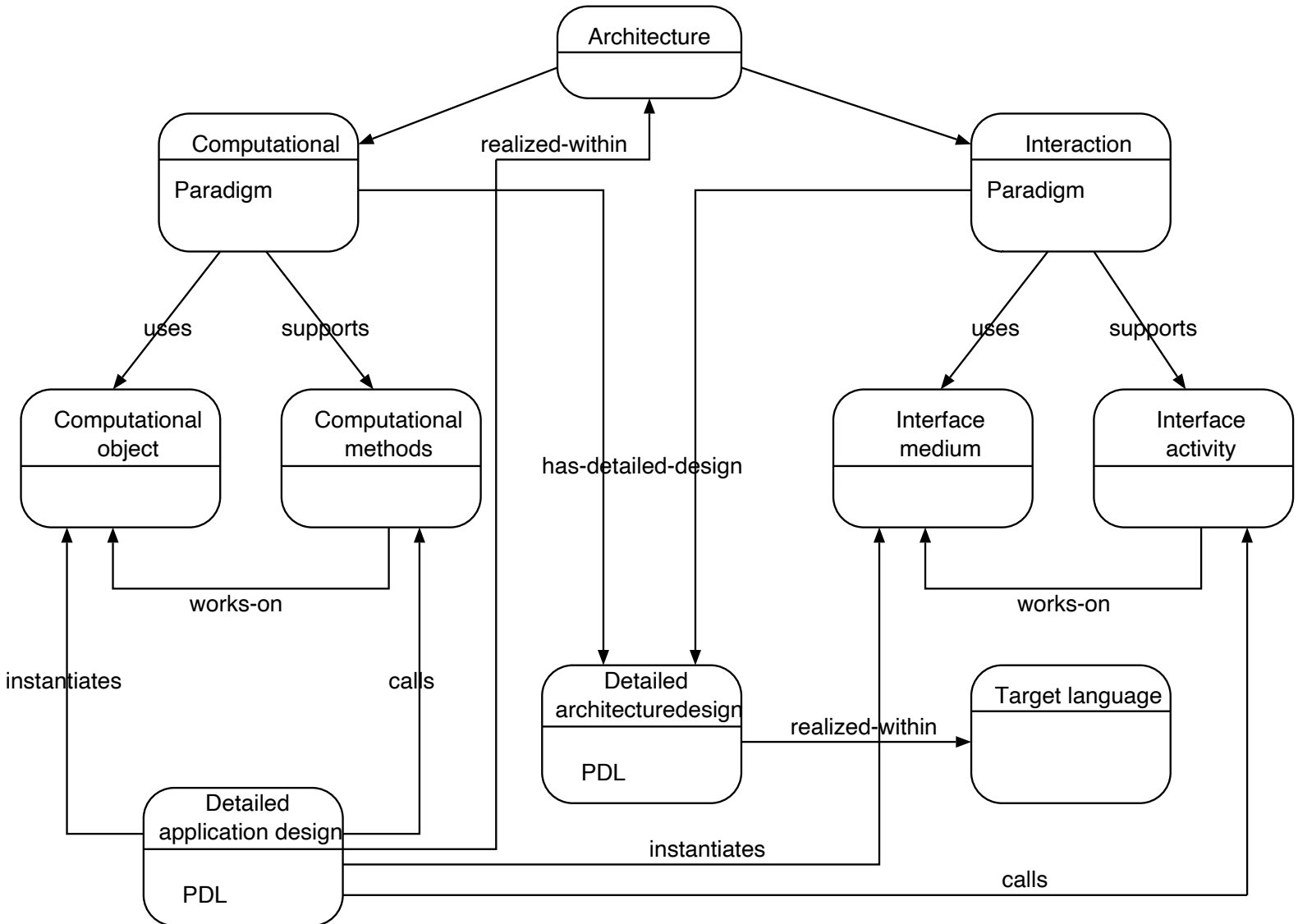
(CommonKADS, Schreiber et al. 1999)

CommonKADS – Detailed Design



(CommonKADS, Schreiber et al. 1999)

CommonKADS – Implementation, Architecture



(CommonKADS, Schreiber et al. 1999)

Knowledge Acquisition – Agile Process Model

Motivation:

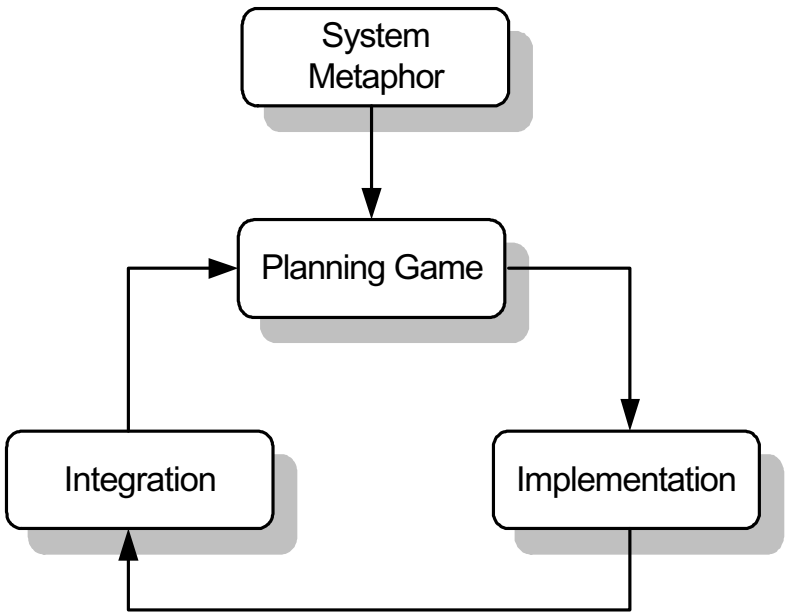
- Standard development models exist, KBS deployed into (very many) real-world applications
- Development & maintenance of KBS still difficult:
  - Costly to develop (required time/resources not predictable)
  - Systems show unexpected/faulty behavior in routine use
  - Sometimes systems do not fulfill requirements/expectations
  - Some systems hardly maintainable in routine use
- $\rightsquigarrow$ Standard process models: CommonKADS
- In Software Engineering, recently:
  - Agile process models
  - Extreme programming (XP)

(cf.. Baumeister et al. (2004) An Agile Process Model for Developing Diagnostic Knowledge Systems. KI Journal 3:12–16)

Agile Process Model

Properties:

- Early, concrete, and continuing feedback
- Incremental planning approach
- Flexible schedule of the development process
- Design process lasts as long as the system lasts



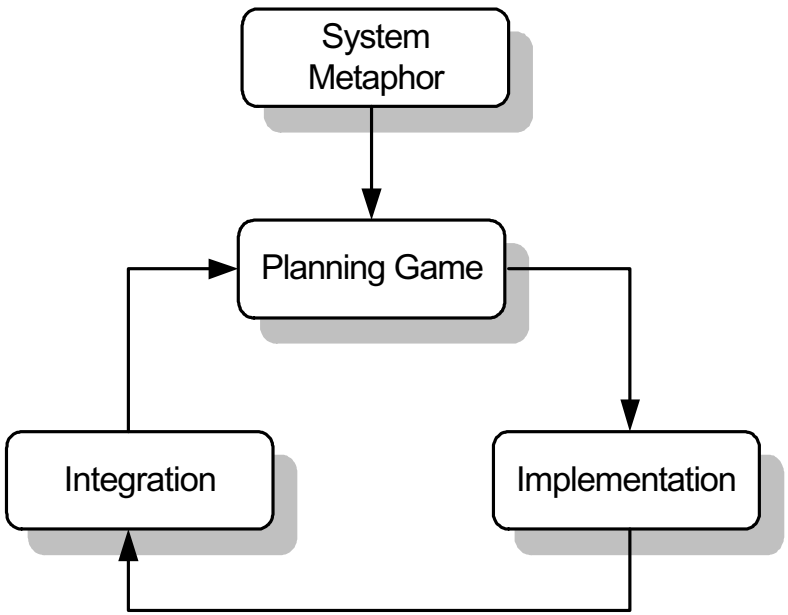
System Metaphor

Local system metaphor: Structuring, e. g., into diagnoses, finding, problem case etc. for diagnostic knowledge systems
Global system metaphor, type of application class e. g.,

- Documentation system: "A documentation system focuses on a high quality data acquisition and usually implements a detailed dialog control. This kind of class is often implemented, when a high quality data entry is the most important feature of the knowledge system. [...]" (Baumeister et al. 2004)
- Embedded system: "Sometimes the diagnostic knowledge system is embedded into a larger context. Then, the system is tightly connected to another application, from which it receives its findings and to which it reports the derived diagnoses. [...]" (Baumeister et al. 2004)
- Consultation system: "Consultation systems are the typical kind of a diagnostic knowledge system: [...] the system applies the findings to infer diagnoses which are presented as solutions to the user. [...]" (Baumeister et al. 2004)
- Information Center: "The information center does not consider a well elaborated diagnostic inference (e.g., as needed for a consultation system), but mostly consists of informal knowledge like documents, multimedia content or structured cases. Usually, this content is highly structured and methods for intelligent retrieval and navigation are available. [...] In comparison to a consultation system the user is more flexible and independent, but also responsible for obtaining a suitable solution for his problem." (Baumeister et al. 2004)

Agile Process Summary

1. Planning game: Document plans via set of story cards; capture task, implementation time, additional notes.
2. Implementation: Putting stories into tasks; *test-first approach* – Unit-Tests (!)
3. Integration using *integration tests* when putting the KBS into production



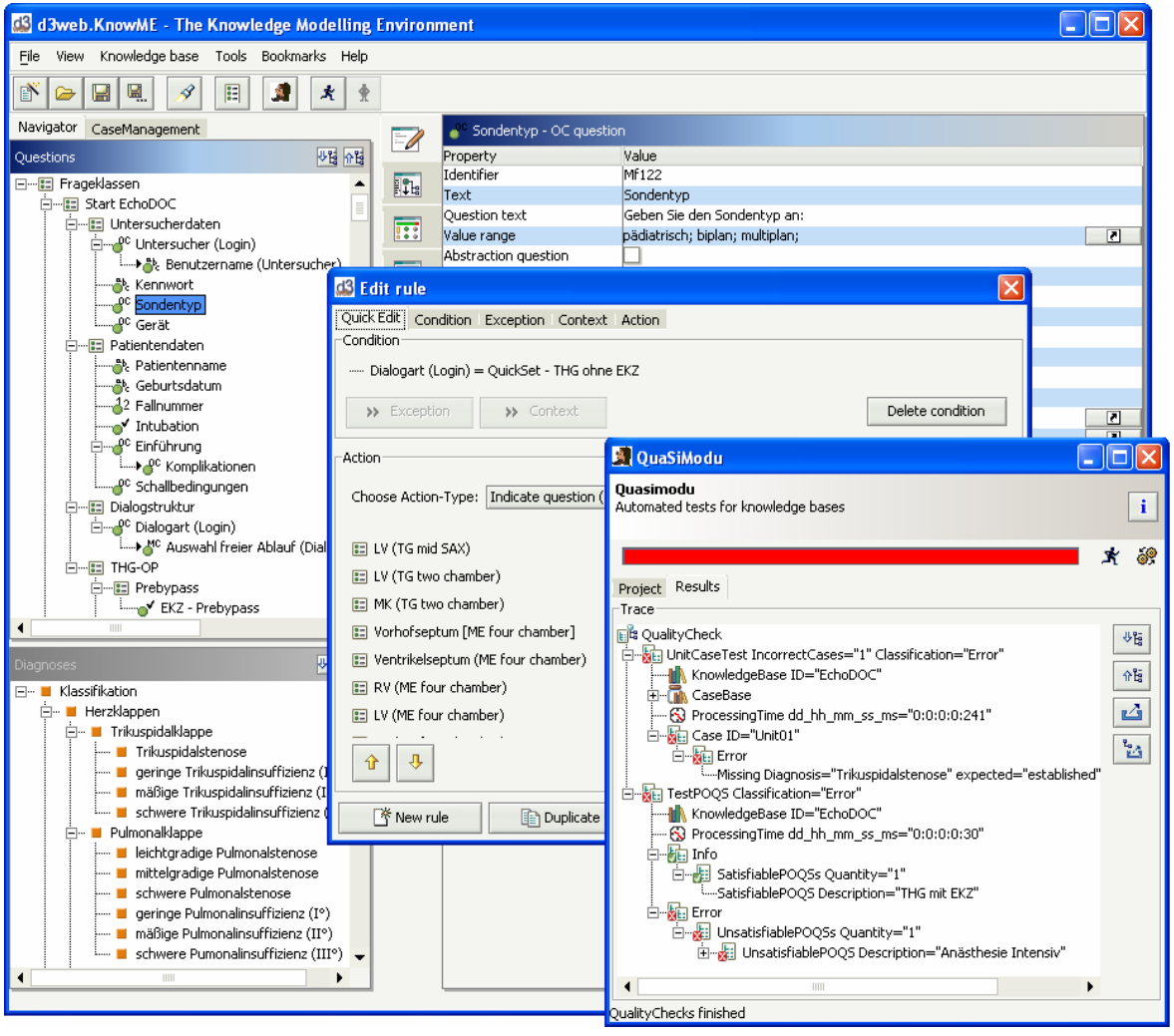
Agile Process – Knowledge Containers

- Ontological knowledge: Basic entities used in the KBS
- Structural knowledge: Inference – deriving a solution for a given input
- Strategic knowledge: Controlling the user dialog of the KBS (e. g., in terms of asking questions, regarding findings etc.)
- Support knowledge: Supply ontological knowledge with additional information (e. g., text book entries, additional (multi-media) information etc.)

Structure of knowledge containers helps in developing KBS by

- Providing an organized view
- Simplify maintenance/validation by modularization
- Simplify planning game by modularization

Agile Process – Automated Tests

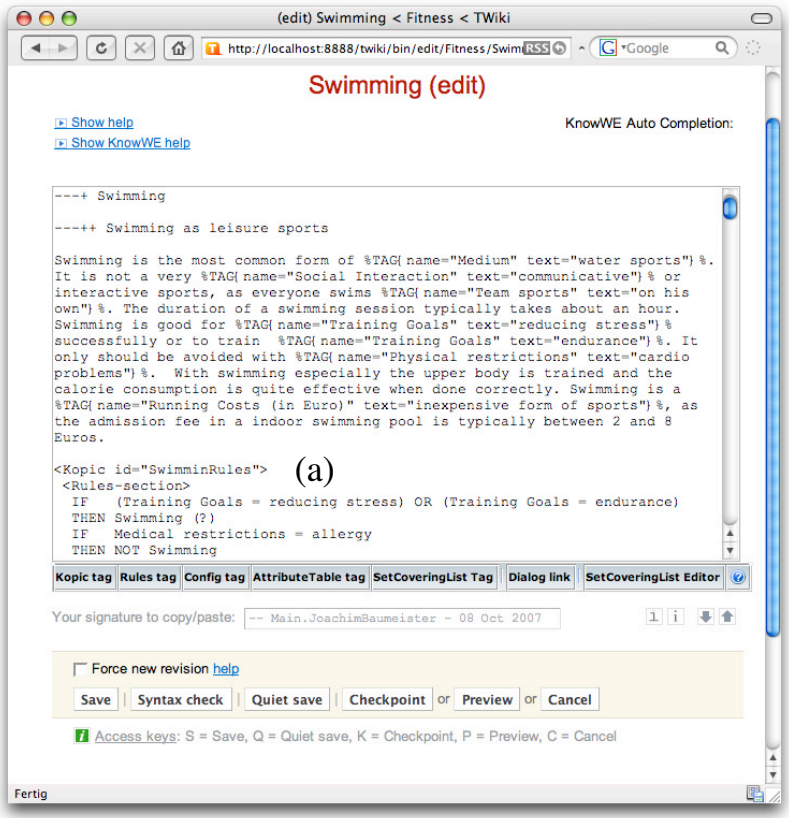


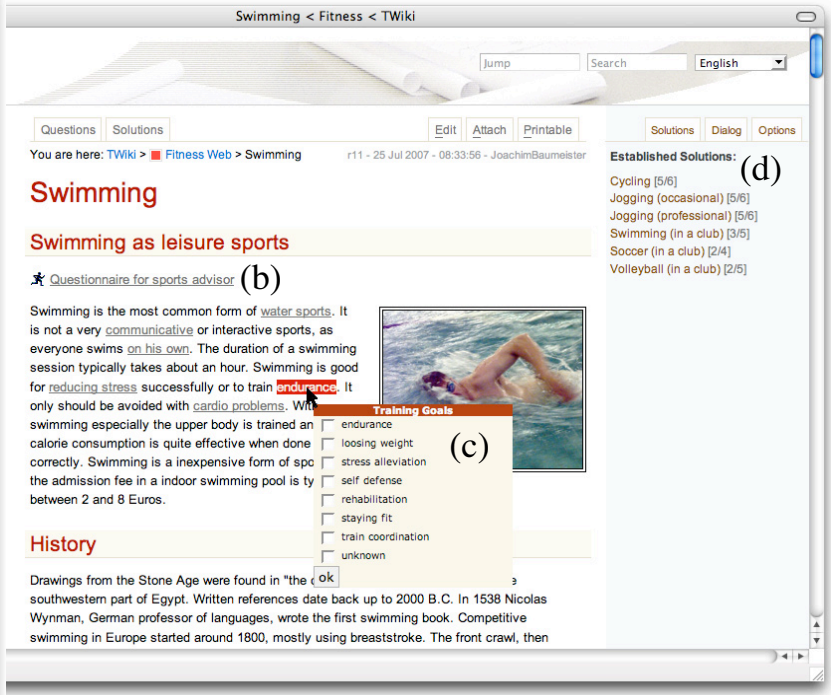
Example: Tests in d3web.KnowME, cf. (Baumeister et al. 2004)

KA using Semantic Wikis

- Semantic Wikis $\rightsquigarrow$ Knowledge Wikis
- Collaborative knowledge engineering
- Knowledge capture at different levels of detail (text, formal representation, rules, ...)
- Knowledge base is entered together with the standard text in the usual edit pane of the Wiki
- Semantics: Via markup text
- Using appropriate knowledge markups ("tags") knowledge can be captured, and compiled to a respective knowledge representation (e. g., rules)
- Markups should be intuitive and understandable even for inexperienced users

Knowledge Wiki – Example: Swimming



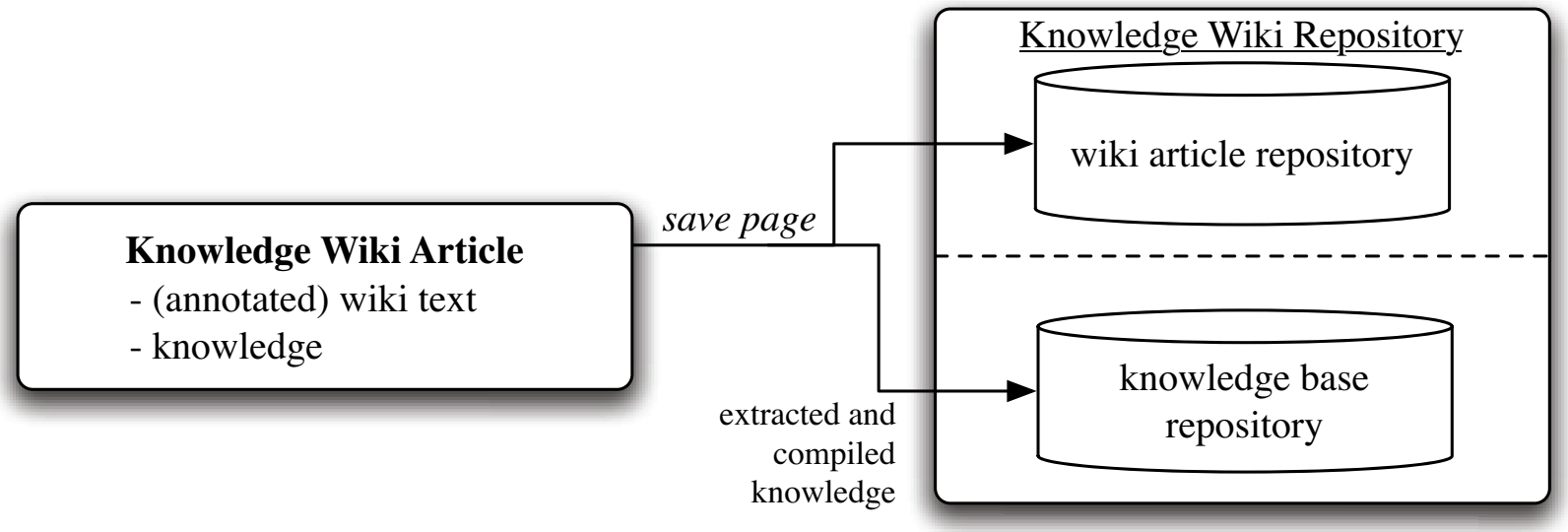


Example: Article for swimming; textual description and derivation rules in the respective wiki pages (cf. Baumeister et al. (2008) Web-based Knowledge Engineering with Knowledge Wikis. AAAI)

Knowledge Wiki – Workflow

Workflow:

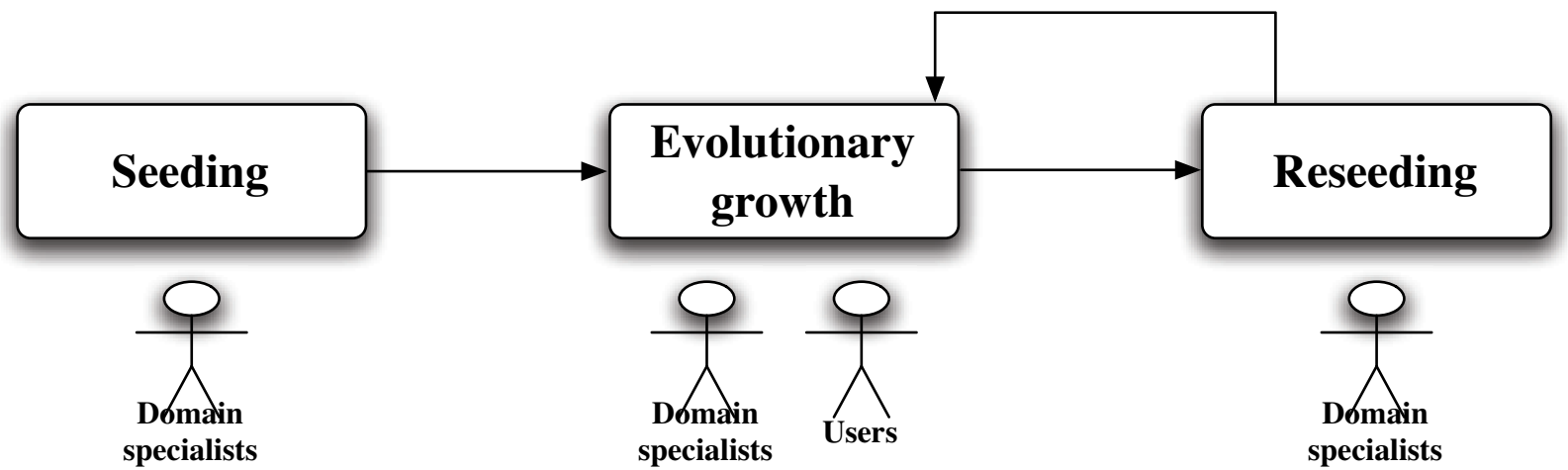
- New content is entered via the edit interface of the Wiki system (on the respective page)
- Explicit knowledge: Entered via markups
- Example: New solution added via new page in the system
- By saving the page, the explicit knowledge is extracted, compiled, and stored in the knowledge base



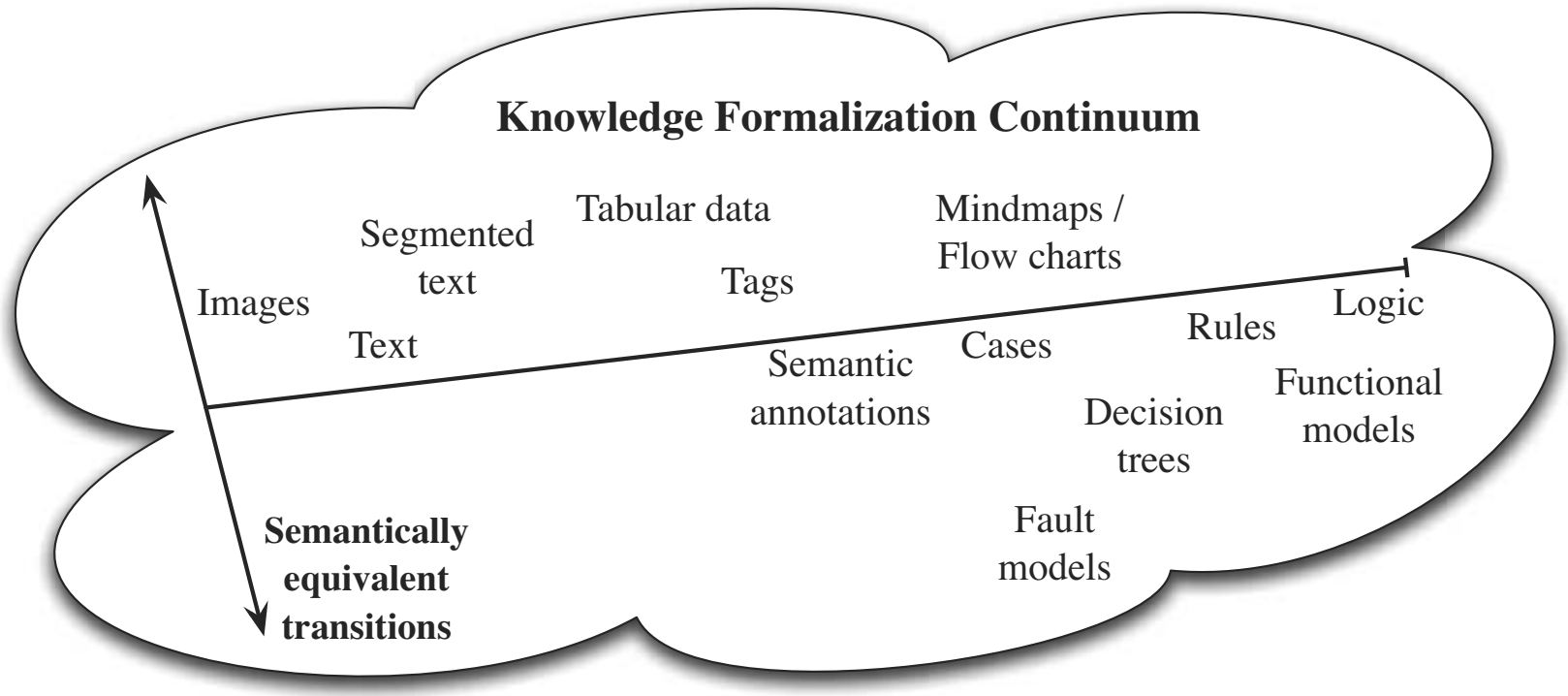
Knowledge Wiki – Evolutionary Process Model

Workflow:

- Seeding: Start with "simple" information like text and images
- Evolutionary growth: Domain specialist (expert) extends this gradually, and also by users (refinement)
- Reseeding:
  - Reorganization (e. g., too much unorganized content)
  - Domain specialists perform refactorings/reorganization, modification of the structure of the Knowledge Wiki

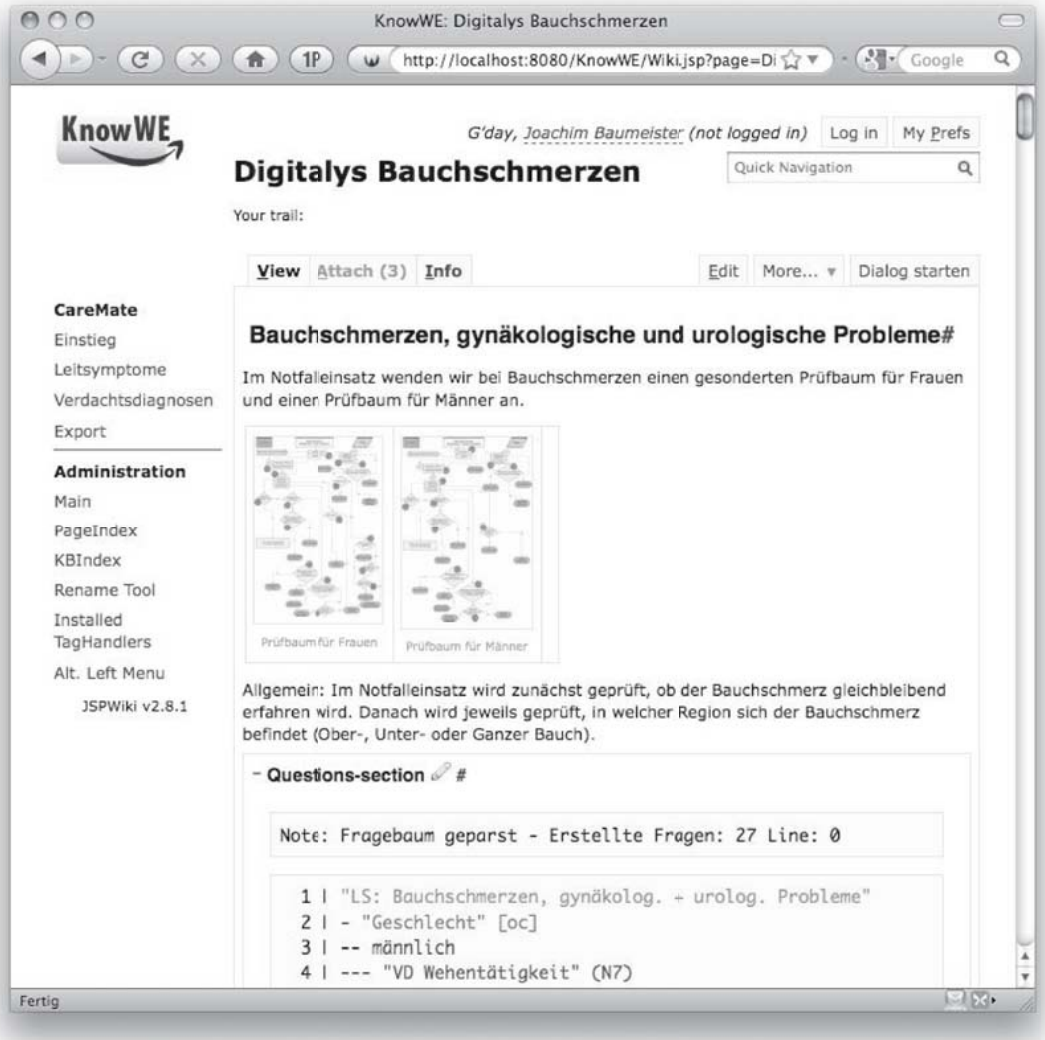


KBS & The Knowledge Formalization Continuum



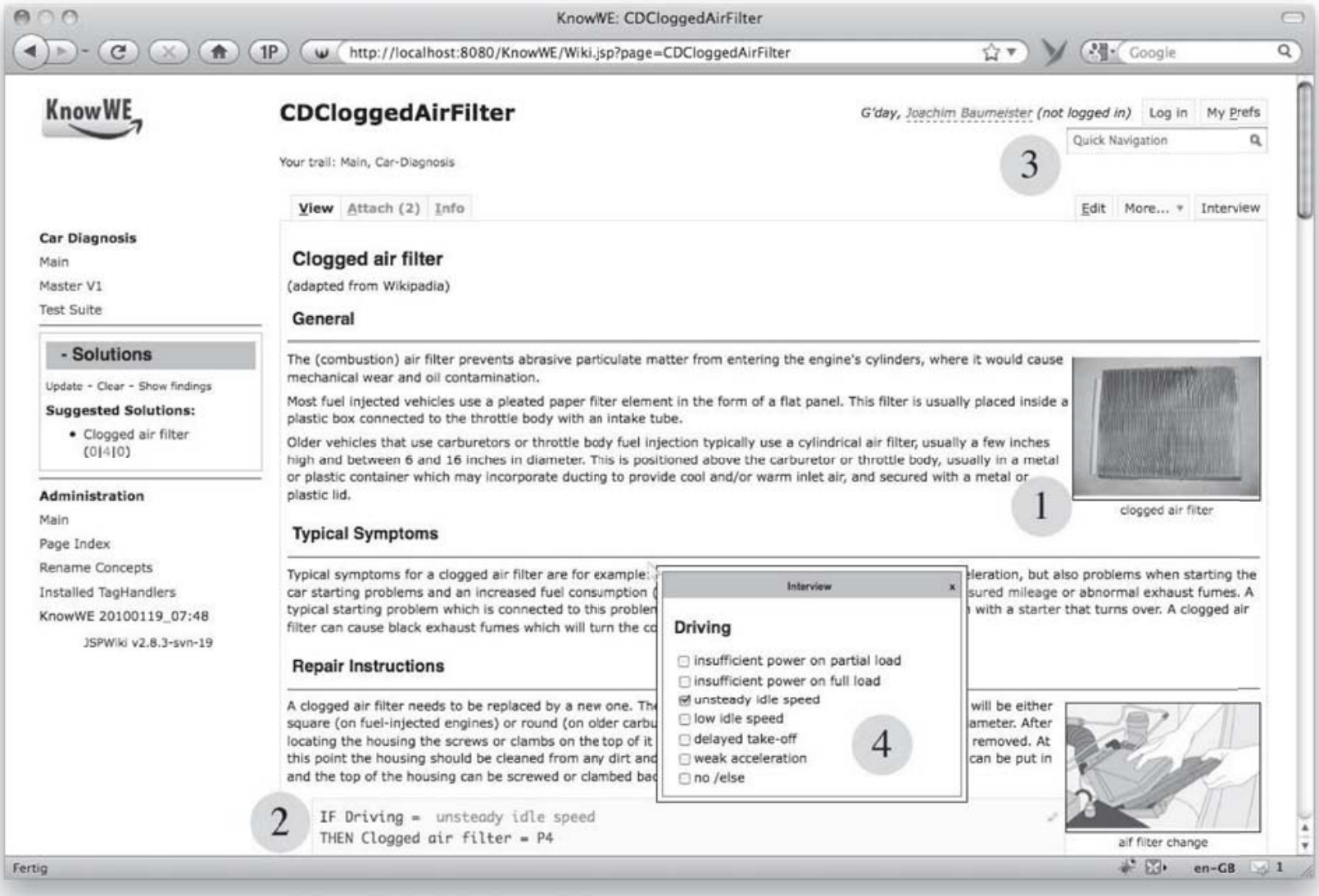
(We refer to Baumeister et al. (2011) Engineering Intelligent Systems on the Knowledge Formalization Continuum. Int. J. Appl. Math Comput. Sci. 21(1):27–39 for this, and following materials)

Example: Digitalys – Stomach Pain



Article of the cardinal symptom *stomach pain* ("Bauchschmerzen") showing some text explaining the general structure of the corresponding decision tree and an overview image, which can be enlarged on click (the original screenshot is depicted in the German language).

Example: Car Diagnosis KBS



Example: Car Diagnosis – Formal Knowledge

The Knowledge Formalization Continuum

- Continuum: "continuous sequence in which adjacent elements are not perceptibly different from each other, but the extremes are quite distinct" (Oxford English Dictionary)
- Interpretation: Mental concept of alternative knowledge representations, which gradual transitions
- Examples of extremes: *text* vs. *logic*
- Transition between representations, e. g.,
  - Natural language generation
  - Machine learning, knowledge discovery, text mining
- $\rightsquigarrow$ **(Semi-)automatic approaches for KA**:
  Machine Learning and *Knowledge Discovery* (e. g., rule-based representations), also *Interpretable Machine Learning*

Towards Automatic Knowledge Acquisition

Motivation

- In general, building knowledge-based systems can be costly, depending on availability of tools, domain specialists, data and knowledge sources ...
- Automatic knowledge acquisition: Learning/extracting knowledge from data
  - Machine Learning
  - Knowledge Discovery (in Databases), Data Mining
- Automatic methods can simplify constructing an initial knowledge base; for example, they can be applied for:
  - Identifying important concepts
  - Initial relations
  - Generating rules
  - ...
- Some challenges: Quality of the extracted knowledge, causality, coverage, ...

(Semi-)Automatic Knowledge Acquisition

- Actually, we already know (something) about (semi-)automatic knowledge acquisition
- $\rightsquigarrow$ Diagnostic Score Pattern
- $\rightsquigarrow$ Learning simple diagnostic scores
- Also: Refinement of existing scoring rules

Definition 1 (Scoring Rule)

A scoring rule r is denoted as follows

$$r = f_1 \diamond_1 \ldots \diamond_{n-1} f_n \xrightarrow{s} o\,,$$

where $f_i \in \Omega_{obs}$ are attribute values logically combined by conjunction or disjunction, i.e., $\diamond_i \in \{\wedge, \vee\}$, $s \in \mathbb{N}$ denotes the scoring point $sp(r)$ of the rule, and $o \in \Omega_{sol}$ is the targeted solution/output.

Example: Semi-Automatic Knowledge Acquisition

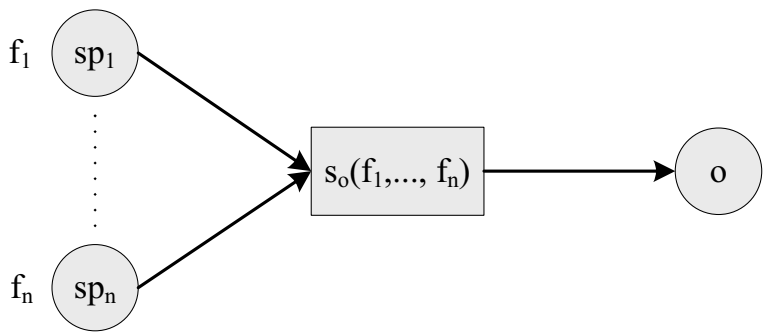
Referring to: (Atzmueller et al. (2007) Rapid Knowledge Capture Using Subgroup Discovery with Incremental Refinement. Proc. International Conference on Knowledge Capture (K-Cap))

Algorithm 1 Using Subgroup Discovery for Rapid Knowledge Capture

1: Consider global blacklist $B \subseteq \Omega_{sol}$
2: **for all** $c \in cases$ **do**
3: runCase(c), and update the frequency of incorrectly solved solutions
4: Retrieve the set SOL_k, i.e., the set of size k containing the most frequent solutions that were solved incorrectly
5: $SOL_k = SOL_k - B$
6: **if** $SOL_k \neq \emptyset$ **then**
7: **for all** $t \in SOL_k$ **do**
8: $S =$ mineSubgroups(t)
9: $S' = \{s \in S \,|\, q(s) > \alpha\}$
10: reduce S' to the m best subgroups
11: **for all** $s \in S'$ **do**
12: **if** $\nexists rule(sd(s) \rightarrow t\}$ **then**
13: compute the subgroup score $r_s = \phi(s) \cdot \mathcal{T}$
14: **if** $|r_s| > 0$ **then**
15: create a rule r with the assigned score r_s, i.e., $r = cond_{f_s} \stackrel{r_s}{\rightarrow} t$, where f_s is the (complex) finding corresponding to the subgroup description $sd(s)$ of s
16: **if** no rule for t was created **then**
17: $B = B \cup \{t\}$
18: Optionally: Apply a refinement strategy (see Alg. 2)

Example: Semi-Automatic Knowledge Refinement

- Since scoring rules are used to directly connect input values (findings) with possible outputs of the system, there exist strong similarities to perceptron networks.
- Thus, the possible input value f of the network is also directly connected with an output o of the network.
- The scoring point s of a rule $cond_f \xrightarrow{s} o$ is related with the perceptron weight of a particular input value.



Example: Semi-Automatic Knowledge Refinement

- Given a collection of input values $f_1, \ldots, f_n$ connected with a solution o by scoring rules $cond_{f_i} \xrightarrow{s_i} o$ the state of the solution is calculated according to the following equation:

$$s_o(f_1, \ldots, f_n) = \begin{cases} derived(o) & \text{if } s_1 + \cdots + s_n > \mathcal{T} \\ \neg\, derived(o) & \text{otherwise} \end{cases}, \quad (1)$$

  where $\mathcal{T} > 0$ is the globally defined threshold for deriving the particular solutions.

- The sum of score points s_i corresponds to the scoring account $\mathcal{A}_{o,c}$ for a given output o and input values f_i in a given case c.

Example: Semi-Automatic Knowledge Refinement

- In general, we can define the derivation state s_o of a given solution o with respect to a given case $c = (\, OBS_c, SOL_c \,)$

$$s_o(c) = \begin{cases} derived(o) & \text{if } \sum_{r \in \mathcal{R}_{o,c}} sp(r) > \mathcal{T} \\ \neg\, derived(o) & \text{otherwise} \end{cases}, \qquad (2)$$

  where $\mathcal{R}_{o,c} = \{\, r \in \mathcal{R} \,|\, r = cond_f \xrightarrow{s} o \wedge f \in OBS_c \,\}$ are all scoring rules in the rule base $\mathcal{R}$ with the solution o in the consequent, and for which the rule condition was satisfied concerning the case c; again, let $\mathcal{T} > 0$ be a globally defined threshold for the derivation of the particular solutions.

- The function $s_o(c)$ is equivalent to the perception function $o(\overrightarrow{s})$, where $\overrightarrow{s}$ is the vector of all input weights.

Incremental Scoring Rule Refinement I

- For the proposed refinement approach, we assume that the case base is already in a correct state, i.e., all cases c contain a correct solution SOL_c for the problem description OBS_c.
- For refinement, only scoring points of existing rules are adapted but no new rules are induced. Here, the developer of the rule base feels certain that all necessary connections between input and output values are modeled, and that only the weights of the particular rules need to be refined.
- We distinguish two refinement strategies:
  - The *general refinement strategy* selects the incorrectly derived output and uniformly adapts all respective scoring points/rules.
  - The *analytical refinement strategy* only adapts the relevant scoring points of rules that have been actually used (i.e. fired) in the incorrectly solved cases.

Incremental Scoring Rule Refinement II

- For both approaches there are reasonable motivations:
  - If the developer of the rule base feels certain that the distribution of the particular scoring points for a given output has been modeled very carefully, then the general refinement strategy should be applied. Here the scoring points of all relations are uniformly adapted, and we therefore preserve the overall proportions of the weights.
  - In contrast, if the developer of the rule base feels not confident with respect to the proportions of the scoring points, then the analytical refinement strategy should be more reasonable. Since only a selection of rules, i.e. the used rules for a case, are considered for refinement, only the corresponding scoring points are disproportionately increased/decreased with respect to the overall proportion.

Incremental Scoring Rule Refinement III

- The general algorithmic framework is given as follows:
  - The available training set of cases is successively given to the rule system in order to derive a solution for every given case c.
  - If the derived solution $SOL_{c'}$ differs from the (correct) solution SOL_c stored in the training case, then the false positive and false negative outputs are determined.
  - For each incorrect output the error deviation $\Delta(o, SOL_{c'})$ is propagated to the scoring points of the relevant rules.
- The algorithm terminates, if all cases were solved correctly or the number of iterations has exceeded a threshold.

Incremental Scoring Rule Refinement IV

Algorithm 2 Incremental Refinement Strategy

repeat
  for all $c \in cases$ **do**
    $c' \leftarrow$ runCase(c)
    if $SOL_c \neq SOL_{c'}$ **then**
      $F = fp(SOL_c, SOL_{c'}) \cup fn(SOL_c, SOL_{c'})$
      for all $o \in F$ **do**
        $\delta_s = \Delta(o, SOL_{c'}) \,/\, |\mathcal{R}_o|$
        for all $r \in \mathcal{R}_o$ **do**
          for $r = cond_f \xrightarrow{s} o$ **do**
            $s \leftarrow s + \delta_s$
until all cases have been solved correctly **or** the number of iterations exceeds a given threshold

The set $\mathcal{R}_o$ is defined according to the actual refinement strategy:

- For the general refinement strategy $\mathcal{R}_o = \{ r \in \mathcal{R} \mid r = f \xrightarrow{s} o \}$, i.e., the set of all scoring rules deriving the specified output o.
- For the analytical refinement strategy we define $\mathcal{R}_o = \mathcal{R}_{o,c'}$, i.e., all rules actually deriving the specified output o in the given case c'.

Incremental Scoring Rule Refinement V

- The functions $fn(SOL_c, SOL_{c'})$ and $fp(SOL_c, SOL_{c'})$ determine the false negative and false positive outputs with respect to the cases c and c'.
- The Δ-error of a given output $o \in \Omega_{sol}$ concerning a tested case c' is defined as the deviation

$$\Delta(o, SOL_{c'}) = \mathcal{T} - \mathcal{A}_{o,c'} \,, \tag{3}$$

where $\mathcal{T}$ is the fixed threshold for deriving a solution, and $\mathcal{A}_{o,c'}$ is the scoring account of the output o in the case c', i.e. the actual sum of the aggregated scoring points with respect to case c'.

Overview: Machine Learning

- Machine Learning: Learning from Data
- Knowledge Discovery & Data Mining
- ML & Computational Sensemaking
- Interpretable & Explainable ML
- Machine Learning on Graphs
- (Deep) Neural Networks
- Informed Learning

In the following subsection/slides (*Intro/Machine Learning Basics*), we will refer to and discuss some materials of the ML book
$\rightsquigarrow$ (Peter Flach, 2012), courtesy of Prof. Dr. Peter A. Flach.

To start: *Looking for Structure – Classifying spam e-mail*.

Assassinating spam e-mail

SpamAssassin is a widely used open-source spam filter. It calculates a score for an incoming e-mail, based on a number of built-in rules or 'tests' in SpamAssassin's terminology, and adds a 'junk' flag and a summary report to the e-mail's headers if the score is 5 or more.

```
-0.1 RCVD_IN_MXRATE_WL       RBL: MXRate recommends allowing
                             [123.45.6.789 listed in sub.mxrate.net]
 0.6 HTML_IMAGE_RATIO_02     BODY: HTML has a low ratio of text to image area
 1.2 TVD_FW_GRAPHIC_NAME_MID BODY: TVD_FW_GRAPHIC_NAME_MID
 0.0 HTML_MESSAGE            BODY: HTML included in message
 0.6 HTML_FONx_FACE_BAD      BODY: HTML font face is not a word
 1.4 SARE_GIF_ATTACH         FULL: Email has a inline gif
 0.1 BOUNCE_MESSAGE          MTA bounce message
 0.1 ANY_BOUNCE_MESSAGE      Message is some kind of bounce message
 1.4 AWL                     AWL: From: address is in the auto white-list
```

From left to right you see the score attached to a particular test, the test identifier, and a short description including a reference to the relevant part of the e-mail. As you see, scores for individual tests can be negative (indicating evidence suggesting the e-mail is ham rather than spam) as well as positive. The overall score of 5.3 suggests the e-mail might be spam.

Linear classification

Suppose we have only two tests and four training e-mails, one of which is spam (see Table 1). Both tests succeed for the spam e-mail; for one ham e-mail neither test succeeds, for another the first test succeeds and the second doesn't, and for the third ham e-mail the first test fails and the second succeeds.

It is easy to see that assigning both tests a weight of 4 correctly 'classifies' these four e-mails into spam and ham. In the mathematical notation introduced in Background 1 we could describe this classifier as $4x_1 + 4x_2 > 5$ or $(4,4) \cdot (x_1, x_2) > 5$.

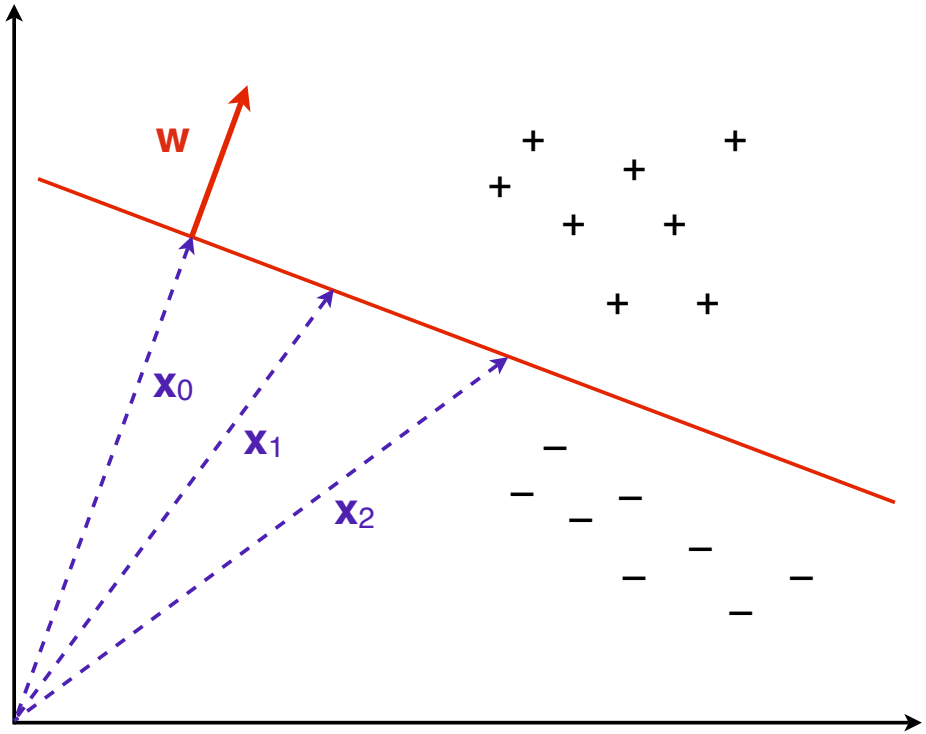
In fact, any weight between 2.5 and 5 will ensure that the threshold of 5 is only exceeded when both tests succeed. We could even consider assigning different weights to the tests – as long as each weight is less than 5 and their sum exceeds 5 – although it is hard to see how this could be justified by the training data.

Spam filtering as a classification task

E-mail	x_1	x_2	Spam?	$4x_1 + 4x_2$
1	1	1	1	8
2	0	0	0	0
3	1	0	0	4
4	0	1	0	4

The columns marked x_1 and x_2 indicate the results of two tests on four different e-mails. The fourth column indicates which of the e-mails are spam. The right-most column demonstrates that by thresholding the function $4x_1 + 4x_2$ at 5, we can separate spam from ham.

Linear classification in two dimensions



The straight line separates the positives from the negatives. It is defined by $\mathbf{w} \cdot \mathbf{x}_i = t$, where $\mathbf{w}$ is a vector perpendicular to the decision boundary and pointing in the direction of the positives, t is the decision threshold, and $\mathbf{x}_i$ points to a point on the decision boundary. In particular, $\mathbf{x}_0$ points in the same direction as $\mathbf{w}$, from which it follows that $\mathbf{w} \cdot \mathbf{x}_0 = ||\mathbf{w}|| \, ||\mathbf{x}_0|| = t$ ($||\mathbf{x}||$ denotes the length of the vector $\mathbf{x}$).

Homogeneous coordinates

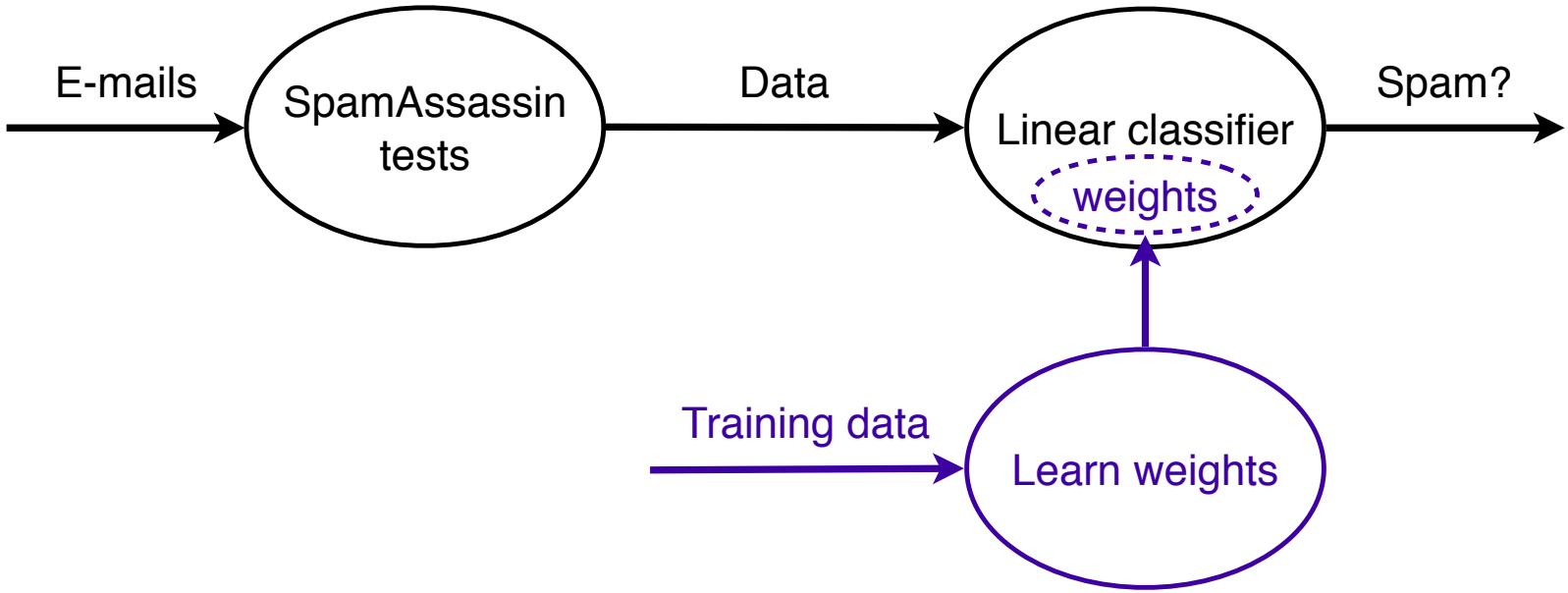
It is sometimes convenient to simplify notation further by introducing an extra constant 'variable' $x_0 = 1$, the weight of which is fixed to $w_0 = -t$.

The extended data point is then $\mathbf{x}^\circ = (1, x_1, \ldots, x_n)$ and the extended weight vector is $\mathbf{w}^\circ = (-t, w_1, \ldots, w_n)$, leading to the decision rule $\mathbf{w}^\circ \cdot \mathbf{x}^\circ > 0$ and the decision boundary $\mathbf{w}^\circ \cdot \mathbf{x}^\circ = 0$.

Thanks to these so-called homogeneous coordinates the decision boundary passes through the origin of the extended coordinate system, at the expense of needing an additional dimension.

☞ note that this doesn't really affect the data, as all data points and the 'real' decision boundary live in the plane $x_0 = 1$.

Machine learning for spam filtering



At the top we see how SpamAssassin approaches the spam e-mail classification task: the text of each e-mail is converted into a data point by means of SpamAssassin's built-in tests, and a linear classifier is applied to obtain a 'spam or ham' decision. At the bottom (in blue) we see the bit that is done by machine learning.

A Bayesian classifier I

Bayesian spam filters maintain a vocabulary of words and phrases – potential spam or ham indicators – for which statistics are collected from a training set.

- For instance, suppose that the word 'Viagra' occurred in four spam e-mails and in one ham e-mail. If we then encounter a new e-mail that contains the word 'Viagra', we might reason that the odds that this e-mail is spam are 4:1, or the probability of it being spam is 0.80 and the probability of it being ham is 0.20.
- The situation is slightly more subtle because we have to take into account the prevalence of spam. Suppose that I receive on average one spam e-mail for every six ham e-mails. This means that I would estimate the odds of an unseen e-mail being spam as 1:6, i.e., non-negligible but not very high either.

A Bayesian classifier II

- If I then learn that the e-mail contains the word ‘Viagra’, which occurs four times as often in spam as in ham, I need to combine these two odds. As we shall see later, Bayes’ rule tells us that we should simply multiply them: 1:6 times 4:1 is 4:6, corresponding to a spam probability of 0.4.

In this way you are combining two independent pieces of evidence, one concerning the prevalence of spam, and the other concerning the occurrence of the word ‘Viagra’, pulling in opposite directions.

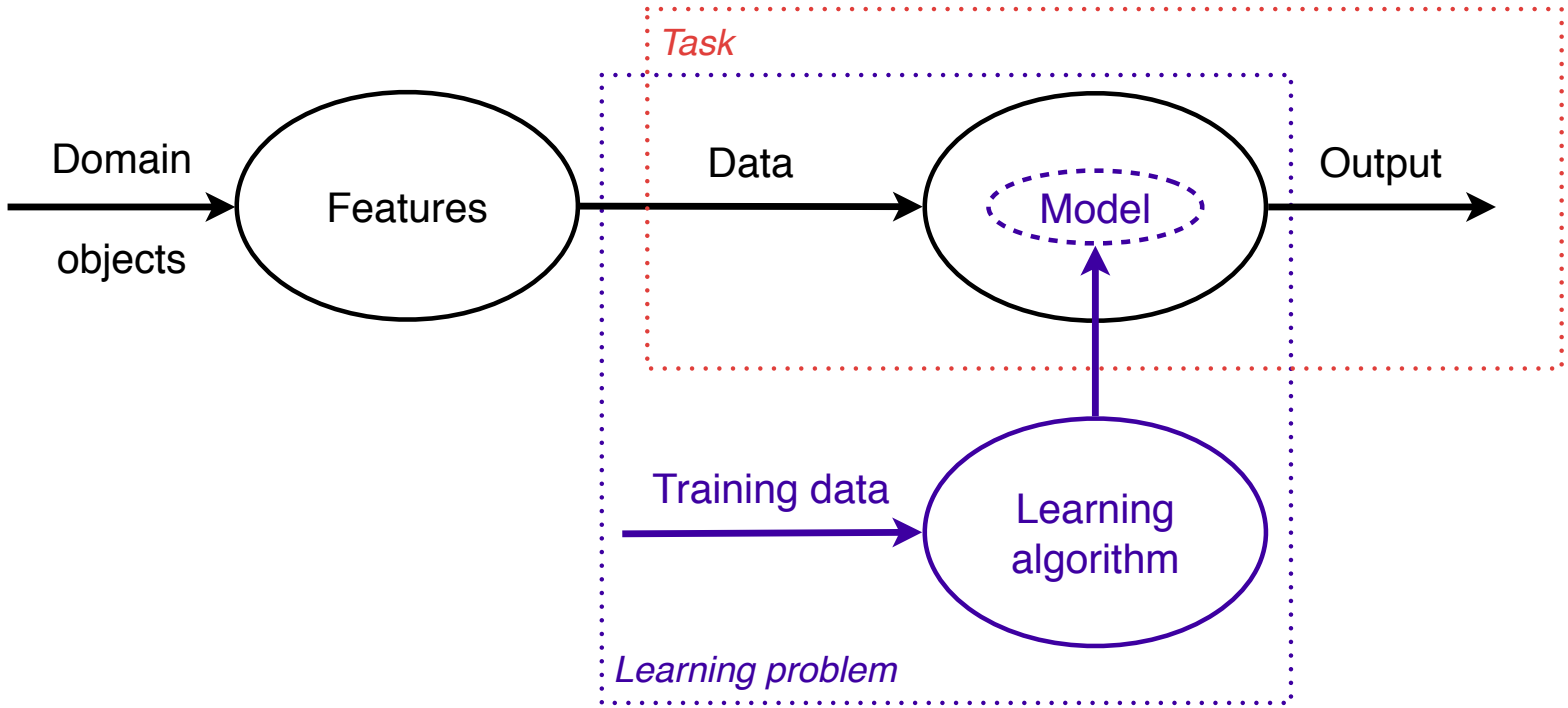
The nice thing about this ‘Bayesian’ classification scheme is that it can be repeated if you have further evidence. For instance, suppose that the odds in favour of spam associated with the phrase ‘blue pill’ is estimated at 3:1, and suppose our e-mail contains both ‘Viagra’ and ‘blue pill’, then the combined odds are 4:1 times 3:1 is 12:1, which is ample to outweigh the 1:6 odds associated with the low prevalence of spam (total odds are 2:1, or a spam probability of 0.67, up from 0.40 without the ‘blue pill’).

A rule-based classifier

- if the e-mail contains the word ‘Viagra’ then estimate the odds of spam as 4:1;
- otherwise, if it contains the phrase ‘blue pill’ then estimate the odds of spam as 3:1;
- otherwise, estimate the odds of spam as 1:6.

The first rule covers all e-mails containing the word ‘Viagra’, regardless of whether they contain the phrase ‘blue pill’, so no overcounting occurs. The second rule *only* covers e-mails containing the phrase ‘blue pill’ but not the word ‘Viagra’, by virtue of the ‘otherwise’ clause. The third rule covers all remaining e-mails: those which neither contain neither ‘Viagra’ nor ‘blue pill’.

How machine learning helps to solve a task



An overview of how machine learning is used to address a given task. A task (red box) requires an appropriate mapping – a model – from data described by features to outputs. Obtaining such a mapping from training data is what constitutes a learning problem (blue box).

Tasks for machine learning

The most common machine learning tasks are *predictive*, in the sense that they concern predicting a target variable from features. .

- Binary and multi-class classification: categorical target
- Regression: numerical target
- Clustering: hidden target

Descriptive tasks are concerned with exploiting underlying structure in the data.

Measuring similarity

If our e-mails are described by word-occurrence features as in the text classification example, the similarity of e-mails would be measured in terms of the words they have in common. For instance, we could take the number of common words in two e-mails and divide it by the number of words occurring in either e-mail (this measure is called the *Jaccard coefficient*).

Suppose that one e-mail contains 42 (different) words and another contains 112 words, and the two e-mails have 23 words in common, then their similarity would be $\frac{23}{42+112-23} = \frac{23}{130} = 0.18$. We can then cluster our e-mails into groups, such that the average similarity of an e-mail to the other e-mails in its group is much larger than the average similarity to e-mails from other groups.

Machine learning settings

	Predictive model	*Descriptive model*
Supervised learning	classification, regression	subgroup discovery
Unsupervised learning	predictive clustering	descriptive clustering, association rule discovery

The rows refer to whether the training data is labelled with a target variable, while the columns indicate whether the models learned are used to predict a target variable or rather describe the given data.

Machine learning models

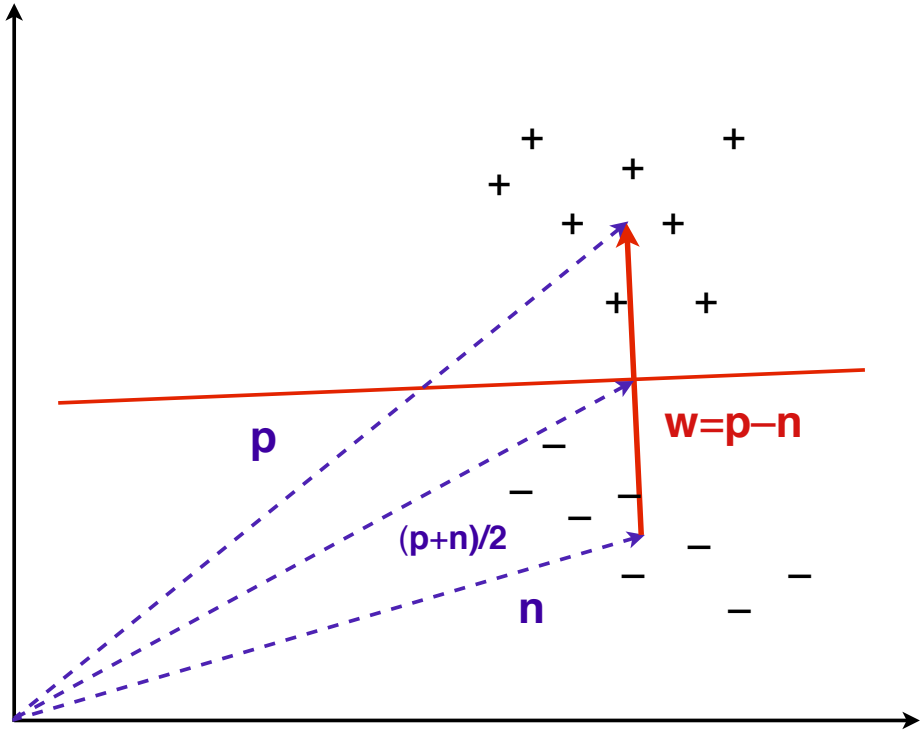
Machine learning models can be distinguished according to their main intuition:

- *Geometric* models use intuitions from geometry such as separating (hyper-)planes, linear transformations and distance metrics.
- *Probabilistic* models view learning as a process of reducing uncertainty, modelled by means of probability distributions.
- *Logical* models are defined in terms of easily interpretable logical expressions.

Alternatively, they can be characterised by their *modus operandi*:

- *Grouping models* divide the instance space into segments; in each segment a very simple (e.g., constant) model is learned.
- *Grading models* learning a single, global model over the instance space.

Basic linear classifier



The basic linear classifier constructs a decision boundary by half-way intersecting the line between the positive and negative centres of mass. It is described by the equation $\mathbf{w} \cdot \mathbf{x} = t$, with $\mathbf{w} = \mathbf{p} - \mathbf{n}$; the decision threshold can be found by noting that $(\mathbf{p} + \mathbf{n})/2$ is on the decision boundary, and hence $t = (\mathbf{p} - \mathbf{n}) \cdot (\mathbf{p} + \mathbf{n})/2 = (||\mathbf{p}||^2 - ||\mathbf{n}||^2)/2$, where $||\mathbf{x}||$ denotes the length of vector $\mathbf{x}$.

A simple probabilistic model

Viagra	lottery	$P(Y = \text{spam}\|\text{Viagra}, \text{lottery})$	$P(Y = \text{ham}\|\text{Viagra}, \text{lottery})$
0	0	0.31	**0.69**
0	1	**0.65**	0.35
1	0	**0.80**	0.20
1	1	0.40	**0.60**

'Viagra' and 'lottery' are two Boolean features; Y is the class variable, with values 'spam' and 'ham'. In each row the most likely class is indicated in bold.

Decision rule

Assuming that X and Y are the only variables we know and care about, the posterior distribution $P(Y|X)$ helps us to answer many questions of interest.

- For instance, to classify a new e-mail we determine whether the words 'Viagra' and 'lottery' occur in it, look up the corresponding probability $P(Y = \mathsf{spam}|\mathsf{Viagra}, \mathsf{lottery})$, and predict spam if this probability exceeds 0.5 and ham otherwise.
- Such a recipe to predict a value of Y on the basis of the values of X and the posterior distribution $P(Y|X)$ is called a *decision rule*.

Likelihood ratio

As a matter of fact, statisticians work very often with different conditional probabilities, given by the *likelihood function* $P(X|Y)$.

- I like to think of these as thought experiments: if somebody were to send me a spam e-mail, how likely would it be that it contains exactly the words of the e-mail I'm looking at? And how likely if it were a ham e-mail instead?
- What really matters is not the magnitude of these likelihoods, but their ratio: how much more likely is it to observe this combination of words in a spam e-mail than it is in a non-spam e-mail.
- For instance, suppose that for a particular e-mail described by X we have $P(X|Y = \text{spam}) = 3.5 \cdot 10^{-5}$ and $P(X|Y = \text{ham}) = 7.4 \cdot 10^{-6}$, then observing X in a spam e-mail is nearly five times more likely than it is in a ham e-mail.
- This suggests the following decision rule: predict spam if the likelihood ratio is larger than 1 and ham otherwise.

Posterior odds

$$\frac{P(Y = \text{spam}|\text{Viagra} = 0, \text{lottery} = 0)}{P(Y = \text{ham}|\text{Viagra} = 0, \text{lottery} = 0)} = \frac{0.31}{0.69} = 0.45$$
$$\frac{P(Y = \text{spam}|\text{Viagra} = 1, \text{lottery} = 1)}{P(Y = \text{ham}|\text{Viagra} = 1, \text{lottery} = 1)} = \frac{0.40}{0.60} = 0.67$$
$$\frac{P(Y = \text{spam}|\text{Viagra} = 0, \text{lottery} = 1)}{P(Y = \text{ham}|\text{Viagra} = 0, \text{lottery} = 1)} = \frac{0.65}{0.35} = 1.9$$
$$\frac{P(Y = \text{spam}|\text{Viagra} = 1, \text{lottery} = 0)}{P(Y = \text{ham}|\text{Viagra} = 1, \text{lottery} = 0)} = \frac{0.80}{0.20} = 4.0$$

Using a MAP decision rule we predict ham in the top two cases and spam in the bottom two. Given that the full posterior distribution is all there is to know about the domain in a statistical sense, these predictions are the best we can do: they are *Bayes-optimal*.

Example marginal likelihoods

Y	$P(\text{Viagra} = 1 \mid Y)$	$P(\text{Viagra} = 0 \mid Y)$
spam	0.40	0.60
ham	0.12	0.88

Y	$P(\text{lottery} = 1 \mid Y)$	$P(\text{lottery} = 0 \mid Y)$
spam	0.21	0.79
ham	0.13	0.87

Using marginal likelihoods

Using the marginal likelihoods from Table 1.3, we can approximate the likelihood ratios (the previously calculated odds from the full posterior distribution are shown in brackets):

$$\frac{P(\text{Viagra}=0|Y=\text{spam})}{P(\text{Viagra}=0|Y=\text{ham})}\frac{P(\text{lottery}=0|Y=\text{spam})}{P(\text{lottery}=0|Y=\text{ham})}=\frac{0.60}{0.88}\frac{0.79}{0.87}=0.62 \quad (0.45)$$

$$\frac{P(\text{Viagra}=0|Y=\text{spam})}{P(\text{Viagra}=0|Y=\text{ham})}\frac{P(\text{lottery}=1|Y=\text{spam})}{P(\text{lottery}=1|Y=\text{ham})}=\frac{0.60}{0.88}\frac{0.21}{0.13}=1.1 \quad (1.9)$$

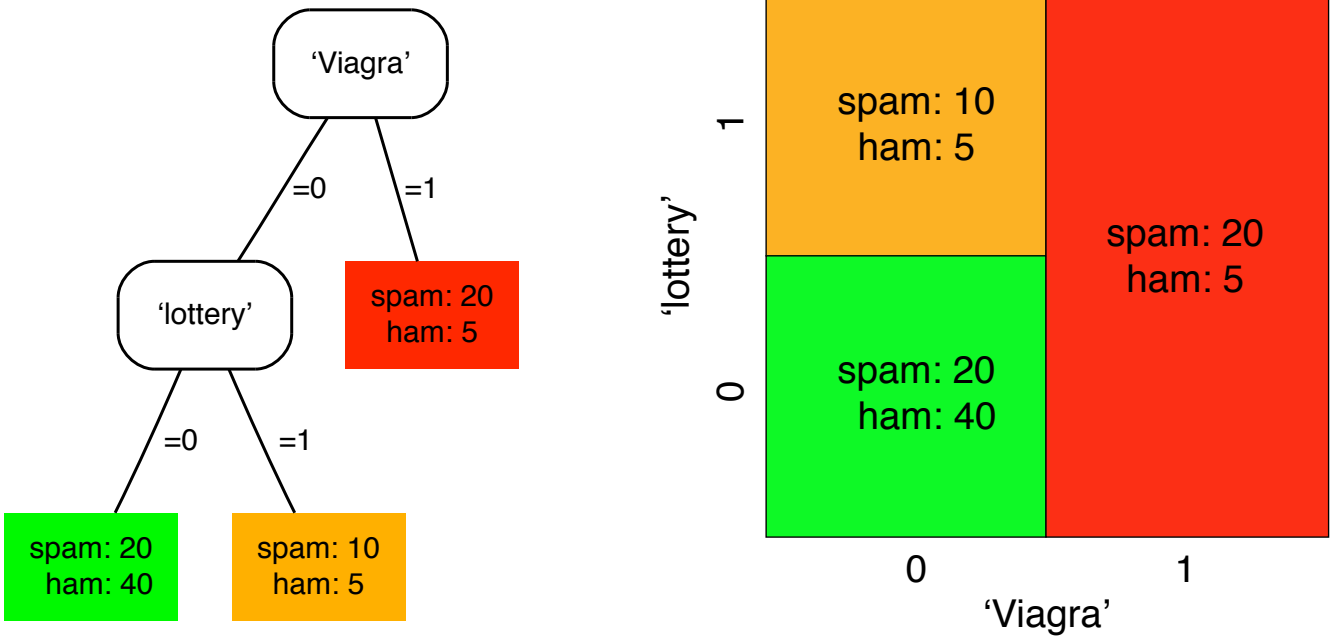
$$\frac{P(\text{Viagra}=1|Y=\text{spam})}{P(\text{Viagra}=1|Y=\text{ham})}\frac{P(\text{lottery}=0|Y=\text{spam})}{P(\text{lottery}=0|Y=\text{ham})}=\frac{0.40}{0.12}\frac{0.79}{0.87}=3.0 \quad (4.0)$$

$$\frac{P(\text{Viagra}=1|Y=\text{spam})}{P(\text{Viagra}=1|Y=\text{ham})}\frac{P(\text{lottery}=1|Y=\text{spam})}{P(\text{lottery}=1|Y=\text{ham})}=\frac{0.40}{0.12}\frac{0.21}{0.13}=5.4 \quad (0.67)$$

We see that, using a maximum likelihood decision rule, our very simple model arrives at the Bayes-optimal prediction in the first three cases, but not in the fourth ('Viagra' and 'lottery' both present), where the marginal likelihoods are actually very misleading.

Figure 1.4, p.32

A feature tree

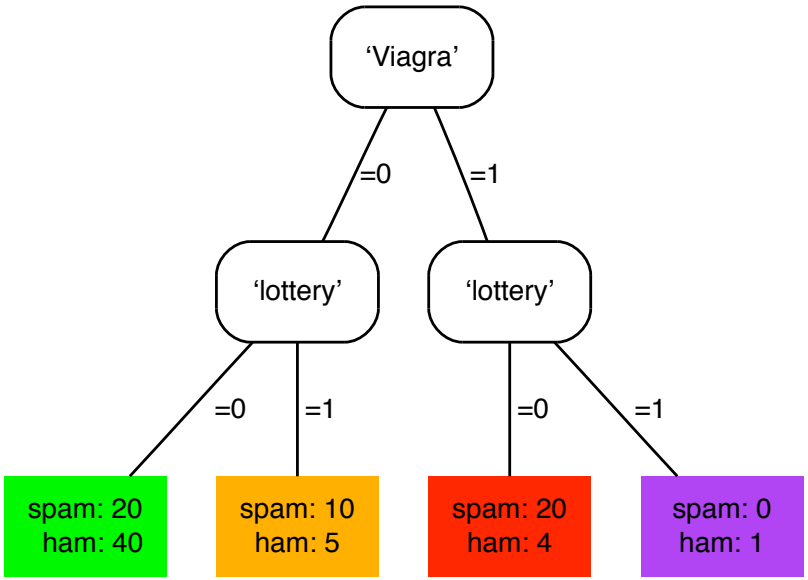


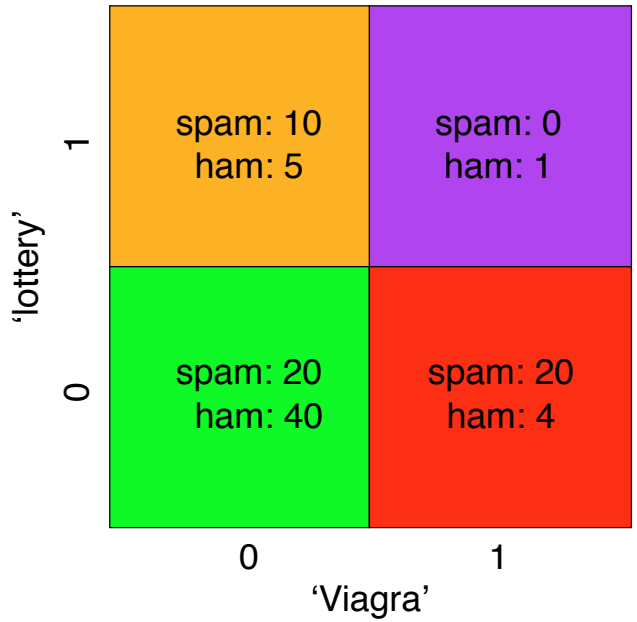
(left) A feature tree combining two Boolean features. Each internal node or split is labelled with a feature, and each edge emanating from a split is labelled with a feature value. Each leaf therefore corresponds to a unique combination of feature values. Also indicated in each leaf is the class distribution derived from the training set. **(right)** A feature tree partitions the instance space into rectangular regions, one for each leaf. We can clearly see that the majority of ham lives in the lower left-hand corner.

Labelling a feature tree

- The leaves of the tree in Figure 1.4 could be labelled, from left to right, as ham – spam – spam, employing a simple decision rule called *majority class*.
- Alternatively, we could label them with the proportion of spam e-mail occurring in each leaf: from left to right, 1/3, 2/3, and 4/5.
- Or, if our task was a regression task, we could label the leaves with predicted real values or even linear functions of some other, real-valued features.

A complete feature tree





(left) A complete feature tree built from two Boolean features. **(right)** The corresponding instance space partition is the finest partition that can be achieved with those two features.

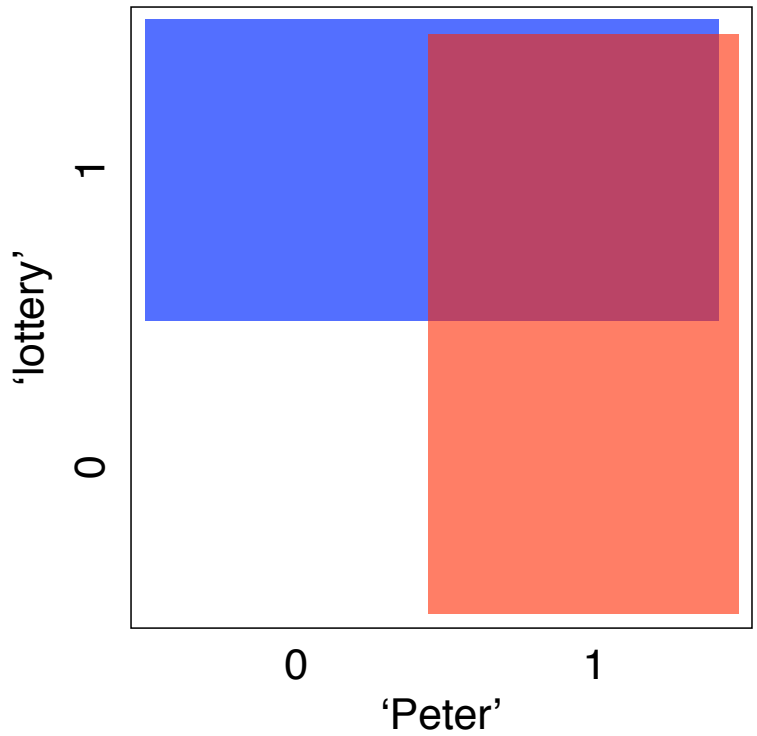
Overlapping rules

Consider the following rules:

$$\cdot\textbf{if } \text{lottery} = 1 \textbf{ then } \text{Class} = \text{Y} = \text{spam}\cdot$$
$$\cdot\textbf{if } \text{Peter} = 1 \textbf{ then } \text{Class} = \text{Y} = \text{ham}\cdot$$

As can be seen in Figure 1.6, these rules overlap for $\text{lottery} = 1 \wedge \text{Peter} = 1$, for which they make contradictory predictions. Furthermore, they fail to make any predictions for $\text{lottery} = 0 \wedge \text{Peter} = 0$.

Overlapping rules



The effect of overlapping rules in instance space. The two rules make contradictory predictions in the top right-hand corner, and no prediction at all in the bottom left-hand corner.

The MLM data set

Suppose we have a number of learning models that we want to describe in terms of a number of properties:

- the extent to which the models are geometric, probabilistic or logical;
- whether they are grouping or grading models;
- the extent to which they can handle discrete and/or real-valued features;
- whether they are used in supervised or unsupervised learning; and
- the extent to which they can handle multi-class problems.

The first two properties could be expressed by discrete features with three and two values, respectively; or if the distinctions are more gradual, each aspect could be rated on some numerical scale. A simple approach would be to measure each property on an integer scale from 0 to 3, as in Table 1.4. This table establishes a data set in which each row represents an instance and each column a feature.

The MLM data set

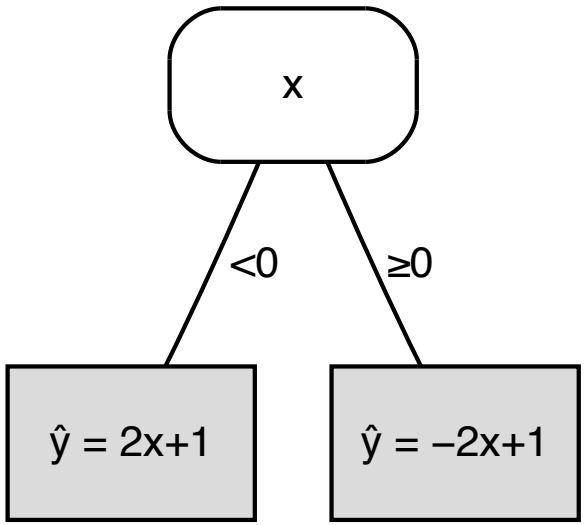
Model	geom	stats	logic	group	grad	disc	real	sup	unsup	multi
Trees	1	0	3	3	0	3	2	3	2	3
Rules	0	0	3	3	1	3	2	3	0	2
naive Bayes	1	3	1	3	1	3	1	3	0	3
kNN	3	1	0	2	2	1	3	3	0	3
Linear Classifier	3	0	0	0	3	1	3	3	0	0
Linear Regression	3	1	0	0	3	0	3	3	0	1
Logistic Regression	3	2	0	0	3	1	3	3	0	0
SVM	2	2	0	0	3	2	3	3	0	0
Kmeans	3	2	0	1	2	1	3	0	3	1
GMM	1	3	0	0	3	1	3	0	3	1
Associations	0	0	3	3	0	3	1	0	3	1

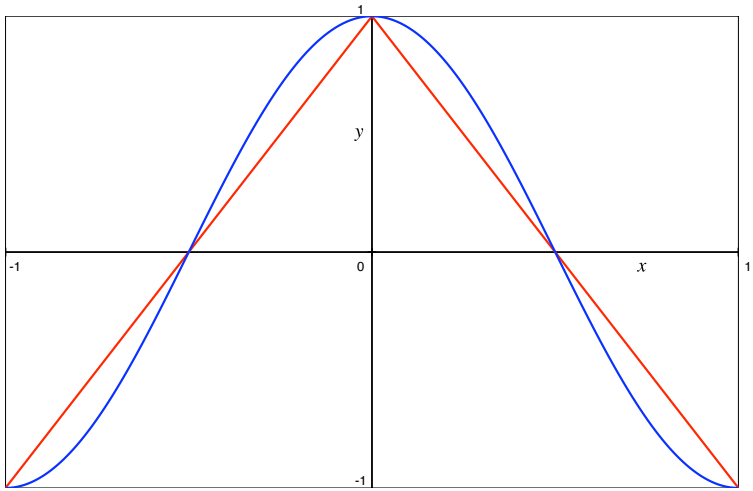
The MLM data set describing properties of machine learning models.

Two uses of features

Suppose we want to approximate $y = \cos \pi x$ on the interval $-1 \leq x \leq 1$. A linear approximation is not much use here, since the best fit would be $y = 0$. However, if we split the x-axis in two intervals $-1 \leq x < 0$ and $0 \leq x \leq 1$, we could find reasonable linear approximations on each interval. We can achieve this by using x both as a splitting feature and as a regression variable (Figure 1.9).

A small regression tree





(left) A regression tree combining a one-split feature tree with linear regression models in the leaves. Notice how x is used as both a splitting feature and a regression variable. **(right)** The function $y = \cos \pi x$ on the interval $-1 \leq x \leq 1$, and the piecewise linear approximation achieved by the regression tree.

The kernel trick

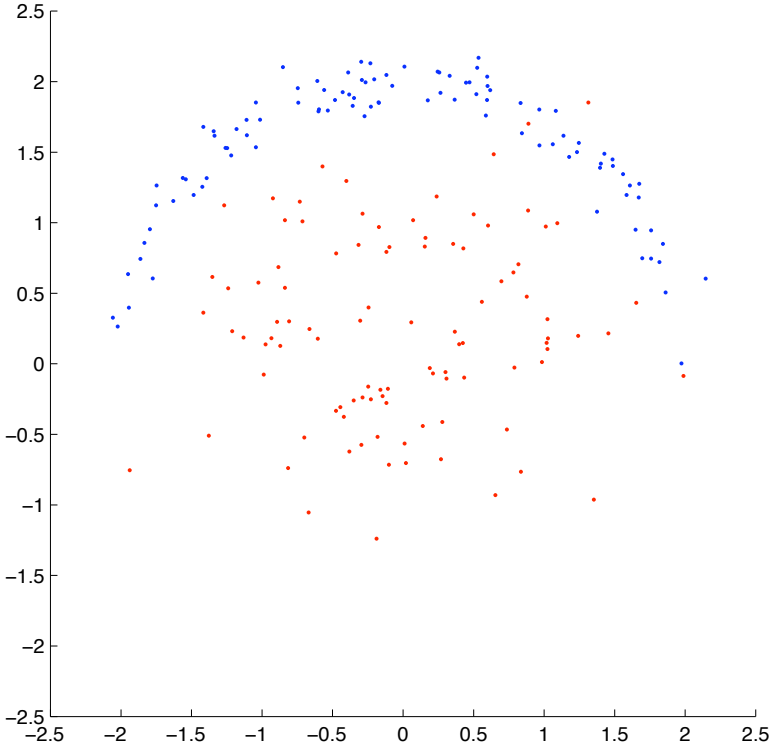
Let $\mathbf{x}_1 = (x_1, y_1)$ and $\mathbf{x}_2 = (x_2, y_2)$ be two data points, and consider the mapping $(x, y) \mapsto (x^2, y^2, \sqrt{2}xy)$ to a three-dimensional feature space. The points in feature space corresponding to $\mathbf{x}_1$ and $\mathbf{x}_2$ are $\mathbf{x}'_1 = (x_1^2, y_1^2, \sqrt{2}x_1 y_1)$ and $\mathbf{x}'_2 = (x_2^2, y_2^2, \sqrt{2}x_2 y_2)$. The dot product of these two feature vectors is

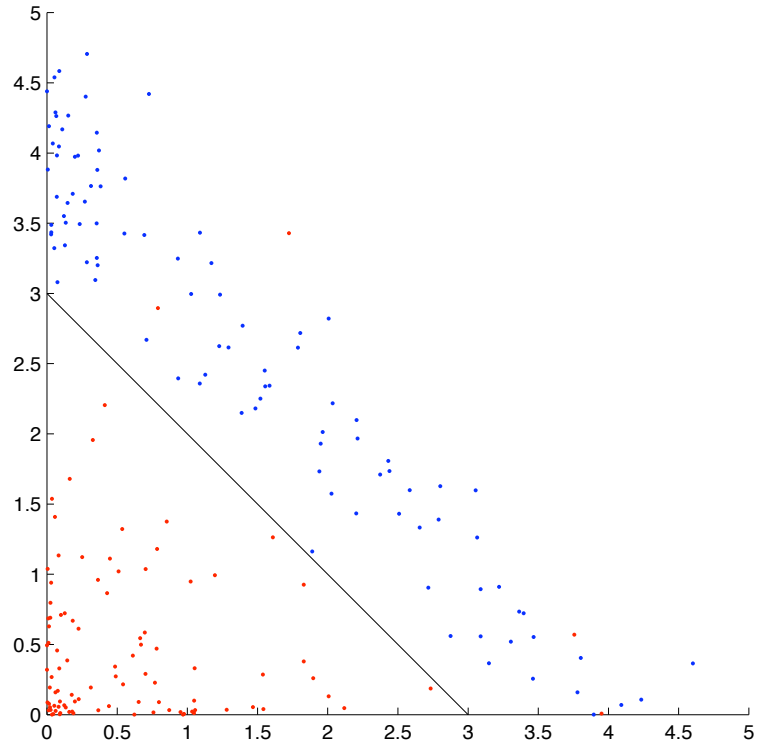
$$\mathbf{x}'_1 \cdot \mathbf{x}'_2 = x_1^2 x_2^2 + y_1^2 y_2^2 + 2x_1 y_1 x_2 y_2 = (x_1 x_2 + y_1 y_2)^2 = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$$

That is, by squaring the dot product in the original space we obtain the dot product in the new space *without actually constructing the feature vectors*! A function that calculates the dot product in feature space directly from the vectors in the original space is called a *kernel* – here the kernel is $\kappa(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$.

Figure 1.11, p.43

Non-linearly separable data





(left) A linear classifier would perform poorly on this data. **(right)** By transforming the original (x, y) data into $(x', y') = (x^2, y^2)$, the data becomes more 'linear', and a linear decision boundary $x' + y' = 3$ separates the data fairly well. In the original space this corresponds to a circle with radius $\sqrt{3}$ around the origin.

Predictive machine learning scenarios

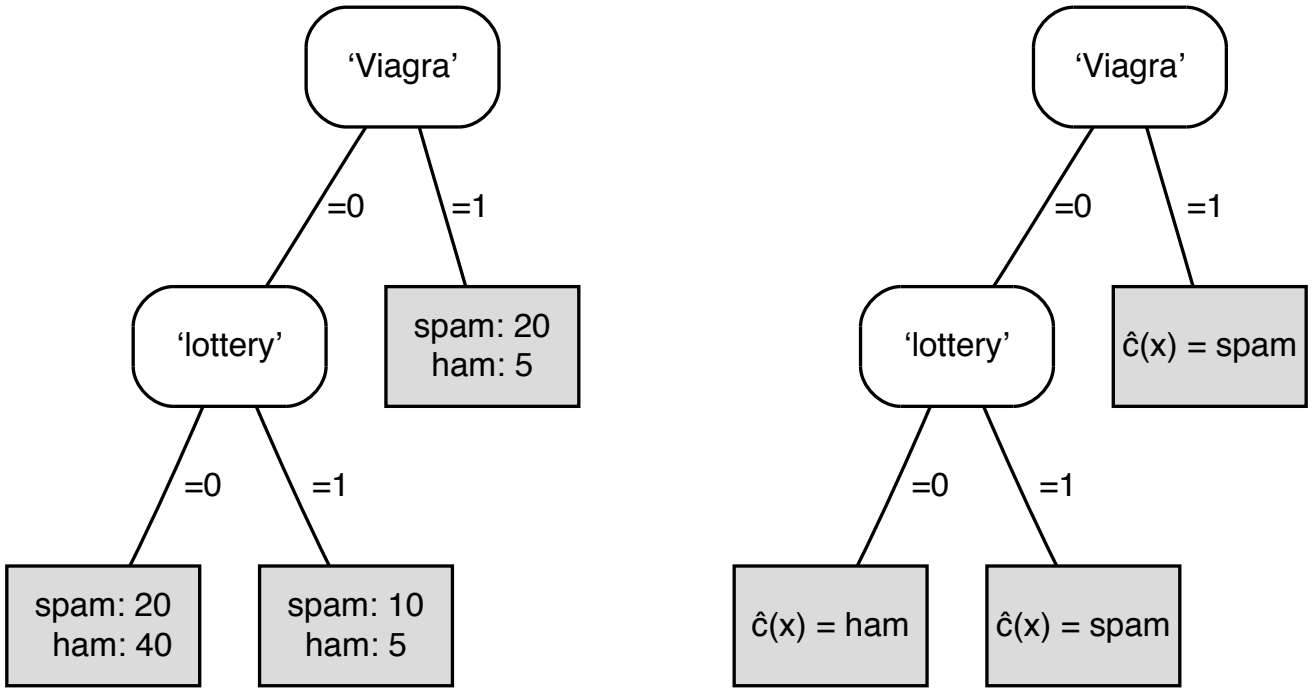
Task	*Label space*	*Output space*	*Learning problem*
Classification	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = \mathscr{C}$	learn an approximation $\hat{c} : \mathscr{X} \to \mathscr{C}$ to the true labelling function c
Scoring and ranking	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = \mathbb{R}^{\lvert\mathscr{C}\rvert}$	learn a model that outputs a score vector over classes
Probability estimation	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = [0,1]^{\lvert\mathscr{C}\rvert}$	learn a model that outputs a probability vector over classes
Regression	$\mathscr{L} = \mathbb{R}$	$\mathscr{Y} = \mathbb{R}$	learn an approximation $\hat{f} : \mathscr{X} \to \mathbb{R}$ to the true labelling function f

A *classifier* is a mapping $\hat{c} : \mathscr{X} \to \mathscr{C}$, where $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ is a finite and usually small set of *class labels*. We will sometimes also use C_i to indicate the set of examples of that class.

We use the 'hat' to indicate that $\hat{c}(x)$ is an estimate of the true but unknown function $c(x)$. Examples for a classifier take the form $(x, c(x))$, where $x \in \mathscr{X}$ is an instance and $c(x)$ is the true class of the instance (sometimes contaminated by noise).

Learning a classifier involves constructing the function $\hat{c}$ such that it matches c as closely as possible (and not just on the training set, but ideally on the entire instance space $\mathscr{X}$).

A decision tree



(left) A feature tree with training set class distribution in the leaves. **(right)** A decision tree obtained using the majority class decision rule.

Contingency table

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**30**	**20**	50
Actual ⊖	**10**	**40**	50
	40	60	100

	⊕	⊖	
⊕	**20**	**30**	50
⊖	**20**	**30**	50
	40	60	100

(left) A two-class contingency table or confusion matrix depicting the performance of the decision tree in Figure 2.1. Numbers on the descending diagonal indicate correct predictions, while the ascending diagonal concerns prediction errors. **(right)** A contingency table with the same marginals but independent rows and columns.

Example 2.1, p.56

Accuracy as a weighted average

Suppose a classifier's predictions on a test set are as in the following table:

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**60**	**15**	75
Actual ⊖	**10**	**15**	25
	70	30	100

From this table, we see that the true positive rate is $tpr = 60/75 = 0.80$ and the true negative rate is $tnr = 15/25 = 0.60$. The overall accuracy is $acc = (60 + 15)/100 = 0.75$, which is no longer the average of true positive and negative rates. However, taking into account the proportion of positives $pos = 0.75$ and the proportion of negatives $neg = 1 - pos = 0.25$, we see that

$$acc = pos \cdot tpr + neg \cdot tnr$$

Performance measures I

Measure	*Definition*	*Equal to*	*Estimates*
number of positives	$Pos = \sum_{x \in Te} I[c(x) = \oplus]$		
number of negatives	$Neg = \sum_{x \in Te} I[c(x) = \ominus]$	$\lvert Te \rvert - Pos$	
number of true positives	$TP = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]$		
number of true negatives	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]$		
number of false positives	$FP = \sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]$	$Neg - TN$	
number of false negatives	$FN = \sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]$	$Pos - TP$	
proportion of positives	$pos = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \oplus]$	$Pos/\lvert Te \rvert$	$P(c(x) = \oplus)$
proportion of negatives	$neg = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \ominus]$	$1 - pos$	$P(c(x) = \ominus)$
class ratio	$clr = pos/neg$	Pos/Neg	
(*) accuracy	$acc = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$
(*) error rate	$err = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$

Performance measures II

Measure	*Definition*	*Equal to*	*Estimates*
true positive rate, sensitivity, recall	$tpr = \frac{\sum_{x \in Te} I[\hat{c}(x)=c(x)=\oplus]}{\sum_{x \in Te} I[c(x)=\oplus]}$	TP/Pos	$P(\hat{c}(x)=\oplus \mid c(x)=\oplus)$
true negative rate, specificity	$tnr = \frac{\sum_{x \in Te} I[\hat{c}(x)=c(x)=\ominus]}{\sum_{x \in Te} I[c(x)=\ominus]}$	TN/Neg	$P(\hat{c}(x)=\ominus \mid c(x)=\ominus)$
false positive rate, false alarm rate	$fpr = \frac{\sum_{x \in Te} I[\hat{c}(x)=\oplus, c(x)=\ominus]}{\sum_{x \in Te} I[c(x)=\ominus]}$	$FP/Neg = 1 - tnr$	$P(\hat{c}(x)=\oplus \mid c(x)=\ominus)$
false negative rate	$fnr = \frac{\sum_{x \in Te} I[\hat{c}(x)=\ominus, c(x)=\oplus]}{\sum_{x \in Te} I[c(x)=\oplus]}$	$FN/Pos = 1 - tpr$	$P(\hat{c}(x)=\ominus \mid c(x)=\oplus)$
precision, confidence	$prec = \frac{\sum_{x \in Te} I[\hat{c}(x)=c(x)=\oplus]}{\sum_{x \in Te} I[\hat{c}(x)=\oplus]}$	$TP/(TP+FP)$	$P(c(x)=\oplus \mid \hat{c}(x)=\oplus)$

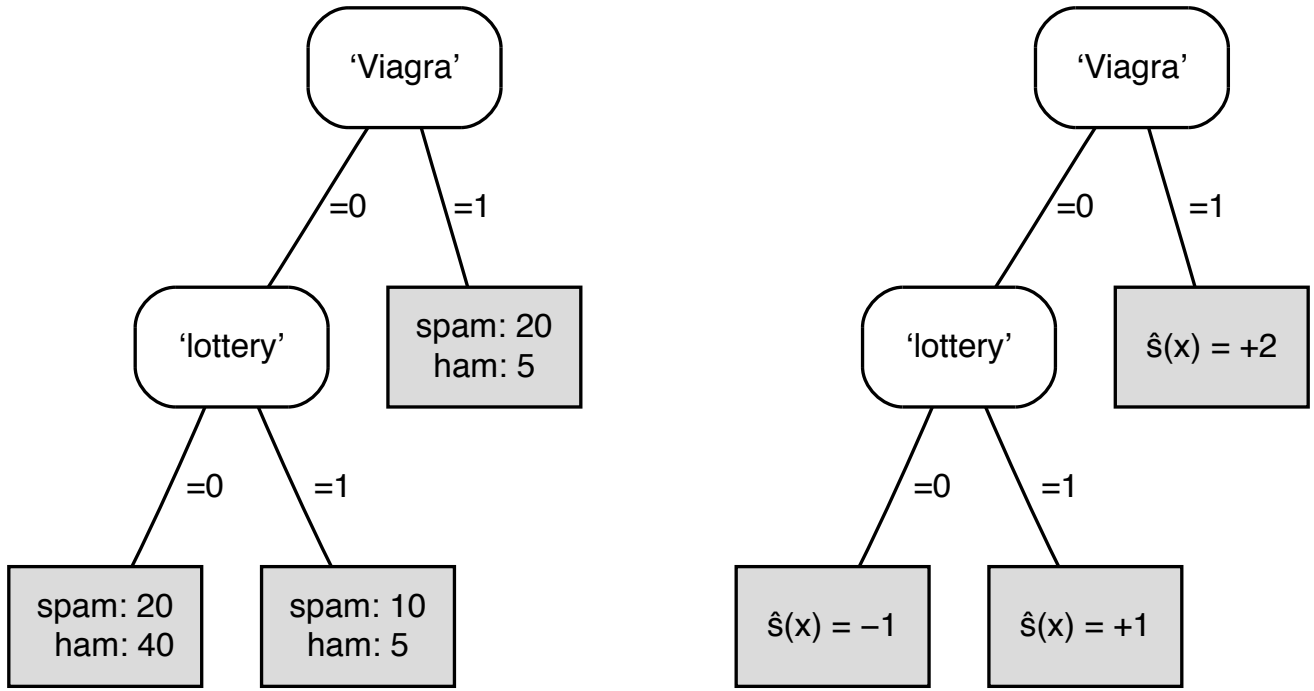
Table: A summary of different quantities and evaluation measures for classifiers on a test set Te. Symbols starting with a capital letter denote absolute frequencies (counts), while lower-case symbols denote relative frequencies or ratios. All except those indicated with (*) are defined only for binary classification.

Scoring classifier

A *scoring classifier* is a mapping $\hat{\mathbf{s}} : \mathscr{X} \to \mathbb{R}^k$, i.e., a mapping from the instance space to a k-vector of real numbers.
The boldface notation indicates that a scoring classifier outputs a vector $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$ rather than a single number; $\hat{s}_i(x)$ is the score assigned to class C_i for instance x.
This score indicates how likely it is that class label C_i applies.

If we only have two classes, it usually suffices to consider the score for only one of the classes; in that case, we use $\hat{s}(x)$ to denote the score of the positive class for instance x.

A scoring tree



(left) A feature tree with training set class distribution in the leaves. **(right)** A scoring tree using the logarithm of the class ratio as scores; spam is taken as the positive class.

Real-valued targets

A *function estimator*, also called a *regressor*, is a mapping $\hat{f} : \mathscr{X} \to \mathbb{R}$. The regression learning problem is to learn a function estimator from examples $(x_i, f(x_i))$.

Note that we switched from a relatively low-resolution target variable to one with infinite resolution. Trying to match this precision in the function estimator will almost certainly lead to overfitting – besides, it is highly likely that some part of the target values in the examples is due to fluctuations that the model is unable to capture.

It is therefore entirely reasonable to assume that the examples are noisy, and that the estimator is only intended to capture the general trend or shape of the function.

Line fitting example

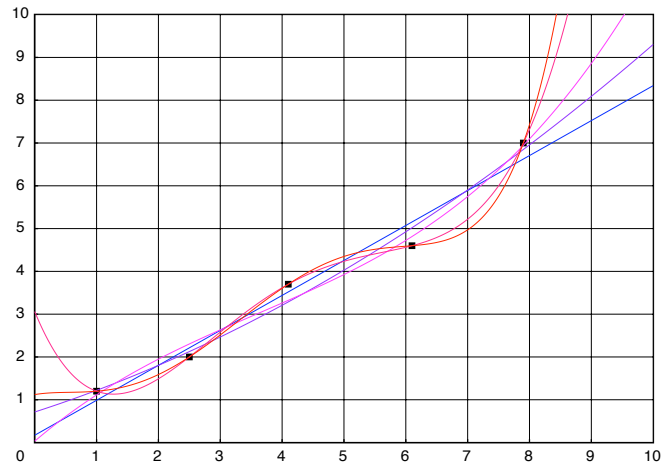
Consider the following set of five points:

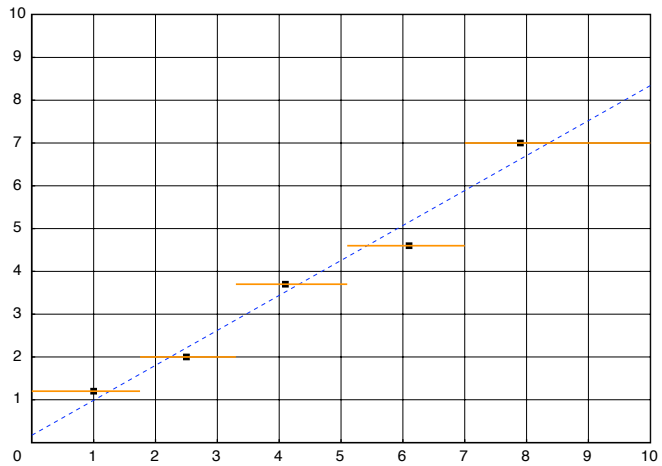
x	y
1.0	1.2
2.5	2.0
4.1	3.7
6.1	4.6
7.9	7.0

We want to estimate y by means of a polynomial in x. Figure 3.2 (left) shows the result for degrees of 1 to 5 using ☞*linear regression*, which will be explained in Chapter 7. The top two degrees fit the given points exactly (in general, any set of n points can be fitted by a polynomial of degree no more than $n-1$), but they differ considerably at the extreme ends: e.g., the polynomial of degree 4 leads to a decreasing trend from $x=0$ to $x=1$, which is not really justified by the data.

Figure 3.2, p.92

Fitting polynomials to data





(left) Polynomials of different degree fitted to a set of five points. From bottom to top in the top right-hand corner: degree 1 (straight line), degree 2 (parabola), degree 3, degree 4 (which is the lowest degree able to fit the points exactly), degree 5. **(right)** A piecewise constant function learned by a grouping model; the dotted reference line is the linear function from the left figure.

Overfitting again

An n-degree polynomial has $n+1$ parameters: e.g., a straight line $y = a \cdot x + b$ has two parameters, and the polynomial of degree 4 that fits the five points exactly has five parameters.

A piecewise constant model with n segments has $2n-1$ parameters: n y-values and $n-1$ x-values where the 'jumps' occur.

So the models that are able to fit the points exactly are the models with more parameters.

A rule of thumb is that, to avoid overfitting, the number of parameters estimated from the data must be considerably less than the number of data points.

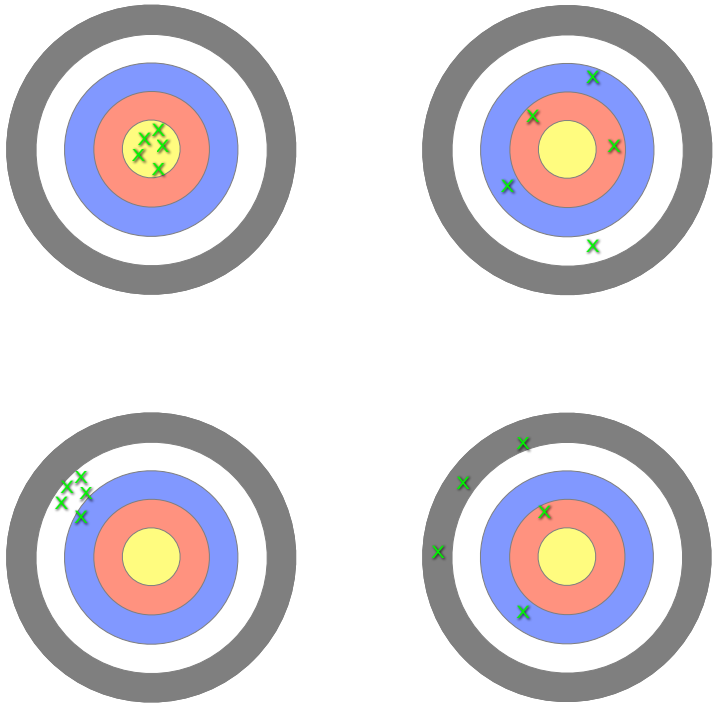
★ Bias and variance I

If we underestimate the number of parameters of the model, we will not be able to decrease the loss to zero, regardless of how much training data we have.

On the other hand, with a larger number of parameters the model will be more dependent on the training sample, and small variations in the training sample can result in a considerably different model.

This is sometimes called the *bias–variance dilemma*: a low-complexity model suffers less from variability due to random variations in the training data, but may introduce a systematic bias that even large amounts of training data can't resolve; on the other hand, a high-complexity model eliminates such bias but can suffer non-systematic errors due to variance.

★ Bias and variance



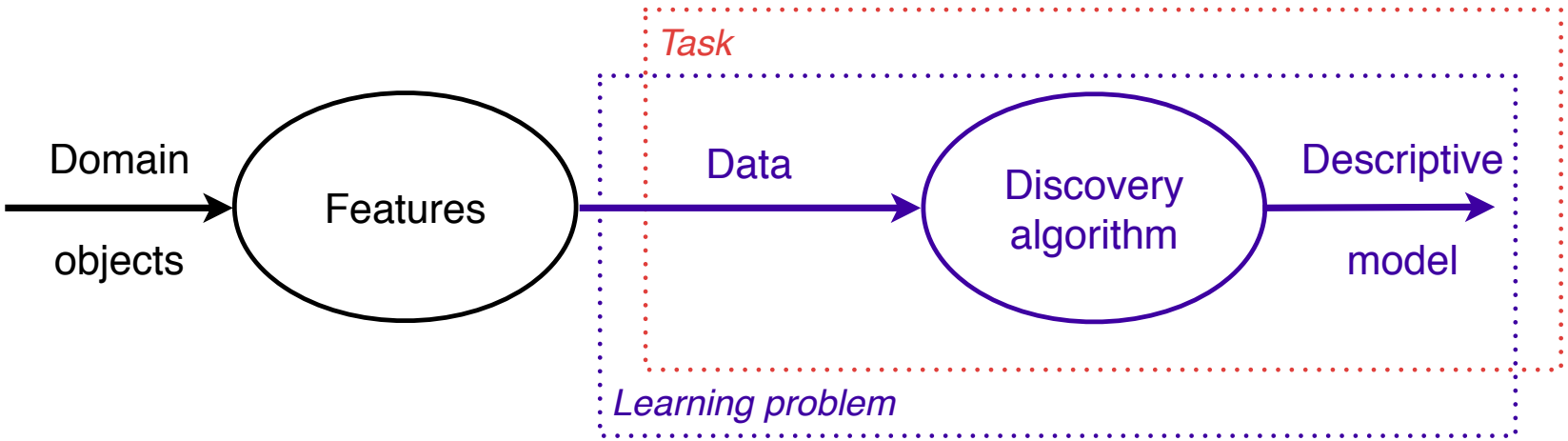
A dartboard metaphor illustrating the concepts of bias and variance. Each dartboard corresponds to a different learning algorithm, and each dart signifies a different training sample. The top row learning algorithms exhibit low bias, staying close to the bull's eye (the true function value for a particular x) on average, while the ones on the bottom row have high bias. The left column shows low variance and the right column high variance.

Unsupervised and descriptive learning

	Predictive model	*Descriptive model*
Supervised learning	classification, regression	**subgroup discovery**
Unsupervised learning	**predictive clustering**	**descriptive clustering**, **association rule discovery**

The learning settings indicated in **bold** are introduced in the remainder of this chapter.

Descriptive learning



In descriptive learning the task and learning problem coincide: we do not have a separate training set, and the task is to produce a descriptive model of the data.

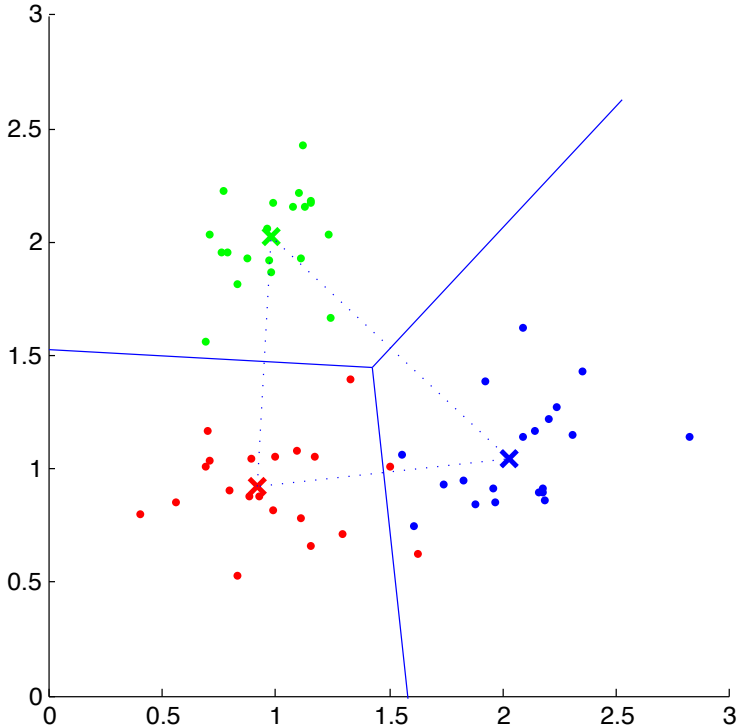
Predictive and descriptive clustering

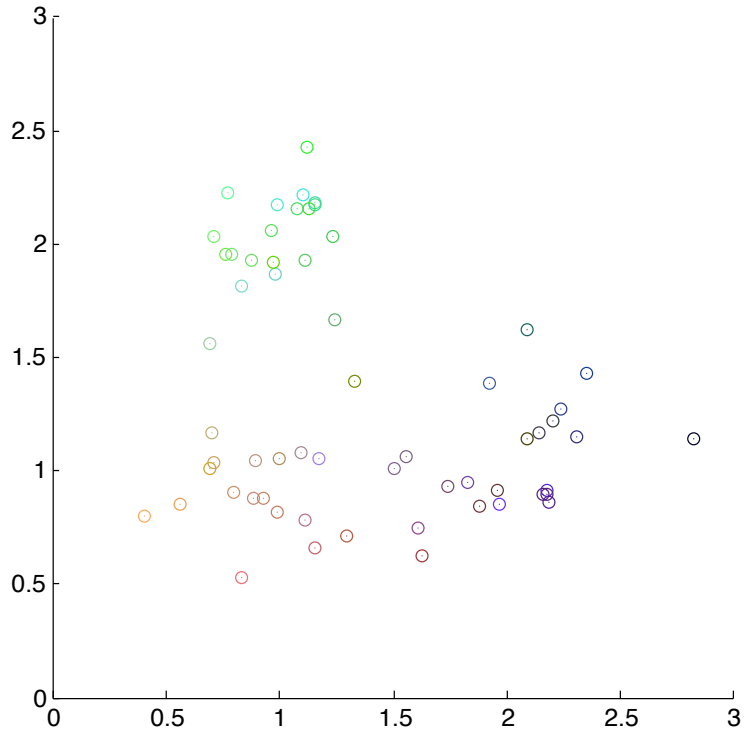
One way to understand clustering is as learning a new labelling function from unlabelled data. So we could define a 'clusterer' in the same way as a classifier, namely as a mapping $\hat{q} : \mathscr{X} \to \mathscr{C}$, where $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ is a set of new labels. This corresponds to a *predictive* view of clustering, as the domain of the mapping is the entire instance space, and hence it generalises to unseen instances.

A *descriptive* clustering model learned from given data $D \subseteq \mathscr{X}$ would be a mapping $\hat{q} : D \to \mathscr{C}$ whose domain is D rather than $\mathscr{X}$. In either case the labels have no intrinsic meaning, other than to express whether two instances belong to the same cluster. So an alternative way to define a clusterer is as an equivalence relation $\hat{q} \subseteq \mathscr{X} \times \mathscr{X}$ or $\hat{q} \subseteq D \times D$ or, equivalently, as a partition of $\mathscr{X}$ or D.

Figure 3.5, p.98

Predictive clustering





(left) An example of a predictive clustering. The coloured dots were sampled from three bivariate Gaussians centred at $(1,1)$, $(1,2)$ and $(2,1)$. The crosses and solid lines are the cluster exemplars and cluster boundaries found by 3-means. **(right)** A soft clustering of the same data found by matrix decomposition.

Evaluating clusterings

Suppose we have five test instances that we think should be clustered as $\{e1, e2\}, \{e3, e4, e5\}$. So out of the $5 \cdot 4 = 20$ possible pairs, 4 are considered 'must-link' pairs and the other 16 as 'must-not-link' pairs. The clustering to be evaluated clusters these as $\{e1, e2, e3\}, \{e4, e5\}$ – so two of the must-link pairs are indeed clustered together ($e1$–$e2$, $e4$–$e5$), the other two are not ($e3$–$e4$, $e3$–$e5$), and so on.
We can tabulate this as follows:

	Are together	*Are not together*	
Should be together	**2**	**2**	4
Should not be together	**2**	**14**	16
	4	16	20

We can now treat this as a two-by-two contingency table, and evaluate it accordingly. For instance, we can take the proportion of pairs on the 'good' diagonal, which is $16/20 = 0.8$. In classification we would call this accuracy, but in the clustering context this is known as the *Rand index*.

Subgroup discovery

Imagine you want to market the new version of a successful product. You have a database of people who have been sent information about the previous version, containing all kinds of demographic, economic and social information about those people, as well as whether or not they purchased the product.

- If you were to build a classifier or ranker to find the most likely customers for your product, it is unlikely to outperform the majority class classifier (typically, relatively few people will have bought the product).
- However, what you are really interested in is finding reasonably sized subsets of people with a proportion of customers that is significantly higher than in the overall population. You can then target those people in your marketing campaign, ignoring the rest of your database.

Association rule discovery

Associations are things that usually occur together. For example, in market basket analysis we are interested in items frequently bought together. An example of an association rule is ·**if** beer **then** crisps·, stating that customers who buy beer tend to also buy crisps.

- In a motorway service station most clients will buy petrol. This means that there will be many frequent item sets involving petrol, such as $\{\text{newspaper}, \text{petrol}\}$.
- This might suggest the construction of an association rule ·**if** newspaper **then** petrol· – however, this is predictable given that $\{\text{petrol}\}$ is already a frequent item set (and clearly at least as frequent as $\{\text{newspaper}, \text{petrol}\}$).
- Of more interest would be the converse rule ·**if** petrol **then** newspaper· which expresses that a considerable proportion of the people buying petrol also buy a newspaper.

Summary

What Did We Learn?

- Intuitions of knowledge acquisition in the context of KBS
- Some Frameworks/Models for Knowledge Acquisition
- Approaches/Example for (Semi-)Automatic KA
- Overview/Intro/Basics: Machine Learning